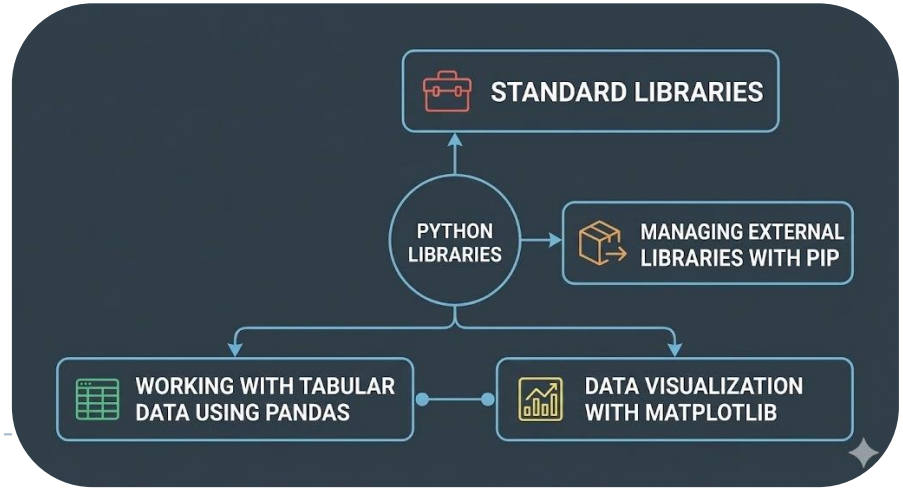
Computational Thinking

Lecture 15: Popular Python Libraries

University of Engineering and Technology
VIETNAM NATIONAL UNIVERSITY HANOI

Outline

- Introduction to Python libraries
- Standard libraries
- Managing external libraries with Pip
- Data visualization with Matplotlib
- Working with tabular data using Pandas



Introduction to Python Libraries

Python's Libraries

- Library: a collection of modules that contain functions, classes and methods
  - **math**: Mathematical operations
  - **datetime**: Date and time operations
  - **random**: Generate random numbers
  - **matplotlib**: Data visualization
  - **pandas**: Working with tabular data
- **Why using libraries!**
  - Code reusability and efficiency
  - Save vast amounts of development time
  - Access to specialized functionality
  - Industry-standard best practices

```
...
import math
...
x = 99
math.sqrt(x)
...
```

Python Library Types

Standard libraries

- These come standard with every Python installation. Ready to use immediately
  - math
  - random
  - datetime
  - …

External libraries

- These need to be installed separately using package managers like pip
  - pandas
  - matplotlib
  - numpy

Standard Libraries

Using math Library

Provides access to common mathematical functions and constants.

Essential constants

π

math.pi

The mathematical constant π (pi), approximately 3.1415.

Used in geometry for circles and spheres.

e

math.e

The mathematical constant e, approximately 2.71828.

Fundamental in calculus and natural logarithms.

Example: Circle Circumference

```
import math

radius = 5
circumference = 2*math.pi*radius
print(circumference)
```

Essential math Functions (1/2)

Precision Control: Rounding Functions

math.ceil(x)

Returns the smallest integer greater than or equal to `x` (rounds up).

```
print(math.ceil(4.1)) # Output: 5
```

math.floor(x)

Returns the largest integer less than or equal to `x` (rounds down).

```
print(math.floor(4.9)) # Output: 4
```

math.trunc(x)

Returns the integer part of `x`, truncating the decimal part.

```
print(math.trunc(4.9))    # Output: 4
print(math.trunc(-4.1))   # Output: -4
```

Essential math Functions (2/2)

Powers and Square Roots

math.pow(base, exp)

Returns base raised to the power of exp. The result is always a float.

```
print(math.pow(2, 3))# Output: 8.0
print(2**3)          # Output: 8
```

math.sqrt(x)

Returns the square root of x. The argument x must be non-negative.

```
print(math.sqrt(25)) # Output: 5.0
```

