

Python Guide

A companion resource to the workshop's Python scripts.

Python Basics

Core concepts that form the foundation of the Python programming language, as seen in [Day 1 - Python Introduction](#).

Variables & Data Types

Variables are labels for data. Python is dynamically typed, so you don't need to declare the type.

- `int`: Whole numbers (e.g., `30`)
- `float`: Decimal numbers (e.g., `25.99`)
- `str`: Text (e.g., `"Alice"`)
- `bool`: True or False

Functions

Reusable blocks of code defined with the `def` keyword. They help organize code and avoid repetition. To define a function:

1. Start with the `def` keyword, followed by a function name.
2. Add parentheses `()`, including any input parameters (arguments) inside.
3. End the line with a colon `:`.
4. Indent the code block that runs when the function is called.

Basic Operators

Symbols that perform operations on values.

- **Arithmetic:** `+`, `-`, `*`, `/`, `**` (exponent)
- **Comparison:** `==`, `!=`, `>`, `<`
- **Logical:** `and`, `or`, `not`

Control Flow

Statements that control the order of code execution.

- `if/elif/else`: Make decisions based on conditions.
- `for` loop: Iterate over a sequence (e.g., a list).
- `while` loop: Repeat as long as a condition is true.

5. Use `return` to send a value back as the output.

Function Example

```
# This function takes two numbers and returns their sum.
def add_numbers(a, b):
    return a + b

# Call the function and store the result in a variable.
sum_result = add_numbers(5, 7)

# Print the result
print(sum_result) # Output will be 12
```

Methods: Functions That Belong to Objects

A method is a function that is associated with an object. In Python, almost everything is an object (a string, a list, a DataFrame, etc.), and these objects come with built-in methods to perform common tasks.

The key difference is syntax: a function is called by name, passing the object as an argument (e.g., `print(my_variable)`), while a method is called directly on the object using dot notation (e.g., `my_object.method_name()`).

Pandas DataFrames have many useful methods. For example, the `.head()` method is used to quickly preview the first few rows of your data.

```
# Assume 'arab_exports' is a pandas DataFrame
# The .head() method is called on the DataFrame object
top_rows = arab_exports.head()

# Display the result
print(top_rows)
```

The Importance of Indentation

Unlike many other programming languages that use curly braces `{}` to define blocks of code, Python uses indentation. This is not just for styling or readability—it's a strict syntax rule. The amount of space (typically four spaces per level, which is equivalent to a single press of the Tab key in most code editors) tells the Python interpreter which lines of code belong to a specific block, such as the body of a loop, a function, or an `if` statement.

Incorrect indentation will cause an `IndentationError` and the program will not run.

```
# CORRECT: The print statement is indented, so it's inside the for loop.
for i in range(3):
    print(f"This is loop number {i}")

# INCORRECT: This will raise an IndentationError.
for i in range(3):
print(f"This will cause an error!")
```

Data Structures

Lists

Ordered, mutable (changeable) collections of items. Defined with square brackets `[]`.

```
my_list = [1, "apple", True]
```

Tuples

Ordered, immutable (unchangeable) collections. Defined with parentheses `()`.

```
my_tuple = (1, "apple", True)
```

Dictionaries

Unordered, mutable collections of key-value pairs. Defined with curly braces `{}`.

```
my_dict = {"user_id": 101, "sta
```

Key Packages and Functions

An overview of the essential libraries and functions used in [Day 2 - Trade Data Exploration](#) and [Day 3 - Indicators and Clustering](#) for data manipulation, visualization, and machine learning.

Pandas (`'pd'`) - Data Manipulation

Reading Data

`pd.read_csv()` and `pd.read_excel()` are used to load data from files into DataFrames. Parameters like `usecols` and `dtype` help load data efficiently.

Viewing Data

`.head()` shows the first few rows.
`.sum()` calculates the total of a column.
`.sort_values()` orders the DataFrame by a column's values.

Transforming Data

`.map()` substitutes each value in a Series with another value. `.apply()` invokes a function on values of a DataFrame. `.str.wrap()` helps format long text for plots.

Plotly Express (`px`) - Data Visualization

Chart Types

`px.bar()` creates bar charts, ideal for comparing categories.
`px.scatter()` creates scatter plots, useful for showing relationships and clusters.

Customization

Arguments like `title`, `labels`, `color`, and `hover_name` control the appearance and interactivity of plots. Functions like `.update_xaxes()` provide finer control.

Scikit-learn (`sklearn`) - Machine Learning

Clustering with KMeans

`KMeans` is an algorithm that groups data points into a specified number of clusters (`n_clusters`). It aims to make the clusters as distinct as possible.

Model Training & Prediction

The `.fit_predict()` method both trains the model on the data and assigns each data point to a cluster. The `inertia_` attribute helps evaluate how compact the clusters are.

Core Concept: Merging DataFrames with `pd.merge()`

Combining datasets is a fundamental task. The `pd.merge()` function joins DataFrames based on a common column (a "key"). The `how` parameter determines which rows are kept. For a more detailed explanation please refer to [merging_explainer.pdf](#).

Left Join (`how="left"`)

Keeps all rows from the **left** DataFrame and adds matching data from the right. If there's no match, it fills with `NaN` (Not a Number).

Right Join (`how="right"`)

Keeps all rows from the **right** DataFrame and adds matching data from the left.

Inner Join (`how="inner"`)

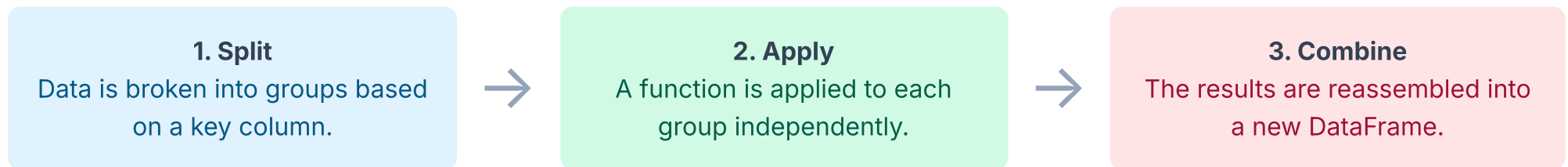
Keeps **only** the rows that have matching keys in **both** DataFrames. This is the default behavior.

Outer Join (`how="outer"`)

Keeps **all** rows from **both** DataFrames. Where data is missing on either side, it's filled with `NaN`.

Core Concept: Grouping Data with `.groupby()`

The `.groupby()` operation is a powerful tool for data analysis that follows the "Split-Apply-Combine" strategy. For a more detailed explanation please refer to [groupby explainer.pdf](#).



Example: Calculating Total Exports by Continent

Goal: Find the total exports for each continent from a trade DataFrame.

```
df.groupby('Continent')['Exports'].sum()
```



The code first **splits** the data by 'Continent', then **applies** the `.sum()` function to the 'Exports' of each group, and finally **combines** the results into a summary table.

Continent	Total Exports
Asia	\$4,950B
Europe	\$2,200B