LEVEL 2: Python for Data Manipulation and Analysis

Day 1

COURSE OUTLINE

- Introduction to Python Programming
- Python Environments
- Data Types and Operations
- Python Data Structures
- Control Flow Statements in Python
- Python Functions
- Python Classes
- NumPy
- Pandas
- Matplotlib

Learning Objectives

- Write basic Python programs and understand Python environments
- Use core data types and data structures effectively
- Control program flow using conditions and loops
- Create reusable code with functions and classes
- Perform numerical computing using NumPy
- Analyze and manipulate data using Pandas
- Visualize data clearly using Matplotlib

Python Programming

Introduction

Python Programming

- A high-level, interpreted programming language
- Known for simple, readable syntax
- Enables fast development and high productivity
- Supports modular and reusable code
- Free, open-source, and works across platforms
- Widely used in software development, data science, AI, and automation

Python Programming

Key Features of Python

Python Programming

Why Python?

Python Programming

Why Python Is Ideal for AI & Machine Learning?

- Rich AI/ML libraries

Python Programming

Why Python Is Ideal for AI & Machine Learning

- Strong global community

PyPI

Python Programming

Why Python Is Ideal for AI & Machine Learning

- High Performance

CUDA

Python Programming

Python Environments

Python Environment

- A Python environment provides an interface for writing and executing Python code, allowing the computer to interpret and run instructions.

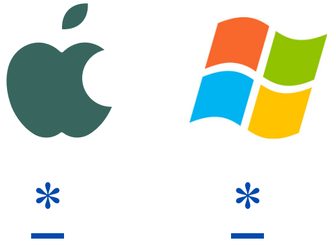
- Google Colab
  - *

Google Colab is a free, cloud-based Python environment that runs in your browser, with no installation required.

Python Environment

- A Python environment provides an interface for writing and executing Python code, allowing the computer to interpret and run instructions.

  - Jupyter Notebook

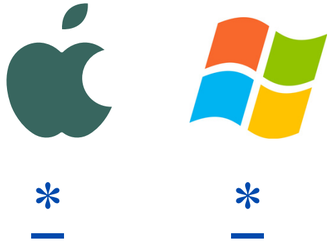


Jupyter Notebook is an interactive, web-based computational environment that enables users to create and share documents containing executable code, narrative text, mathematical equations, and visualizations.

Python Environment

- A Python environment provides an interface for writing and executing Python code, enabling the computer to interpret and run the instructions.

- VS Code



VS Code is a code editor used to write, run, and debug Python programs.

Python Environment

- A Python environment provides an interface for writing and executing Python code, allowing the computer to interpret and run instructions.

  - Other Python Environments

vertex.ai

Python Programming

Data Types and Operations

Data Types and Operations

Data Types

Category	Data Type	Description	Example
Numeric	int	Whole numbers	10, -3
Numeric	float	Decimal numbers	3.14, -0.5
Numeric	complex	Numbers with real & imaginary parts	2+3j
Text	str	Text (characters)	"Hello"
Boolean	bool	True or False	True, False

Data Types and Operations

Data Types

Category	Data Type	Description	Example
Sequence	list	Ordered, changeable collection	[1, 2, 3]
Sequence	tuple	Ordered, unchangeable collection	(1, 2, 3)
Set	set	Unordered, unique elements	{1, 2, 3}
Mapping	dict	Key–value pairs	{"a": 1}
None	NoneType	Represents no value	None

Data Types and Operations

Arithmetic Operators

Operator	Name	Description	Example	Result
+	Addition	Adds two values	5 + 3	8
-	Subtraction	Subtracts one value from another	7 3	5
*	Multiplication	Multiplies two values	4 * 3	12
/	Division	Divides two values	7 / 3	2.5
//	Floor Division	Divides and rounds down	10 // 4	2
%	Modulus	Returns remainder	10 % 4	2
**	Exponentiation	Raises to a power	2 ** 3	8

Data Types and Operations

Comparison Operators

Operator	Description	Example	Result
==	Equal to	5 == 5	TRUE
!=	Not equal	5 != 3	TRUE
>	Greater than	7 > 4	TRUE
<	Less than	3 < 8	TRUE
>=	Greater or equal	5 >= 5	TRUE
<=	Less or equal	2 <= 6	TRUE

Data Types and Operations

Logical Operators

Operator	Name	Example	Result
and	Logical AND	(5 > 3) and (2 < 4)	TRUE
or	Logical OR	(5 > 3) or (2 > 4)	TRUE
not	Logical NOT	not (5 > 3)	FALSE

Data Types and Operations

Assignment Operators

Operator	Description	Example	Equivalent
=	Assign	x = 5	x = 5
+=	Add & assign	x += 3	x = x + 3
-=	Subtract & assign	x -= 2	x = x - 2
*=	Multiply & assign	x *= 4	x = x * 4
/=	Divide & assign	x /= 2	x = x / 2

Data Types and Operations

Membership Operators

Operator	Description	Example	Result
in	Value exists	3 in [1,2,3]	TRUE
not in	Value not exists	5 not in [1,2,3]	TRUE

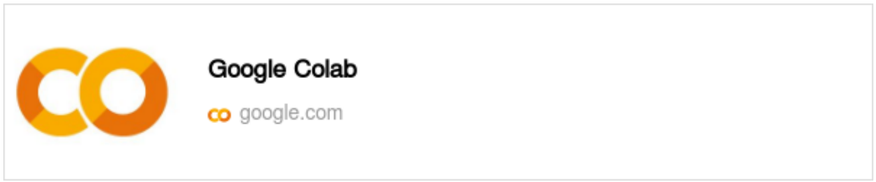
Data Types and Operations

Identity Operators

Operator	Description	Example	Result
is	Same object	a is b	True/False
is not	Different object	a is not b	True/False

Data Types and Operations

- 01-numbers.ipynb



Data Types and Operations

- A string is a sequence of characters
- Used to represent text
- Defined using quotes
  - Double quotes " "
  - Single quotes ' '

```
city = "Riyadh"
country = 'Saudi Arabia'
```

Data Types and Operations

- Each character has a position (index)
- Indexing starts from 0

```
text = "Python"
text[0]   # 'P'
text[3]   # 'h'
```

- String Slicing

```
name = "Mohammed"
name[0:4]   # 'Moha'
name[4:]    # 'mmed'
```

Data Types and Operations

String Concatenation

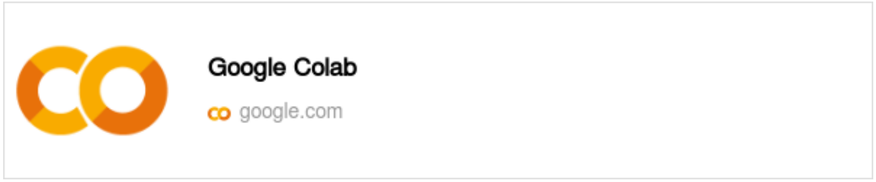
```
first = "Mohammed"
last = "Ali"
full_name = first + " " + last
```