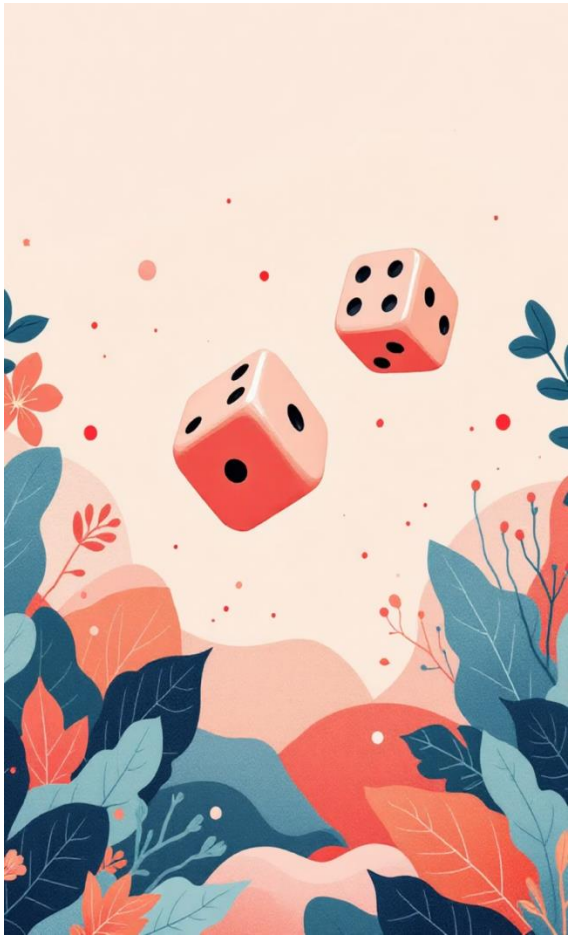
Application: Pythagorean Theorem

Calculate the hypotenuse of a right triangle using math.sqrt and math.pow.

```
a = 3
b = 4
hypotenuse = math.sqrt(math.pow(a, 2) + math.pow(b, 2))
print(f"Hypotenuse: {hypotenuse}") # Output: 5.0
```

Using random Library

- Why random: simulate uncertainty
  - Scientific simulations
  - Game
  - Cryptography



Example

```
import random

print(random.random()) # e.g., 0.95...

# returning an integer from a to b
print(random.randint(1, 10)) # e.g., 5
```

Making Random Selections

Random Choices: random.choice(sequence)

- Picking an element at random from a list, tuple, or string:

```
choices = ["Rock", "Paper",
"Scissors"]
choice =
random.choice(choices)
print(choice)
```

Shuffling Sequences: random.shuffle(list)

- Randomize the order of items in a list, performing an in-place shuffle.

```
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(my_list)
```

Using datetime Library

- datetime: managing and manipulating time in a human-readable format.

Example: Grabbing the present moment

```
import datetime
current_time = datetime.datetime.now()
print(current_time)
```

This object is typically displayed in a clear and standardized format:

```
YYYY-MM-DD HH:MM:SS.microseconds
```

Each component is easily accessible: year, month, day, hour, minute, second, and even microseconds.

Constructing Specific Timestamps

- Creating a specific date:

  datetime.date(year, month, day).

```
Ex: christmas = datetime.date(2023, 12, 25)
Returned: a Date object
```

- Creating a specific time:

  datetime.time(hour, minute, second).

```
Ex: noon = datetime.time(12, 0, 0)
Returned: a Time object
```

- Creating a combined datetime:

  datetime.datetime(year, month, day, hour, minute, second).

```
event = datetime.datetime(2024, 7, 4,
9, 30, 0)
Returned: a Datetime object
```

Performing Time Calculations with Timedelta

- The timedelta object represents a duration, allowing you to add or subtract time from datetime objects.
- Adding and subtracting durations:

```
from datetime import datetime, timedelta

today = datetime.now()
seven_days_later = today + timedelta(days=7)
print(f"Today: {today.date()}")
print(f"Deadline in 7 days:
{seven_days_later.date()}")
```

There are various units: days, seconds, minutes, hours, weeks.

Formatting Dates and Times with strftime

- Convert a datetime object into a string according to a specified format code:
  - %Y: Full year (e.g., 2023)
  - %m: Month as a zero-padded decimal number (01-12)
  - %d: Day of the month as a zero-padded decimal (01-31)
  - %H: Hour (24-hour clock) as a zero-padded decimal (00-23)
  - %M: Minute as a zero-padded decimal (00-59)
  - %S: Second as a zero-padded decimal (00-59)
  - %A: Weekday as locale's full name (e.g., Monday)

```
from datetime import datetime
today = datetime.now()
formatted_date = today.strftime("%m/%d/%Y")
print(formatted_date)
```

Managing External Libraries with PIP

PIP: The Python Package Installer

- Pip: Pip Installs Packages - the standard package-management system of Python.
  - Connects directly to the PyPI (Python Package Index), a massive repository where developers share their Python libraries.
  - PIP automatically handles dependencies, ensuring all necessary components are installed.
  - Install a new library:

```
pip install <library_name>
```

  - For example:
    - pip install pandas
    - pip install matplotlib

Common PIP's commands

- `pip list`
  - Provides a neat list of all installed packages and their versions.
- `pip uninstall`
  - Safely removes a library
- `pip install --upgrade`
  - Updates an existing package to its latest version

Visualizing Data with Matplotlib

Why Visualize Data?

