List:

Adding Elements

Using append():

python

Copy code

fruits.append("orange")

print(fruits) # Output: ['apple', 'banana', 'cherry', 'orange']

Using insert():

python

Copy code

fruits.insert(1, "kiwi")

print(fruits) # Output: ['apple', 'kiwi', 'banana', 'cherry', 'orange']

. Removing Elements

Using remove():

python

Copy code

fruits.remove("banana")

print(fruits) # Output: ['apple', 'kiwi', 'cherry', 'orange']

Using pop():

python

Copy code

last\_fruit = fruits.pop() # Removes and returns the last item

print(last\_fruit) # Output: "orange"

print(fruits) # Output: ['apple', 'kiwi', 'cherry']

List Comprehensions

python

Copy code

# Create a new list with squares of numbers

numbers = [1, 2, 3, 4]

squares = [x\*\*2 for x in numbers]

print(squares) # Output: [1, 4, 9, 16]

Filtering Lists

python

Copy code

# Filter even numbers from a list

numbers = [1, 2, 3, 4, 5, 6]

evens = [num for num in numbers if num % 2 == 0]

print(evens) # Output: [2, 4, 6]

Using sorted():

python

Copy code

sorted\_fruits = sorted(fruits, reverse=True)

print(sorted\_fruits) # Output: ['cherry', 'banana', 'apple']

Copying Lists

python

Copy code

# Using slice to create a copy

fruits\_copy = fruits[:]

print(fruits\_copy) # Output: ['cherry', 'banana', 'apple']

Finding Index of an Element

python

Copy code

index = fruits.index("banana") # Find the index of "banana"

print(index) # Output: 1

Count Occurrences of an Element

python

Copy code

fruits = ["apple", "banana", "apple"]

count\_apples = fruits.count("apple")

print(count\_apples) # Output: 2

-----------------------------------------------

Combining Lists with Conditions

You can combine elements from multiple lists based on certain conditions.

python

Copy code

list1 = [1, 2, 3]

list2 = [4, 5, 6]

# Combine elements from both lists where the sum is greater than 7

combined = [(x, y) for x in list1 for y in list2 if x + y > 7]

print(combined) # Output: [(3, 5), (3, 6)]

Flattening a Nested List

Flatten a nested list using list comprehension.

python

Copy code

nested\_list = [[1, 2, 3], [4, 5], [6, 7, 8]]

# Flatten the nested list

flattened = [item for sublist in nested\_list for item in sublist]

print(flattened) # Output: [1, 2, 3, 4, 5, 6, 7, 8]

Using zip() to Combine Lists

You can combine lists element-wise using the zip() function.

python

Copy code

names = ["Alice", "Bob", "Charlie"]

ages = [25, 30, 35]

# Combine names and ages

combined = list(zip(names, ages))

print(combined) # Output: [('Alice', 25), ('Bob', 30), ('Charlie', 35)]

Unzipping a Combined List

You can unzip a combined list back into two separate lists.

python

Copy code

combined = [('Alice', 25), ('Bob', 30), ('Charlie', 35)]

# Unzip the combined list

names, ages = zip(\*combined)

print(names) # Output: ('Alice', 'Bob', 'Charlie')

print(ages) # Output: (25, 30, 35)

Removing Duplicates While Maintaining Order

You can remove duplicates while maintaining the order of elements.

python

Copy code

items = [1, 2, 2, 3, 4, 4, 5]

# Remove duplicates while preserving order

unique\_items = []

[unique\_items.append(x) for x in items if x not in unique\_items]

print(unique\_items) # Output: [1, 2, 3, 4, 5]

List of Lists (Matrix) Operations

You can perform operations on a list of lists (matrix), such as transposing it.

python

Copy code

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Transpose the matrix

transposed = [[row[i] for row in matrix] for i in range(len(matrix[0]))]

print(transposed)

# Output: [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

Finding the Intersection of Two Lists

Find common elements between two lists.