String Repetition

```
name = "Riyadh"
name * 3
```

Data Types and Operations

- 02-Strings.ipynb



Python Programming

Python Data Structures

Python Data Structures

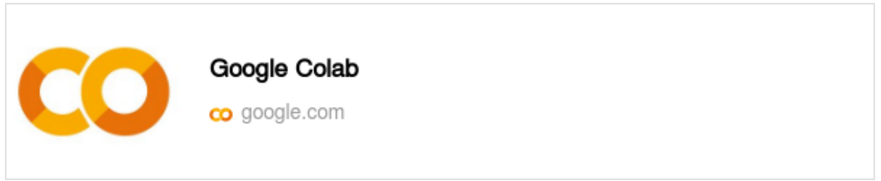
Lists

- A list is an ordered collection of items
- Lists can store multiple values
- Lists are mutable (can be changed)
- Lists are created using square brackets [].

```
cities = ["Riyadh", "Jeddah", "Dammam"]
```

Python Data Structures

[03-Lists.ipynb]



Python Data Structures

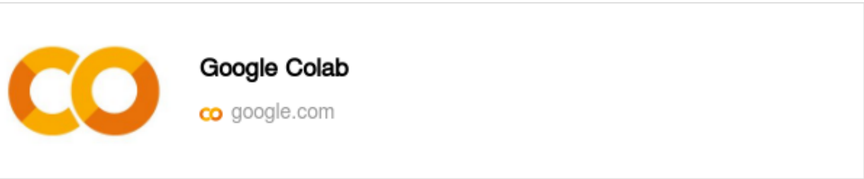
Dictionaries

- A dictionary stores data as key–value pairs
- Each key maps to a value
- Dictionaries are mutable
- Dictionaries use curly braces {}

```
student = {
    "name": "Mohammed",
    "age": 21,
    "city": "Riyadh"
}
```

Python Data Structures

- 04-Dictionaries.ipynb



Python Data Structures

Tuple

- A tuple is an ordered collection of items
- Tuples are immutable (cannot be changed)
- Used for fixed data

```
person = ("Mohammed", 21, "Riyadh")
```

Python Data Structures

- 05-Tuples.ipynb

Google Colab

co google.com

Python Data Structures

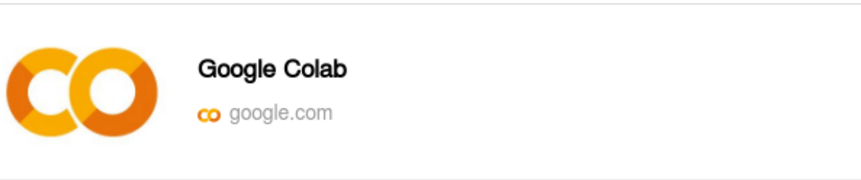
Set

- A set is an unordered collection of unique elements
- Duplicate values are automatically removed
- Uses curly braces {}
- Sets are mutable

```
numbers = {1, 2, 3}
empty_set = set()   # not {}
```

Python Data Structures

06-Sets.ipynb



Python Data Structures

Data Structure	Syntax	Ordered	Mutable	Allows Duplicates	Access Method	Common Use Case
List	[ ]	Yes	Yes	Yes	Index	Store and modify collections of items
Tuple	( )	Yes	No	Yes	Index	Store fixed, unchangeable data
Set	{ }	No	Yes	No	Membership (in)	Remove duplicates, fast lookup
Dictionary	{key: value}	Yes*	Yes	(keys)	Key	Store structured data (records)

* Dictionaries are ordered from Python 3.7+

Python Programming

Control Flow Statements in Python

Control Flow Statements in Python

if

- if is a conditional statement
- It is used to make decisions in code
- Code runs only if the condition is True

```
score = 75
if score >= 60:
    print("Pass")
```

Control Flow Statements in Python

- if with else

```
city = "Riyadh"

if city == "Riyadh":
    print("Welcome to the capital")
else:
    print("Welcome")
```

Control Flow Statements in Python

- if, elif, else

```
score = 85

if score >= 90:
    print("Excellent")
elif score >= 70:
    print("Good")
else:
    print("Needs improvement")
```

Control Flow Statements in Python

Indentation Is Important

- Python uses indentation, not { }

```
if True:
    print("Correct")

# if True:
# print("Error")
```

Common Mistakes

Using = instead of ==

Forgetting indentation

Missing **:** at the end of if

Control Flow Statements in Python

for Loop

- A for loop is used to repeat actions
- It iterates over a sequence (list, string, range, etc.)
- Executes the block once per item

```
for i in range(5):
    print(i)
```

1,2,3,4,5

Control Flow Statements in Python

- for with if

```
scores = [95, 72, 40]

for score in scores:
    if score >= 60:
        print(score, "Pass")
    else:
        print(score, "Fail")
```

95 Pass
72 Pass
40 Fail

Control Flow Statements in Python

while loops

- A while loop repeats code as long as a condition is True
- The condition is checked before each iteration

```
count = 0

while count < 5:
    print(count)
    count += 1
```

Common Mistakes

- Forgetting to update the condition variable
- Infinite loops
- Missing indentation

Control Flow Statements in Python

while loops

- Infinite Loop ∞

```
while True:
 print("This runs forever")
```

- Using break

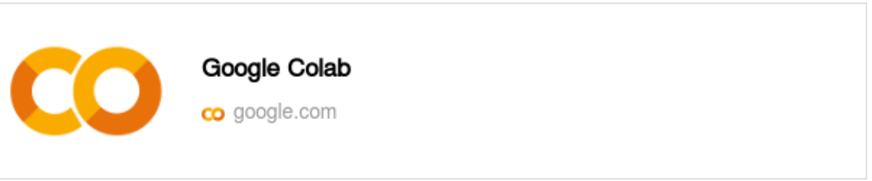
```
while True:
    user_input = input("Type 'exit' to stop: ")
    if user_input == "exit":
        break
```

Control Flow Statements in Python

Statement	Purpose	When to Use	Typical Example
if	Decision making	When you need to choose between actions	Pass / Fail
for	Iteration	When number of iterations is known	Loop through a list
while	Conditional looping	When repetitions depend on a condition	User input loop

Control Flow Statements in Python

- 07-Control Flow.ipynb



Python Programming

Python Functions

Python Functions

Functions

- def → defines a function
- Function name follows naming rules
- Code block is indented