- Effective Information Communication
  - "A picture is worth a thousand words." Visual representations simplify complex datasets, making them easily understandable.
- Spot Trends & Outliers
  - Charts and graphs help us quickly identify patterns, trends, and unusual data points (outliers) that might be missed in raw numbers.
  - Frequently used in Data Science, Machine Learning,…

Introducing Matplotlib

- Matplotlib: A comprehensive library for creating static, animated, and interactive visualizations in Python.
- One of the most widely used 2D plotting libraries, offering a versatile toolkit for various data visualization needs.
- Getting Started: to use Matplotlib, you'll typically import its pyplot module:

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

➡ This convention allows us to call plotting functions easily, like plt.plot() or plt.show().

Basic Plotting Workflow

1. Prepare Data

Gather and structure your data into appropriate formats (e.g., lists, arrays) that Matplotlib can process.

2. Plot the Data

Use specific Matplotlib functions (like plt.plot(), plt.bar()) to generate the desired visualization.

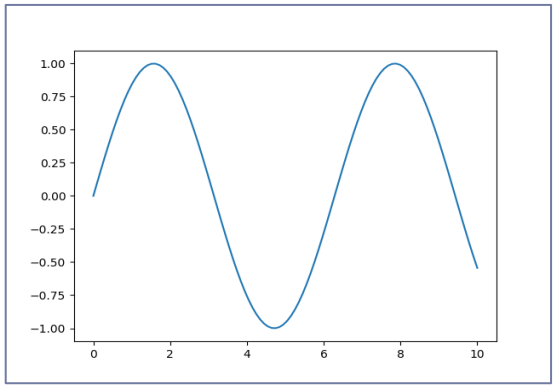
3. Show the Plot

Display the created visualization on your screen using plt.show().

Line Plots: Tracking Changes Over Time

- Line plots are often used for showing trends or changes in data over a continuous range, most commonly time.
  - E.g., stock prices over months, temperature fluctuations, or population growth.
- Core function: plt.plot(x, y)
  - x: the independent variable (e.g., time)
  - y: the dependent variable (e.g., temperature).

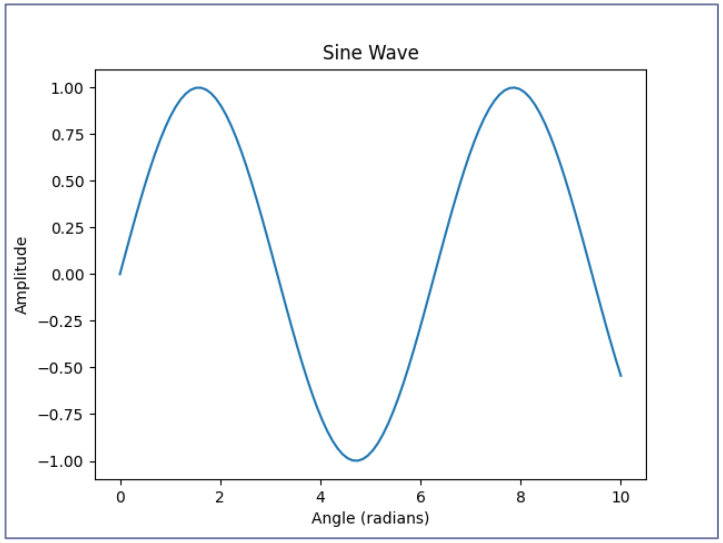
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```



Enhancing Readability: Titles and Labels

- Clear titles and axis labels are crucial for making plots understandable and professional.
- Essential Functions:
  - plt.title('Your Plot Title'): Sets the main title of the plot.
  - plt.xlabel('X-axis Label'): Labels the horizontal axis.
  - plt.ylabel('Y-axis Label'): Labels the vertical axis.