python

Copy code

list1 = [1, 2, 3, 4]

list2 = [3, 4, 5, 6]

# Find the intersection

intersection = [value for value in list1 if value in list2]

print(intersection) # Output: [3, 4]

Finding the Union of Two Lists

Combine two lists and remove duplicates.

python

Copy code

list1 = [1, 2, 3, 4]

list2 = [3, 4, 5, 6]

# Find the union

union = list(set(list1) | set(list2))

print(union) # Output: [1, 2, 3, 4, 5, 6]

Using map() to Apply a Function to a List

You can use map() to apply a function to each element of a list.

python

Copy code

numbers = [1, 2, 3, 4]

# Square each number in the list

squared = list(map(lambda x: x \*\* 2, numbers))

print(squared) # Output: [1, 4, 9, 16]

Filtering and Transforming a List in One Step

Using filter() and map() together.

python

Copy code

numbers = [1, 2, 3, 4, 5, 6]

# Filter even numbers and square them

squared\_evens = list(map(lambda x: x \*\* 2, filter(lambda x: x % 2 == 0, numbers)))

print(squared\_evens) # Output: [4, 16, 36]

Using reduce() to Accumulate Values

Using the reduce() function from the functools module to accumulate values.

python

Copy code

from functools import reduce

numbers = [1, 2, 3, 4]

# Calculate the product of the numbers

product = reduce(lambda x, y: x \* y, numbers)

print(product) # Output: 24

List Sorting with Custom Keys

You can sort lists using custom sorting keys.

python

Copy code

words = ["banana", "apple", "cherry", "date"]

# Sort by length of words

sorted\_words = sorted(words, key=len)

print(sorted\_words) # Output: ['date', 'apple', 'banana', 'cherry']

Zipping Multiple Lists with Different Lengths

Handling lists of different lengths with zip\_longest().

python

Copy code

from itertools import zip\_longest

list1 = [1, 2, 3]

list2 = ['a', 'b']

# Zip with fill value for missing elements

zipped = list(zip\_longest(list1, list2, fillvalue='missing'))

print(zipped) # Output: [(1, 'a'), (2, 'b'), (3, 'missing')]

# Flatten a matrix

matrix = [[1, 2], [3, 4], [5, 6]]

flattened = [num for row in matrix for num in row]

print(flattened) # Output: [1, 2, 3, 4, 5, 6]

map()

map() applies a function to all the items in an input list or any iterable.

Basic Example:

python

Copy code

# Square each number in the list

numbers = [1, 2, 3, 4]

squares = list(map(lambda x: x \*\* 2, numbers))

print(squares) # Output: [1, 4, 9, 16]

With Multiple Lists:

python

Copy code

# Add corresponding elements of two lists

list1 = [1, 2, 3]

list2 = [4, 5, 6]

summed = list(map(lambda x, y: x + y, list1, list2))

print(summed) # Output: [5, 7, 9]

filter()

filter() constructs a list of elements from an iterable for which a function returns True.

Basic Example:

python

Copy code

# Filter out odd numbers

numbers = [1, 2, 3, 4, 5, 6]

evens = list(filter(lambda x: x % 2 == 0, numbers))

print(evens) # Output: [2, 4, 6]

With Strings:

python

Copy code

# Filter out words longer than 3 letters

words = ["cat", "dog", "elephant", "rat"]

short\_words = list(filter(lambda word: len(word) <= 3, words))

print(short\_words) # Output: ['cat', 'dog', 'rat']

reduce()

reduce() is used to apply a function to elements of an iterable cumulatively to reduce it to a single value. It is available in the functools module.

Basic Example:

python

Copy code

from functools import reduce

# Calculate the product of numbers

numbers = [1, 2, 3, 4]

product = reduce(lambda x, y: x \* y, numbers)

print(product) # Output: 24

Combining map(), filter(), and List Comprehension

You can combine these operations to perform complex transformations.