- Function Syntax

```
def greet():
    print("Hello!")
```

- Calling a Function

```
great()
```

* Function runs only when called

Python Functions

Function with Parameters

- Function Syntax

```
def greet(name):
    print("Hello", name)
```

- Calling a Function

```
greet("Mohammed")
```

Multiple Parameters

- Function Syntax

```
def add(a, b):
    print(a + b)
```

- Calling a Function

```
add(3, 5)
```

Python Functions

Returning Values

- Function Syntax

```
def add(a, b):
    return a + b
```

- Calling a Function

```
result = add(4, 6)
print(result)
```

*Return sends a value back

print	return
Displays output	Sends value back
Used for display	Used for logic
Returns None	Returns data

Python Programming

Python Classes

Python Classes

Class

- A class is a blueprint for creating objects
- It groups data (attributes) and functions (methods)
- Used to model real-world entities

- Class

```
class Student:
    name = "Mohammed"
    age = 21
```

- Object (Instance)

```
s1 = Student()
print(s1.name)
print(s1.age)
```

Python Classes

The __init__ Method (Constructor)

- Class

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

- Object (Instance)

```
s1 = Student("Ali", 20)
s2 = Student("Sara", 22)
```

- Runs automatically when the object is created
- self refers to the current object

Python Classes

- Instance Attributes

```
print(s1.name)
print(s1.age)
```

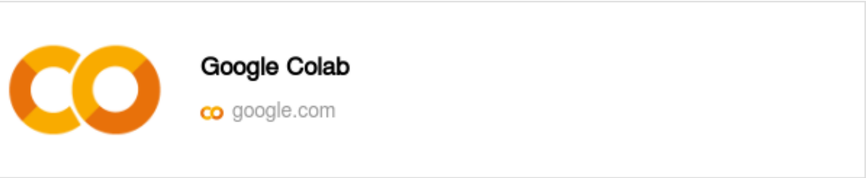
- Methods (Functions Inside a Class)

```
class Student:
    def greet(self):
        print("Hello, my name is", self.name)
```

```
s1.greet()
```

Control Flow Statements in Python

- 08-Functions_and_Classes.ipynb



Python Programming

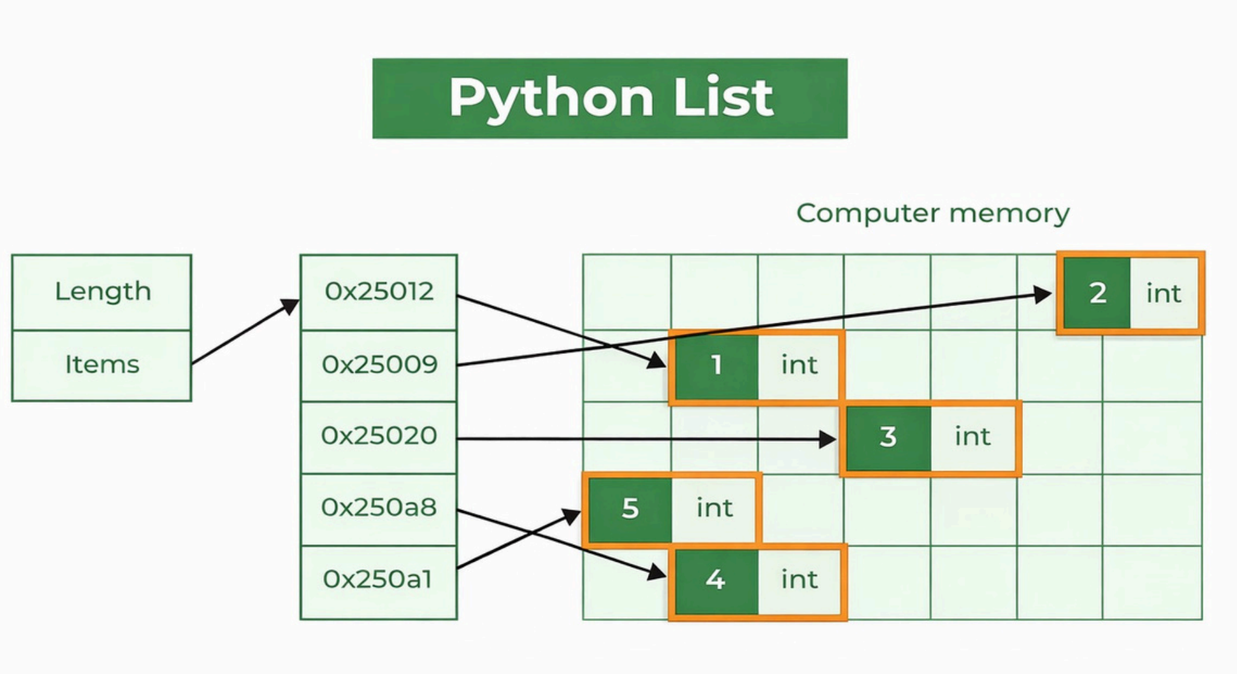
NumPy

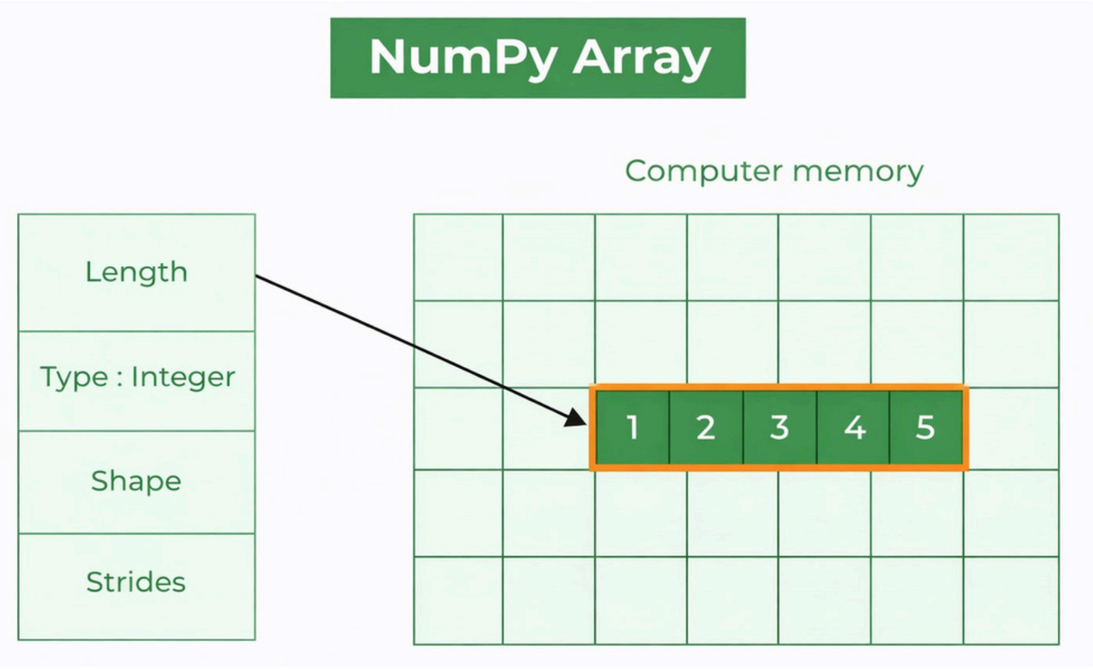
NumPy

- NumPy is a Python library for numerical computing
- It provides fast, efficient array operations
- Widely used in data science, ML, and scientific computing