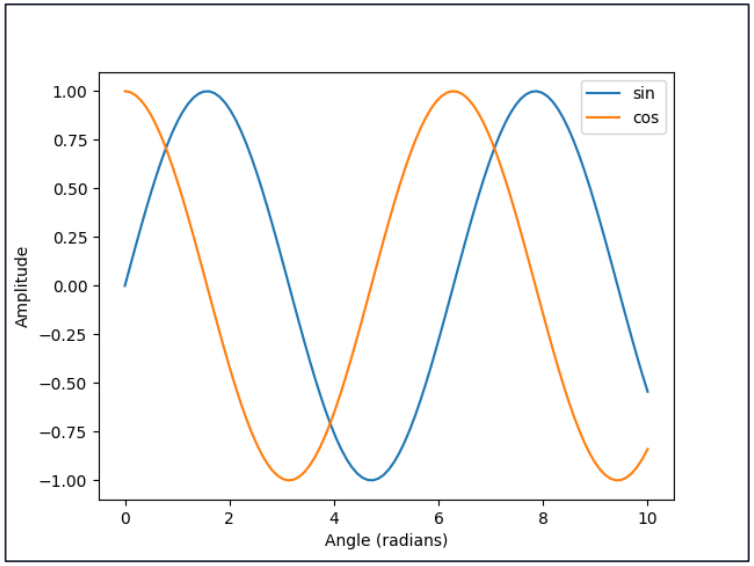
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.title('Sine Wave')
plt.xlabel('Angle (radians)')
plt.ylabel('Amplitude')
plt.show()
```



Adding Legends for Clarity

- When plotting multiple lines or datasets on a single chart, a legend helps distinguish between them.
- How to use it?
  - Provide a label argument for each plt.plot() call
  - After plotting each line, use plt.legend()

```
plt.plot(x, y1, label='sin')
plt.plot(x, y2, label='cos')
plt.legend()
```



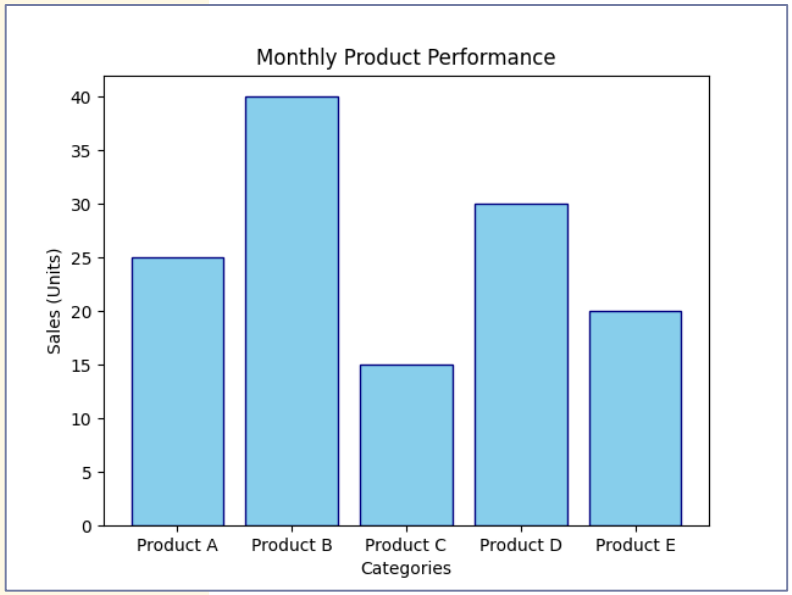
Bar Charts: Comparing Categories

- Bar charts are excellent for comparing discrete categories or showing the distribution of data across distinct groups.
- When to use it? To compare quantities between different categories, such as sales figures per region, student counts per major, or product popularity.
- Core function: `plt.bar(categories, values)`
- Customize color:

```
plt.bar(categories, values, color='salmon’)
plt.bar(categories, values, color=['red', 'green', 'blue'])
```

Bar Chart Example

```
import matplotlib.pyplot as plt

categories = ['Product A', 'Product B',
              'Product C', 'Product D',
              'Product E']
values = [25, 40, 15, 30, 20]

