

Working with Data in Python Cheat Sheet

Reading and writing files

Package/Method	Description	Syntax and Code Example
File opening modes	Different modes to open files for specific operations.	<p>Syntax: r (reading) w (writing) a (appending) + (updating: read/write) b (binary, otherwise text)</p> <p>Examples:</p> <pre>with open("data.txt", "r") as file: content = file.read() print(content) with open("output.txt", "w") as file: file.write("Hello World!")</pre>
File reading methods	Different methods to read file content in various ways.	<p>Syntax:</p> <pre>file.readlines() # reads all lines as a list readline() # reads the next line as a string file.read() # reads the entire file content as a string</pre> <p>Example:</p> <pre>with open("data.txt", "r") as file: lines = file.readlines() next_line = file.readline() content = file.read()</pre>
File writing methods	Different write methods to write content to a file.	<p>Syntax:</p> <pre>file.write(content) # writes a string to the file file.writelines(lines) # writes a list of strings to the file</pre> <p>Example:</p> <pre>lines = ["Hello\n", "World\n"] with open("output.txt", "w") as file: file.writelines(lines)</pre>
Iterating over lines	Iterates through each line in the file	<p>Syntax:</p> <pre>for line in file: # Code to process each line</pre>

	using a 'loop'.	<p>Example:</p> <pre>with open("data.txt", "r") as file: for line in file: print(line)</pre>
Open() and close()	Opens a file, performs operations, and explicitly closes the file using the close() method.	<p>Syntax:</p> <pre>file = open(filename, mode) # Code that uses the file file.close()</pre> <p>Example:</p> <pre>file = open("data.txt", "r") content = file.read() file.close()</pre>
with open()	Opens a file using a with block, ensuring automatic file closure after usage.	<p>Syntax:</p> <pre>with open(filename, mode) as file: # Code that uses the file</pre> <p>Example:</p> <pre>with open("data.txt", "r") as file: content = file.read()</pre>

Pandas

Package/Method	Description	Syntax and Code Example
.read_csv()	Reads data from a '.CSV' file and creates a DataFrame.	Syntax: dataframe_name = pd.read_csv("filename.csv") Example: df = pd.read_csv("data.csv")

		<p>Syntax:</p> <pre>dataframe_name = pd.read_excel("filename.xlsx")</pre>
.read_excel()	Reads data from an Excel file and creates a DataFrame.	<p>Example:</p> <pre>df = pd.read_excel("data.xlsx")</pre>
.to_csv()	Writes DataFrame to a CSV file.	<p>Syntax:</p> <pre>dataframe_name.to_csv("output.csv", index=False)</pre> <p>Example:</p> <pre>df.to_csv("output.csv", index=False)</pre>
Access Columns	Accesses a specific column using [] in the DataFrame.	<p>Syntax:</p> <pre>dataframe_name["column_name"] # Accesses single column dataframe_name[["column1", "column2"]] # Accesses multiple columns</pre> <p>Example:</p> <pre>df["age"] df[["name", "age"]]</pre>
describe()	Generates statistics summary of numeric columns in the DataFrame.	<p>Syntax:</p> <pre>dataframe_name.describe()</pre>

		<p>Example:</p> <pre>df.describe()</pre>
drop()	Removes specified rows or columns from the DataFrame. axis=1 indicates columns. axis=0 indicates rows.	<p>Syntax:</p> <pre>dataframe_name.drop(["column1", "column2"], axis=1, inplace=True) dataframe_name.drop(index=[row1, row2], axis=0, inplace=True)</pre> <p>Example:</p> <pre>df.drop(["age", "salary"], axis=1, inplace=True) # Will drop columns df.drop(index=[5, 10], axis=0, inplace=True) # Will drop rows</pre>
dropna()	Removes rows with missing NaN values from the DataFrame. axis=0 indicates rows.	<p>Syntax:</p> <pre>dataframe_name.dropna(axis=0, inplace=True)</pre> <p>Example:</p> <pre>df.dropna(axis=0, inplace=True)</pre>
duplicated()	Duplicate or repetitive values or records within a data set.	<p>Syntax:</p> <pre>dataframe_name.duplicated()</pre>

		Example: <code>duplicate_rows = df[df.duplicated()]</code>
Filter Rows	Creates a new DataFrame with rows that meet specified conditions.	Syntax: <code>filtered_df = dataframe_name[(Conditional_statements)]</code> Example: <code>filtered_df = df[(df["age"] > 30) & (df["salary"] < 50000)]</code>
groupby()	Splits a DataFrame into groups based on specified criteria, enabling subsequent aggregation, transformation, or analysis within each group.	Syntax: <code>grouped = dataframe_name.groupby(by, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, observed=False, dropna=True)</code> Example: <code>grouped = df.groupby(["category", "region"]).agg({"sales": "sum"})</code>
head()	Displays the first n rows of the DataFrame.	Syntax: <code>dataframe_name.head(n)</code>

		<p>Example:</p> <pre>df.head(5)</pre>
Import pandas	Imports the Pandas library with the alias pd.	<p>Syntax:</p> <pre>import pandas as pd</pre> <p>Example:</p> <pre>import pandas as pd</pre>
info()	Provides information about the DataFrame, including data types and memory usage.	<p>Syntax:</p> <pre>dataframe_name.info()</pre> <p>Example:</p> <pre>df.info()</pre>
merge()	Merges two DataFrames based on multiple common columns.	<p>Syntax:</p> <pre>merged_df = pd.merge(df1, df2, on=["column1", "column2"])</pre> <p>Example:</p> <pre>merged_df = pd.merge(sales, products, on=["product_id", "category_id"])</pre>

print DataFrame	Displays the content of the DataFrame.	Syntax: <pre>print(df) # or just type df</pre> Example: <pre>print(df) df</pre>
replace()	Replaces specific values in a column with new values.	Syntax: <pre>dataframe_name["column_name"].replace(old_value, new_value, inplace=True)</pre> Example: <pre>df["status"].replace("In Progress", "Active", inplace=True)</pre>
tail()	Displays the last n rows of the DataFrame.	Syntax: <pre>dataframe_name.tail(n)</pre> Example: <pre>df.tail(5)</pre>

Numpy

Package/Method	Description	Syntax and Code Example
Importing NumPy	Imports the NumPy library.	<p>Syntax:</p> <pre>import numpy as np</pre> <p>Example:</p> <pre>import numpy as np</pre>
np.array()	Creates a one or multi-dimensional array,	<p>Syntax:</p> <pre>array_1d = np.array([list1 values]) # 1D Array array_2d = np.array([[list1 values], [list2 values]]) # 2D Array</pre> <p>Example:</p> <pre>array_1d = np.array([1, 2, 3]) # 1D Array array_2d = np.array([[1, 2], [3, 4]]) # 2D Array</pre>
Numpy Array Attributes	<ul style="list-style-type: none"> - Calculates the mean of array elements - Calculates the sum of array elements - Finds the minimum value in the array - Finds the maximum value in the array - Computes dot product of two arrays 	<p>Example:</p> <pre>np.mean(array) np.sum(array) np.min(array) np.max(array) np.dot(array_1, array_2)</pre>