NumPy

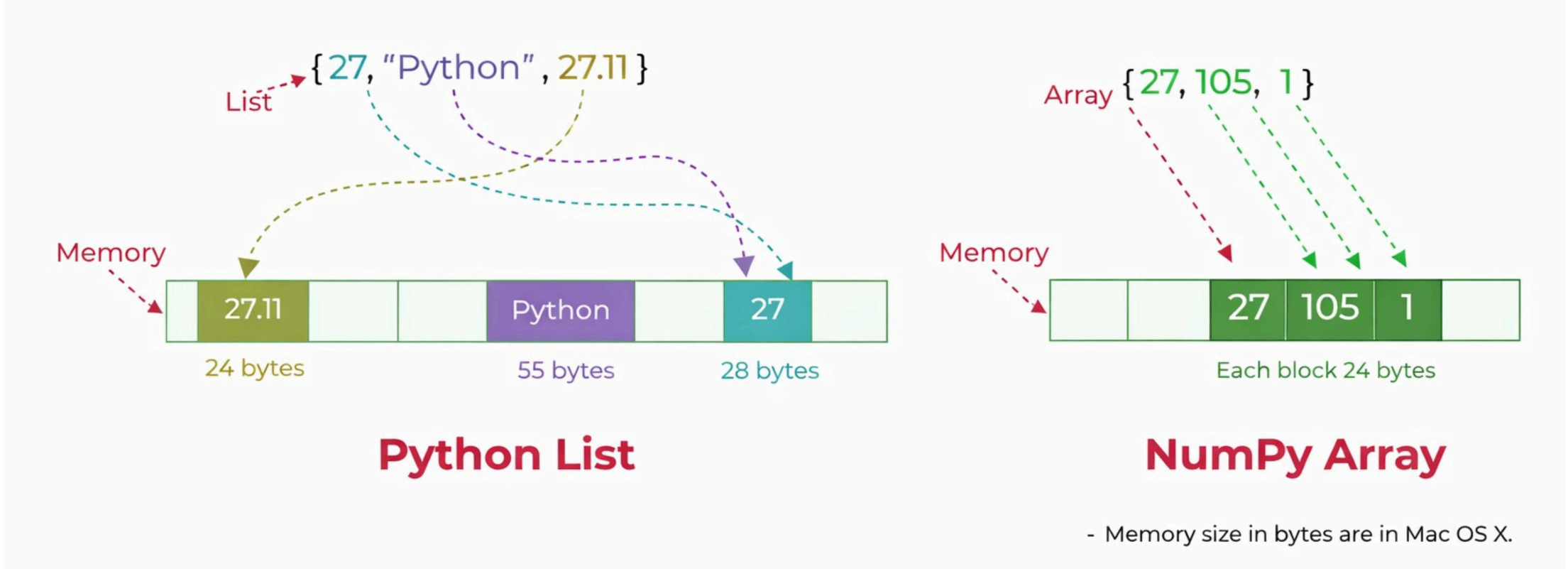
- Python Lists VS Numpy Arrays





NumPy

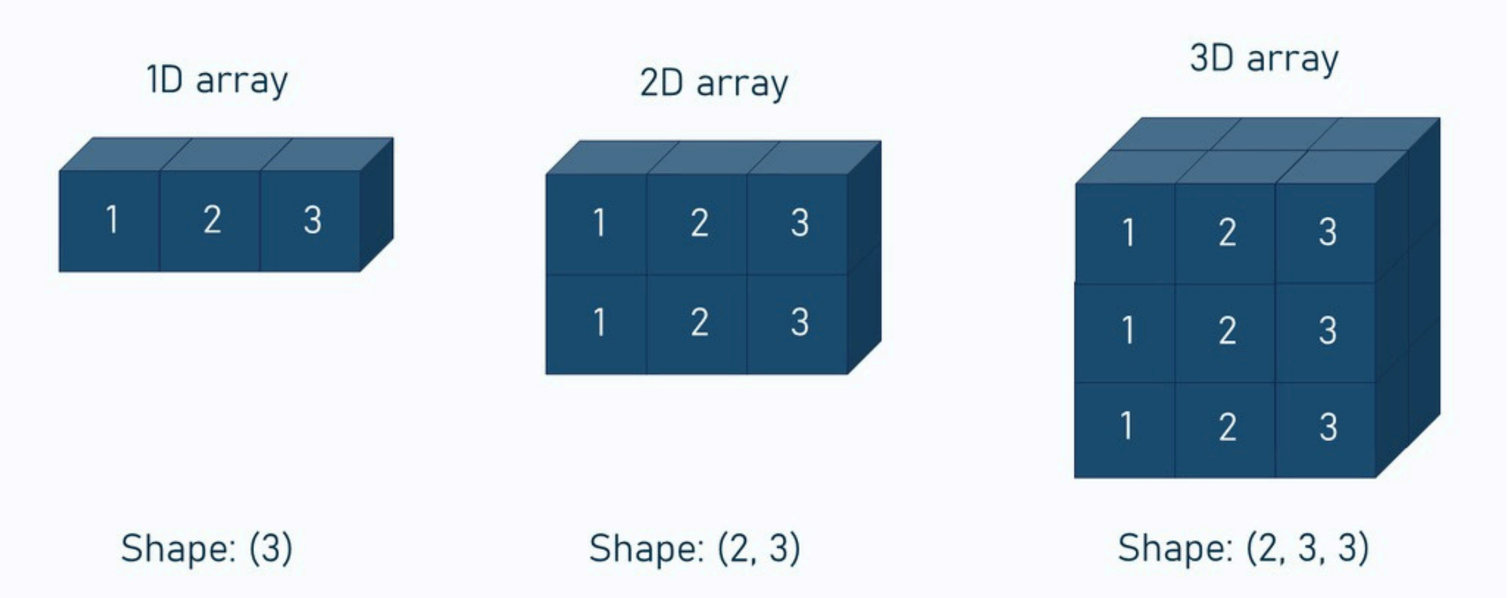
- Python Lists VS Numpy Arrays



NumPy

- NumPy Arrays

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



NumPy

- Arrays filled with values

```
np.zeros(5)
np.ones((2, 3))
```

- Random Arrays

```
np.random.rand(3)
np.random.randn(2, 2)
```

- Sequences of numbers

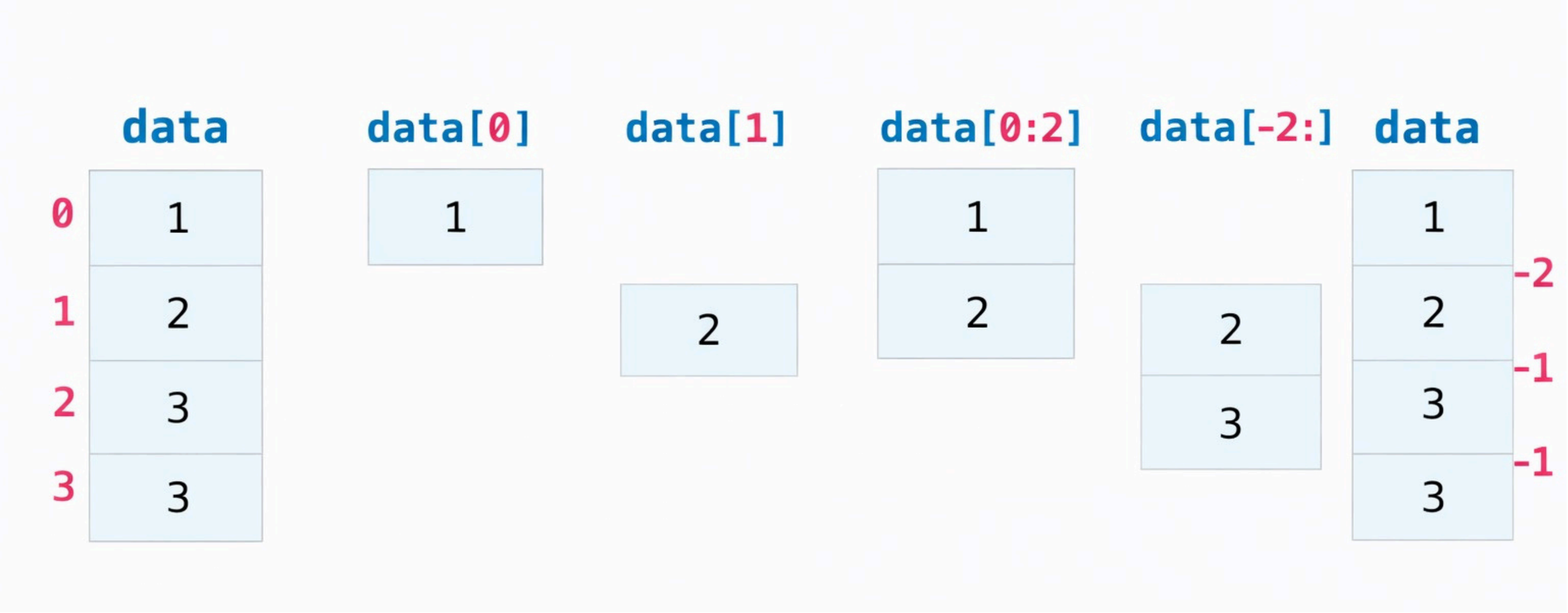
```
np.arange(0, 10, 2)
np.linspace(0, 1, 5)
```