plt.bar(categories, values,
        color='skyblue',
        edgecolor='navy')
plt.xlabel('Categories')
plt.ylabel('Sales (Units)')
plt.title('Monthly Product
Performance')
plt.show()
```



Pie Charts: Showing Proportions

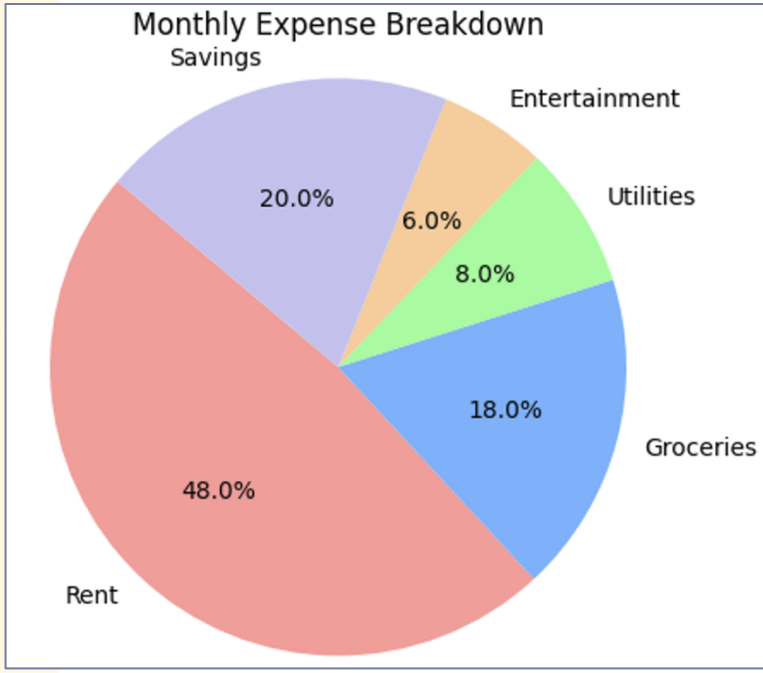
- Pie charts are used to visualize the composition of a whole, illustrating how different parts contribute to the total.
- When to use it? Ideal for displaying percentages, market share, budget allocations, or any data where you want to show parts of a whole.
- Core function:

```
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
startangle=90)
```

The autopct argument formats the percentage values displayed on each slice.

Pie Chart Example

```
import matplotlib.pyplot as plt


labels = ['Rent', 'Groceries',
'Utilities', 'Entertainment',
'Savings']
sizes = [1200, 450, 200, 150, 500]
colors =
['#ff9999','#66b3ff','#99ff99','#ffcc
9','#c2c2f0']
plt.pie(sizes, labels=labels,
autopct='%1.1f%%', startangle=140,
colors=colors)
plt.axis('equal')
plt.title('Monthly Expense Breakdown')
plt.show()
```



Histograms: Understanding Data Distribution

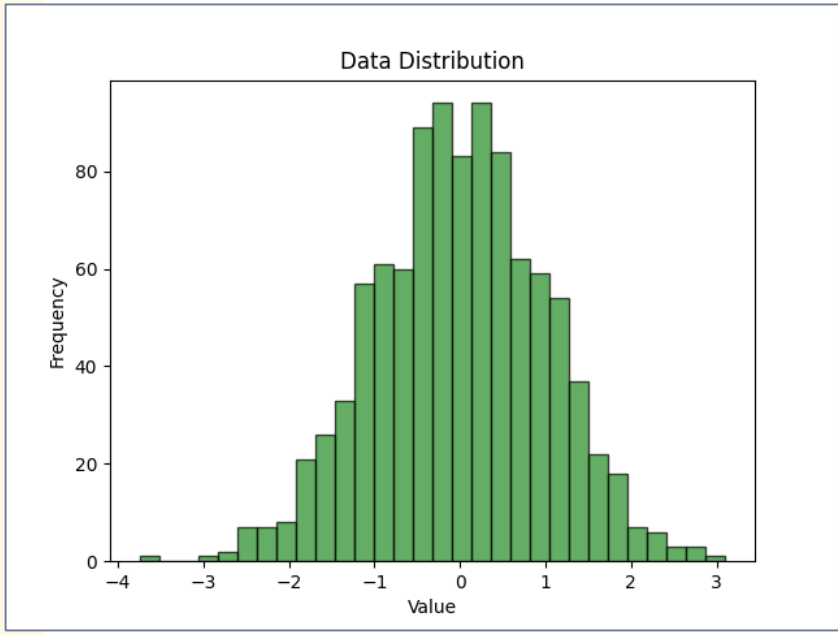
- Histograms are essential for visualizing the distribution of a continuous dataset, showing the frequency of values within specific ranges.
- When to use it? To understand the shape of data, identify peaks, spread, and skewness, such as age distribution, test scores.
- Core function:

```
plt.hist(data, bins=number_of_bins, edgecolor='black')
```

The bins argument defines how many intervals or ranges the data will be divided into.

Histogram Example

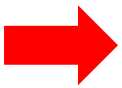
```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.randn(1000)

plt.hist(data, bins=30,
color='forestgreen',
edgecolor='black', alpha=0.7)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Data Distribution')
plt.show()
```



Data Processing with Pandas

Real-world Datasets

- Data surrounds us in various formats, often more complex than simple lists or dictionaries.
  - Tabular data
  - Image, multimedia data
  - Text
- Tabular data types
  - Excel Spreadsheets: Familiar tabular data, widely used for organization.
  - CSV Files: Comma-separated values
- Limitations of Python's Built-in Structures: Python's lists and dictionaries become cumbersome for large-scale, complex tabular data.



Introducing pandas, the Python Data Analysis Library
for efficient data manipulation and analysis of tabular data.

Introducing Pandas

- pandas is the standard tool across research and industry for working with tabular data.
- Using pandas, we can:
  - **Arrange** data in a tabular format.
  - **Extract** useful information filtered by specific conditions.
  - **Operate** on data to gain new insights.
  - **Apply** NumPy functions to our data.
  - Perform **vectorized** computations to speed up our analysis.

Series

- A Series is a 1-dimensional array-like object. It contains:
  - A sequence of values of the same type.
  - A sequence of data labels, called the index.

pd is the conventional alias for pandas

```
import pandas as pd
s = pd.Series(["welcome", "to", "COM1050"])
```

```
0    welcome
1         to
2    COM1050
dtype: object
```

Index, accessed by calling s.index

```
RangeIndex(start=0, stop=3, step=1)
```

Values, accessed by calling s.values

```
array(['welcome', 'to', 'COM1050'],
dtype=object)
```

Selection in Series

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a     4
b    -2
c     0
d     6
dtype: int64
```

- We can select a single value or a set of values in a Series:
  - A single label: `s["a"]` → 4
  - A list of labels: `s[["a", "c"]]` →

```
a     4
c     0
dtype: int64
```

  - A filtering condition:
    - 1) Apply a boolean condition to the Series, creating a new boolean Series (often called a "boolean mask").
    - 2) Index into our original Series using the boolean mask. pandas selects only the entries in the Series that satisfy the condition.

```
s > 0
```

```
a     True
b    False
c    False
d     True
dtype: bool
```

```
s[s > 0]
```

Boolean mask

```
a     4
d     6
dtype: int64
```

DataFrame: The Heart of Pandas

- DataFrame is:
  - a table in pandas.
  - a Python object.
- We can think of DataFrames as collections of Series that all share the same **Index**

Index

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

A DataFrame

```
0          Andrew Jackson
1        John Quincy Adams
2          Andrew Jackson
3        John Quincy Adams
4          Andrew Jackson
              ...
182          Donald Trump
183         Kamala Harris
184            Jill Stein
185         Robert Kennedy
186           Chase Oliver
Name: Candidate, Length: 187, dtype: object
```

A Series named "Candidate"

Creating a DataFrame

- The syntax of creating DataFrame is:

```
pandas.DataFrame(data, index, columns)
```

- We can also create a DataFrame:
  - From a CSV or an Excel file
  - From a dictionary
  - From a Series
- Understanding the syntax we show you is far more important than memorizing it.
- Knowing what you want to do is more essential than knowing precisely how to code it.

Creating a DataFrame From a CSV File

```
elections = pd.read_csv("elections.csv")
```

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

The DataFrame *elections*

Creating a DataFrame From a Dictionary

Specify columns of the DataFrame

```
pd.DataFrame({"Fruit":["Strawberry", "Orange"],
              "Price": [5.49, 3.99]})
```

Looks like JSON file format!

```
pd.DataFrame([{"Fruit":"Strawberry", "Price":5.49},
              {"Fruit":"Orange", "Price":3.99}])
```

Specify rows of the DataFrame

	Fruit	Price
0	Strawberry	5.49
1	Orange	3.99

Creating a DataFrame From a Series

```
s_a = pd.Series(["a1", "a2", "a3"], index = ["r1", "r2", "r3"])
s_b = pd.Series(["b1", "b2", "b3"], index = ["r1", "r2", "r3"])

pd.DataFrame({"A-column":s_a, "B-column":s_b})
```

	A-column	B-column
r1	a1	b1
r2	a2	b2
r3	a3	b3

```
pd.DataFrame(s_a)
```

```
s_a.to_frame()
```

	0
r1	a1
r2	a2
r3	a3

The DataFrame API

- The API for the DataFrame class is enormous.
  - API: "Application Programming Interface".
  - The API is the set of abstractions supported by the class (e.g., DataFrame.index)
- Full documentation is at pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
- We want you to get familiar with the real world programming practice of… looking up syntax!
  - Ask an LLM, check the pandas documentation, read Stack Overflow, Google, etc.