- dtype

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

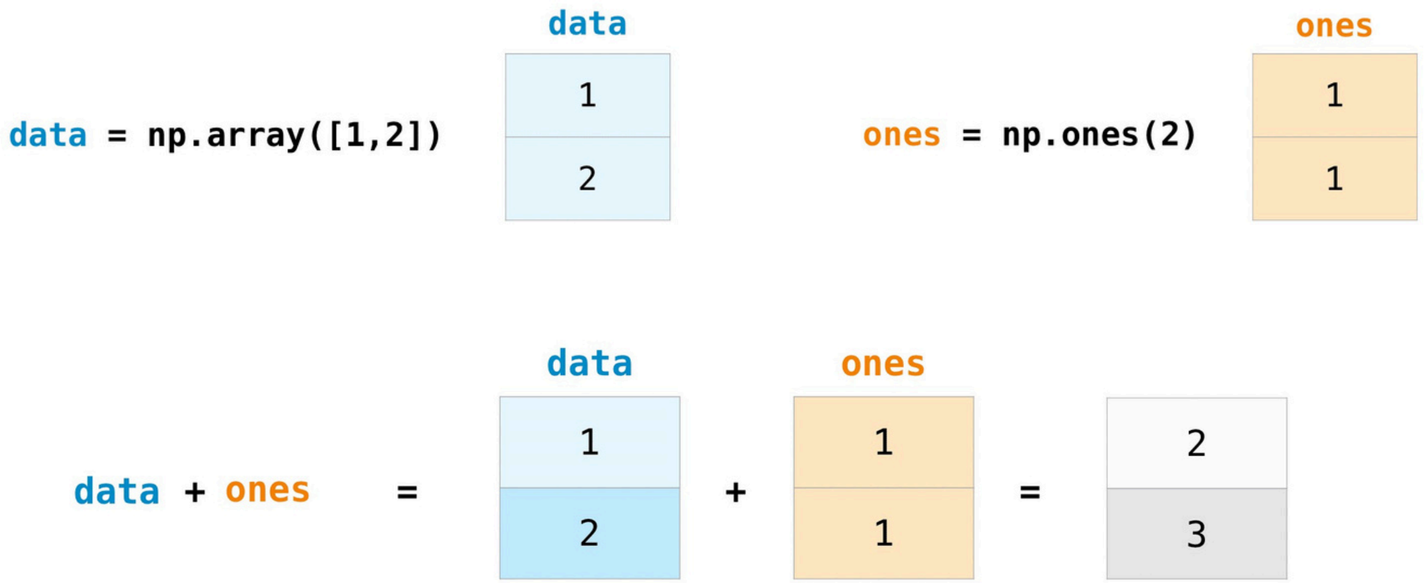
NumPy

- Indexing and slicing



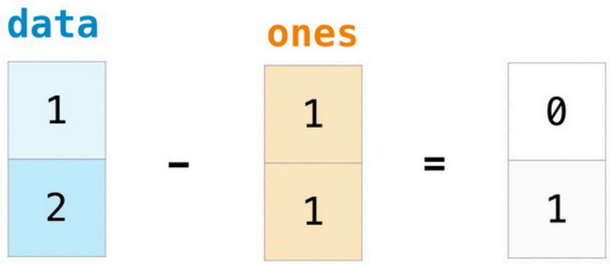
NumPy

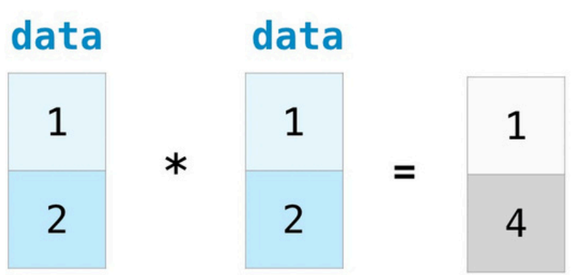
Array operations

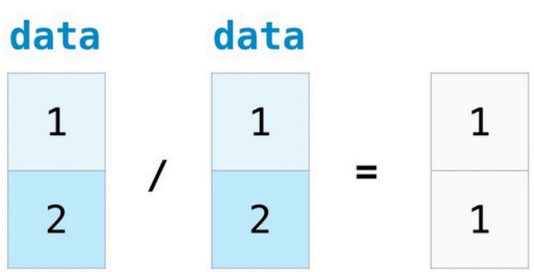


NumPy

Array operations

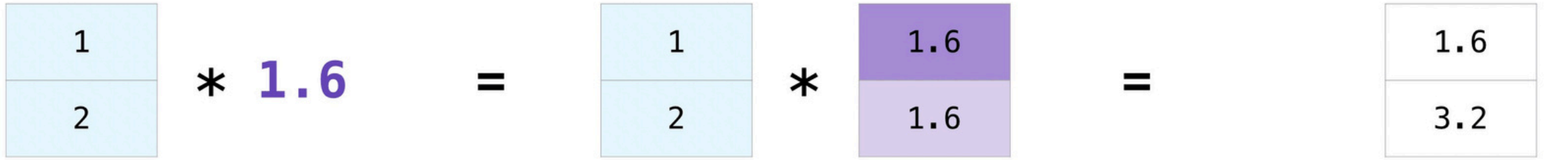