Extracting Data

- There are lots of different ways to extract rows and columns of interest from a DataFrame:
  - Grab the first or last n rows in the DataFrame
    - df.head(n) returns the first n rows of the DataFrame df
    - df.tail(n) returns the last n rows
  - Grab data with a certain label
  - Grab data at a certain position.

Label-based Extraction: .loc

- A more complex task: Extract data with specific column or index labels.

```
df.loc[row_labels, column_labels]
```

- The .loc accessor allows us to specify **labels** of rows and columns we wish to extract.
- Labels are the bolded text at the top and left of a DataFrame.

Row labels

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864

Column labels

Label-based Extraction: .loc

- Each of the two arguments to .loc can be:
  - A list

```
elections.loc[[87, 25, 179], ["Year", "Candidate", "Result"]]
```

  - A slice (inclusive of the right-hand side of the slice).

```
elections.loc[[87, 25, 179], "Popular vote":"%"]
```

  - A single value.

```
elections.loc[0, "Candidate"]
```

Integer-based Extraction: .iloc

- A different scenario: We want to extract data according to its numeric position.
  - Example: Grab the 1st, 2nd, and 3rd columns of the DataFrame.

```
df.iloc[row_integers, column_integers]
```

.iloc stands for integer location

- The .iloc accessor allows us to specify the integers of rows and columns we wish to extract.
  - Python convention: The first position has integer index 0.

Column integers

Row integers

	0	1	2	3	4	5
	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...

Integer-based Extraction: .iloc

- Each of the two arguments to .iloc can be:
  - A list:

```
elections.iloc[[1, 2, 3], [0, 1, 2]]
```

  - A slice (exclusive of the right hand side of the slice, akin to range(a,b)).

```
elections.iloc[[1, 2, 3], 0:3]
```

  - A single value.

```
elections.iloc[0, 1]
```

Context-dependent Extraction: [ ]

- [] only takes one argument, which may be:
  - A slice of row numbers (right-hand exclusive).
  - A list of column labels.
  - A single column label.
- In short: [ ] can be much more concise than .loc or .iloc
  - Ex: Extracting the "Candidate" column. It is far simpler to write elections["Candidate"] than elections.loc[:, "Candidate"]
- [] is context sensitive! But, it's the most popular.

Context-dependent Extraction: [ ]

- Using a slice of row integers (right-hand exclusive).

```
elections[3:7]
```

	Year	Candidate	Party	Popular vote	Result	%
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
5	1832	Henry Clay	National Republican	484205	loss	37.603628
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583

Context-dependent Extraction: [ ]

- Using a list of column labels.

```
elections[["Year", "Candidate", "Result"]]
```

Common typo: Forget inner brackets!

	Year	Candidate	Result
0	1824	Andrew Jackson	loss
1	1824	John Quincy Adams	win
2	1828	Andrew Jackson	win
3	1828	John Quincy Adams	loss
4	1832	Andrew Jackson	win
...	...	...	...
182	2024	Donald Trump	win
183	2024	Kamala Harris	loss
184	2024	Jill Stein	loss
185	2024	Robert Kennedy	loss
186	2024	Chase Oliver	loss

NumPy

- Pandas Series and DataFrames support many operations, including NumPy operations, so long as the data is numeric.

Filter for all rows where **Name** is 'Yash'.

```
yash_count = babynames[babynames["Name"]=="Yash"]["Count"]
```

```
np.mean(yash_count)
17.142857142857142
```

```
np.max(yash_count)
29
```

	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134
...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5
407424	CA	M	2022	Zia	5
407425	CA	M	2022	Zora	5
407426	CA	M	2022	Zuriel	5
407427	CA	M	2022	Zylo	5

babynames

```
331824     8
334114     9
336390    11
338773    12
341387    10
343571    14
345767    24
348230    29
350889    24
353445    29
356221    25
358978    27
361831    29
364905    24
367867    23
370945    18
374055    14
376756    18
379660    18
383338     9
385903    12
388529    17
391485    16
394906    10
397874     9
400171    15
403092    13
406006    13
Name: Count, dtype: int64
```

yash_count

Built-In pandas Methods

- pandas also provides an enormous number of useful utility functions, including:
  - size/shape
  - describe
  - sample
  - value_counts
  - unique
  - sort_values
  - …

.sort_values()