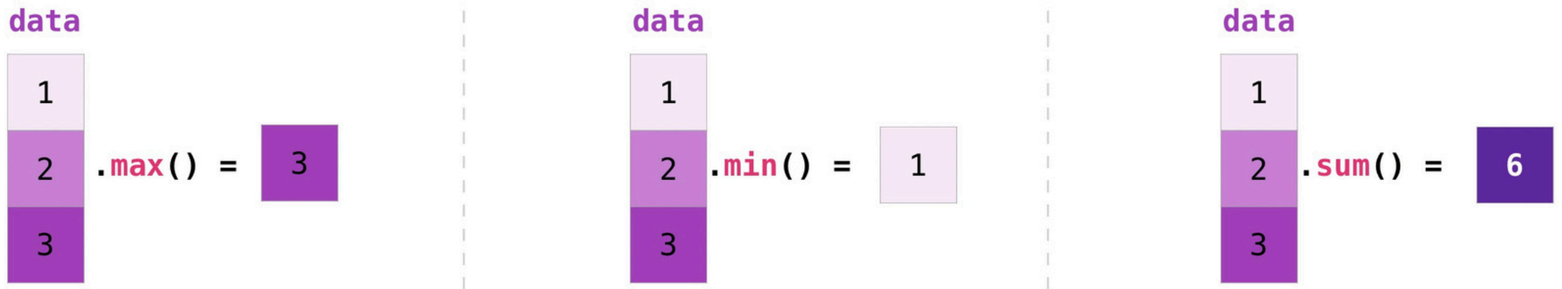


NumPy

Broadcasting



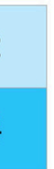
maximum, minimum, sum



NumPy

Matrices

```
np.array([[1,2],[3,4],[5,6]])
```



1	2
3	4
5	6

data

	0	1
0	1	2
1	3	4
2	5	6

data[0,1]

	0	1
0	1	2
1	3	4
2	5	6

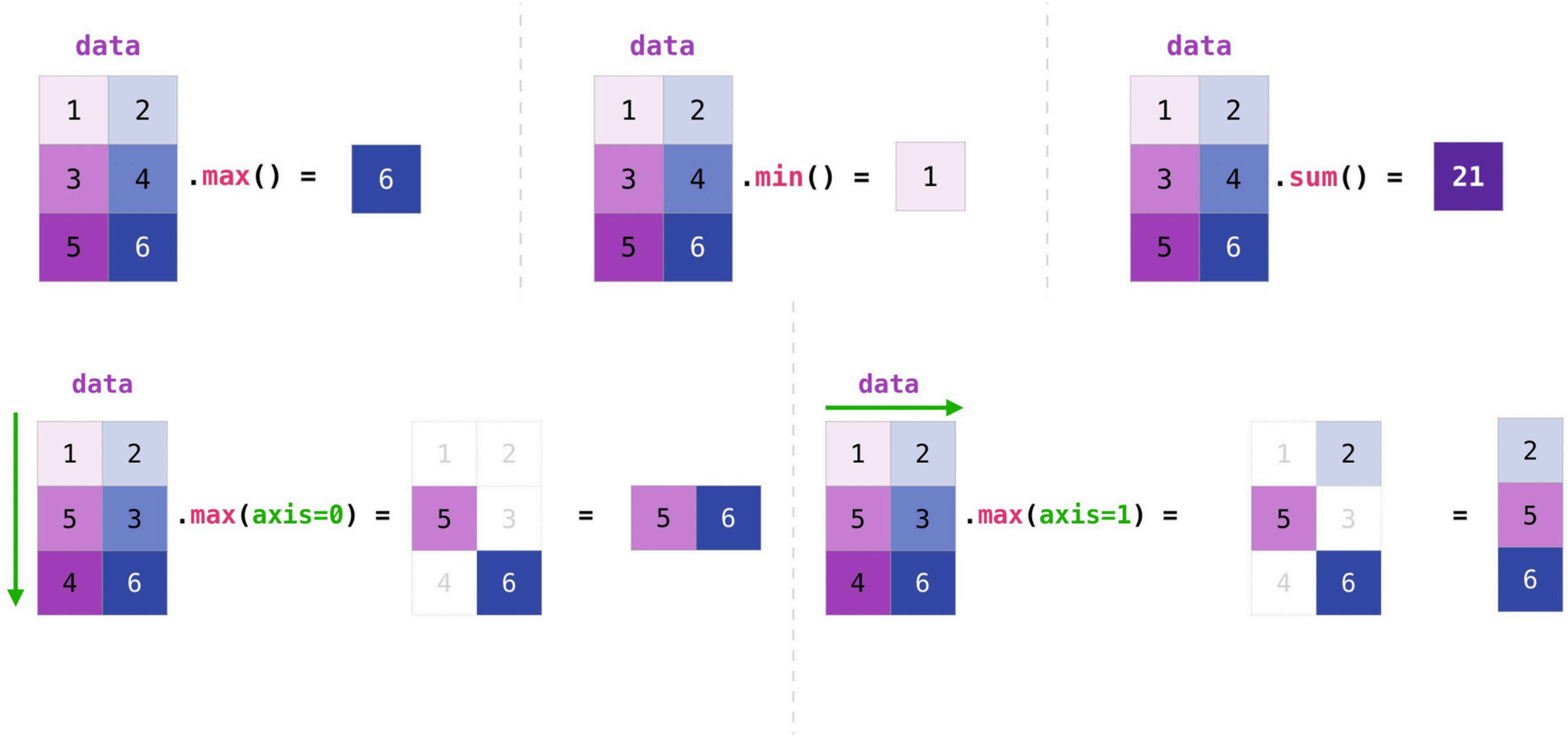
data[1:3]

	0	1
0	1	2
1	3	4
2	5	6

data[0:2,0]

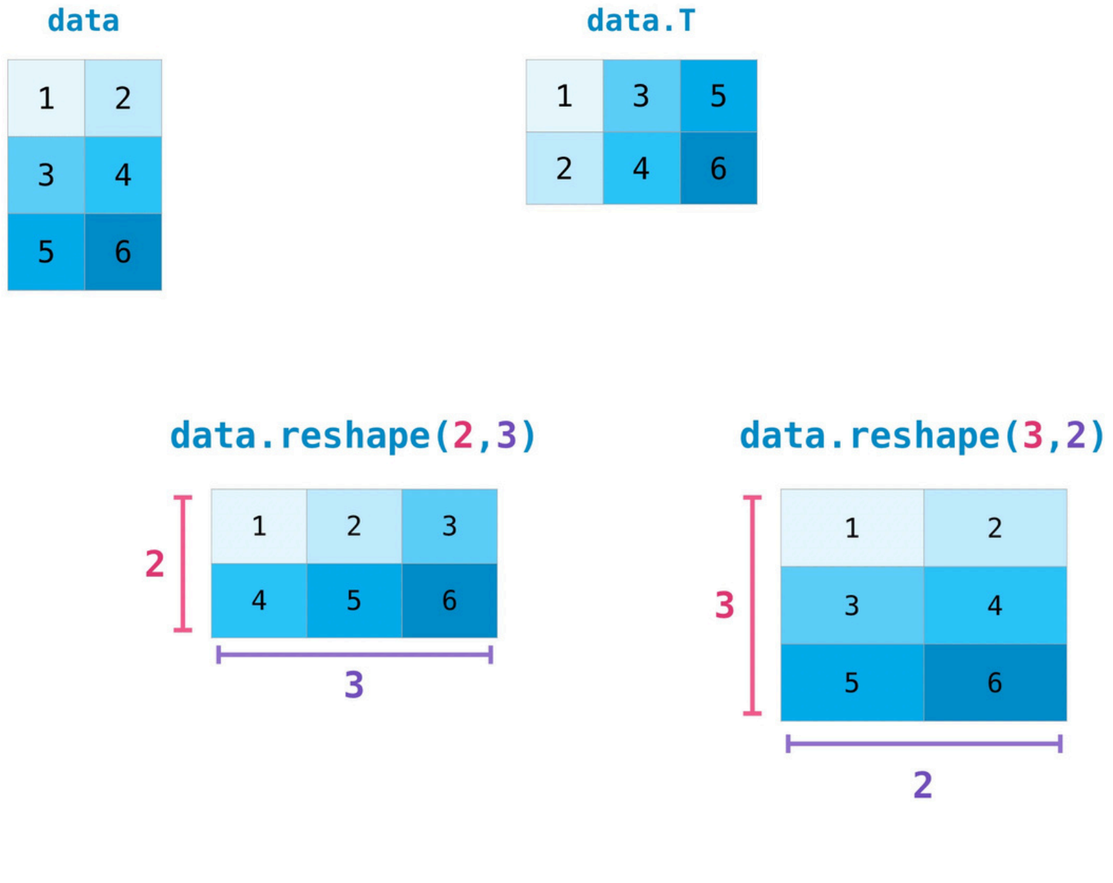
	0	1
0	1	2
1	3	4
2	5	6

Aggregation Functions



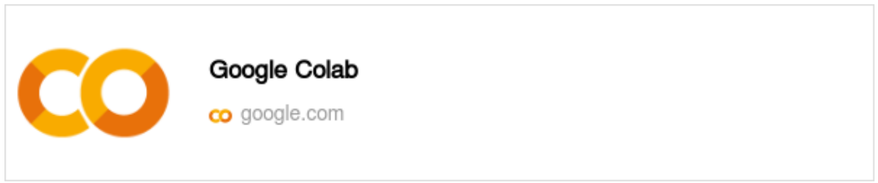
NumPy

Transposing and reshaping a matrix



NumPy

- 09-NumPy.ipynb



Python Programming

Pandas

Pandas

- Pandas is a Python library for data manipulation and analysis

  - Built on top of NumPy
  - Designed to work with tabular and labeled data
  - Widely used in data science and machine learning

Pandas

Pandas Data Structures

- Series → 1D labeled array
- DataFrame → 2D table with rows and columns
- Index provides meaningful labels