- The .sort_values() method sorts a DataFrame or Series.

```
babynames["Name"].sort_values()
```

```
366001      Aadan
384005      Aadan
369120      Aadan
398211    Aadarsh
370306      Aaden
           ...
220691      Zyrah
197529      Zyrah
217429      Zyrah
232167      Zyrah
404544      Zyrus
Name: Name, Length: 407428, dtype: object
```

Notice the index changes order too!

```
babynames.sort_values(by="Count", ascending=False)
```

	State	Sex	Year	Name	Count
268041	CA	M	1957	Michael	8260
267017	CA	M	1956	Michael	8258
317387	CA	M	1990	Michael	8246
281850	CA	M	1969	Michael	8245
283146	CA	M	1970	Michael	8196
...	...	...	...	...	...
317292	CA	M	1989	Olegario	5
317291	CA	M	1989	Norbert	5
317290	CA	M	1989	Niles	5
317289	CA	M	1989	Nikola	5
407427	CA	M	2022	Zylo	5

407428 rows × 5 columns

By default, rows are sorted in *ascending* order.

Common typo: Forgetting if ascending/descending is the default. Solution? Always be explicit!

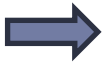
Applying a Function to a Row

- dt.apply(f) for a DataFrame dt and function f creates an array of the results of applying f to each row of dt.
  - Ex., dt.apply(sum) would return the sum of each row as an array.

Applying a Function to a Row

```
import pandas as pd

def calculate_total(row):
    """Calculates price * quantity and adds a shipping fee."""
    total = (row['Price'] * row['Quantity']) + 10
    return total
data = {
        'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard'],
        'Price': [1200, 25, 300, 75],
         'Quantity': [2, 10, 3, 5]
         }
df = pd.DataFrame(data)
df['Total_with_Shipping'] = df.apply(calculate_total, axis=1)
```

	Product	Price	Quantity
0	Laptop	1200	2
1	Mouse	25	10
2	Monitor	300	3
3	Keyboard	75	5

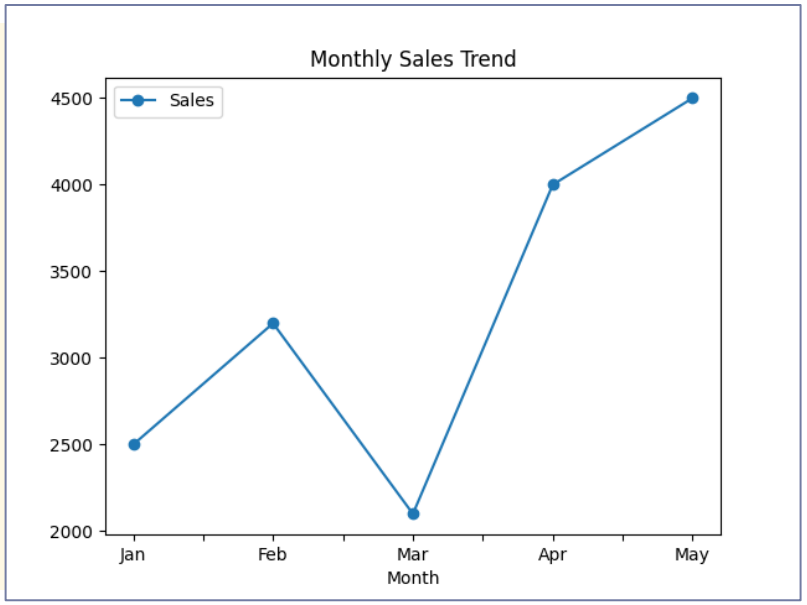


	Product	Price	Quantity	Total_with_Shipping
0	Laptop	1200	2	2410
1	Mouse	25	10	260
2	Monitor	300	3	910
3	Keyboard	75	5	385

Plotting in Pandas

- pandas also has built-in plotting capabilities powered by Matplotlib.
- Need to install both pandas and matplotlib.
- Core function: df.plot(…, kind="plot type"), plot type can be: line, bar, pie, hist, …

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'Month': ['Jan', 'Feb', 'Mar', 'Apr',
'May'],
        'Sales': [2500, 3200, 2100, 4000,
4500]}
df = pd.DataFrame(data)

df.plot(x='Month', y='Sales', kind='line',
        marker='o', title='Monthly Sales
Trend')
plt.show()
```



Summary - Key Takeaways

- **math:** common mathematical functions and constants.
- **random:** simulate uncertainty, random selection.
- **datetime:** access to date and time.
- **matplotlib:** data visualization, charts.
- **pandas:** working with tabular data.
- **pip:** install external library in Python.