Series 1

INDEX	DATA
0	A
1	B
2	C
3	D
4	E
5	F

Series 2

INDEX	DATA
A	1
B	2
C	3
D	4
E	5
F	6

Series 3

INDEX	DATA
0	[1, 2]
1	A
2	1
3	(4, 5)
4	{"a": 1}
5	6

Series 4

INDEX	DATA
Jan-18	11
Feb-18	23
Mar-18	43
Apr-18	21
May-18	17
Jun-18	6

Pandas

Creating a DataFrame

```
pd.DataFrame({"A": [1, 2], "B": [3, 4]})
```

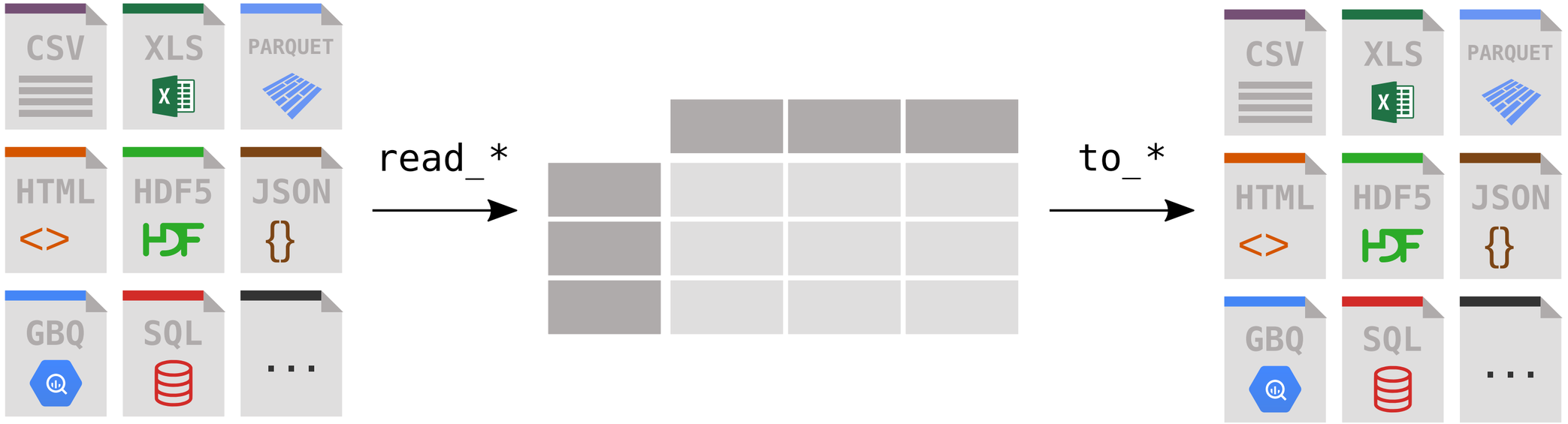
```
pd.DataFrame(np.array([[1, 2], [3, 4]]))
```

```
pd.DataFrame([[1, 2], [3, 4]], columns=["A", "B"])
```

```
pd.DataFrame(np.array([[1, 2], [3, 4]]))
```

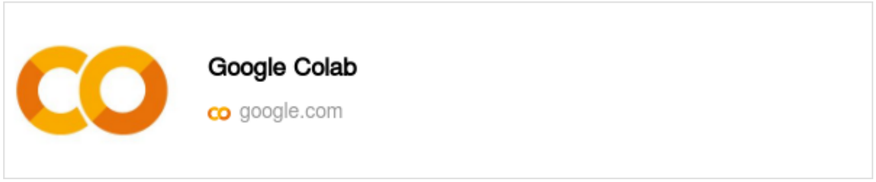
Pandas

Reading Data and Exporting Data



Pandas

- 10-Pandas.ipynb



Python Programming

Matplotlib

Matplotlib

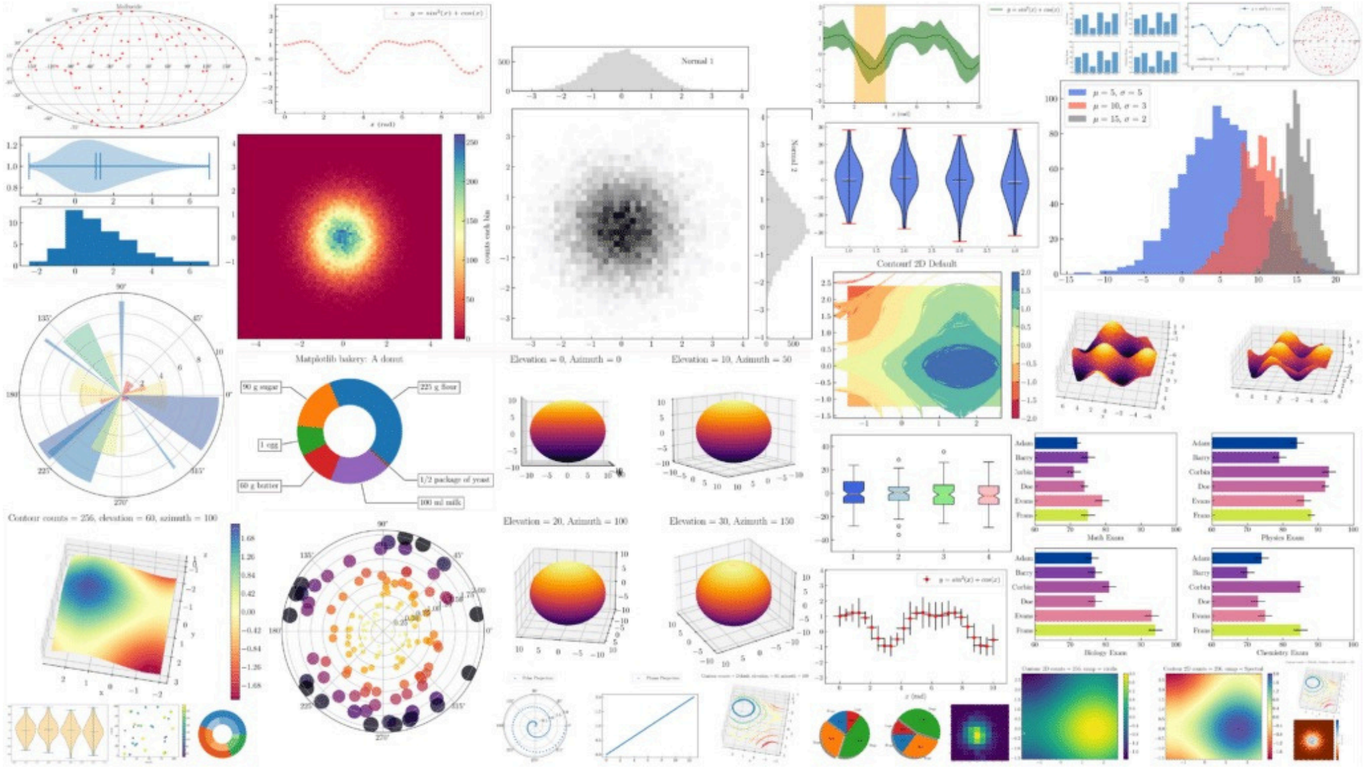
- Matplotlib is a Python library for data visualization
  - Used to create static, animated, and interactive plots
  - Works seamlessly with NumPy and Pandas
  - Foundation for many other plotting libraries

Matplotlib

Matplotlib is a Python library for data visualization

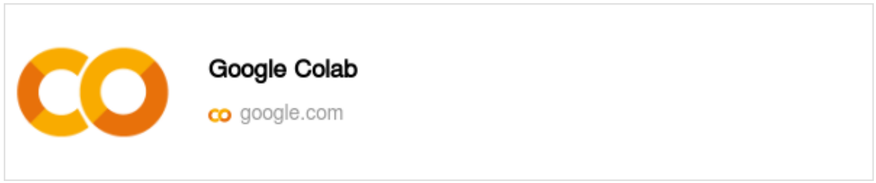
- Line Plot – trends over time or ordered data
- Scatter Plot – relationships between variables
- Bar Chart – compare categories
- Histogram – data distribution
- Box Plot – spread, median, outliers

- Pie Chart – proportions
- Area Plot – cumulative trends
- Heatmap – values across two dimensions
- Error Bar Plot – uncertainty and variation
- Subplots – multiple plots in one figure



Matplotlib

[11-Matplotlib.ipynb]




Q
A

Thank You