Filter even numbers and then square them:

python

Copy code

numbers = [1, 2, 3, 4, 5, 6]

# Filter even numbers and then square them

squared\_evens = list(map(lambda x: x \*\* 2, filter(lambda x: x % 2 == 0, numbers)))

print(squared\_evens) # Output: [4, 16, 36]

Creating a Dictionary from Lists:

python

Copy code

keys = ['name', 'age', 'city']

values = ['Alice', 25, 'New York']

# Create a dictionary using list comprehension

dictionary = {k: v for k, v in zip(keys, values)}

print(dictionary) # Output: {'name': 'Alice', 'age': 25, 'city': 'New York'}

Using reduce() for String Concatenation

You can use reduce() to concatenate strings.

Concatenating a list of strings:

python

Copy code

from functools import reduce

words = ['hello', 'world', 'from', 'python']

# Concatenate all words

sentence = reduce(lambda x, y: x + ' ' + y, words)

print(sentence) # Output: 'hello world from python'

Combining map(), filter(), and reduce()

For more complex scenarios, these can be combined.

Filter out words longer than 3 characters, convert to uppercase, and concatenate:

python

Copy code

from functools import reduce

words = ['this', 'is', 'a', 'test', 'of', 'map', 'filter', 'reduce']

# Filter, map, and reduce

result = reduce(lambda x, y: x + ' ' + y,

map(str.upper,

filter(lambda word: len(word) > 3, words)))

print(result) # Output: 'THIS TEST FILTER REDUCE'

------------------------------------------------------------------

py os:

here are one-liner notes for the functions used in the previous code, along with examples:

os.path.exists(path): Checks if a path exists.

python

Copy code

if not os.path.exists('destination\_directory'):

os.mkdir('destination\_directory')

os.mkdir(path): Creates a new directory at the specified path.

python

Copy code

os.mkdir('new\_directory')

os.listdir(path): Lists all files and directories in the specified path.

python

Copy code

files = os.listdir('source\_directory')

str.endswith(suffix): Checks if a string ends with the specified suffix (used to filter Excel files).

python

Copy code

if filename.endswith('.xlsx'):

# Do something

random.randint(a, b): Generates a random integer between a and b.

python

Copy code

random\_num = random.randint(1000, 9999)

os.path.splitext(path): Splits the file name and extension.

python

Copy code

file\_name, file\_extension = os.path.splitext('file.xlsx')

os.path.join(dir, file): Joins directory and file name into a full path.

python

Copy code

full\_path = os.path.join('directory', 'file.xlsx')

shutil.move(src, dst): Moves a file or directory to a new location.

python

Copy code

shutil.move('source/file.xlsx', 'destination/new\_file.xlsx')

----------------------------------------------------

pypanfilter

1. Selecting Rows Based on Condition(s)

Single Condition:

python

Copy code

# Select rows where 'Age' is greater than 30

df\_filtered = df[df['Age'] > 30]

print(df\_filtered)

Multiple Conditions (AND):

python

Copy code

# Select rows where 'Age' > 30 AND 'Salary' > 50000

df\_filtered = df[(df['Age'] > 30) & (df['Salary'] > 50000)]

print(df\_filtered)

Multiple Conditions (OR):

python

Copy code

# Select rows where 'Age' > 30 OR 'Salary' > 50000

df\_filtered = df[(df['Age'] > 30) | (df['Salary'] > 50000)]

print(df\_filtered)

Negating a Condition:

python

Copy code

# Select rows where 'Age' is NOT greater than 30

df\_filtered = df[~(df['Age'] > 30)]

print(df\_filtered)

2. Filtering Based on String Values

Exact Match:

python

Copy code

# Select rows where 'Department' is 'IT'

df\_filtered = df[df['Department'] == 'IT']

print(df\_filtered)

Partial String Match using .str.contains():

python

Copy code

# Select rows where 'Department' contains 'Tech'

df\_filtered = df[df['Department'].str.contains('Tech', case=False)]

print(df\_filtered)

String Starts with:

python

Copy code

# Select rows where 'Name' starts with 'A'

df\_filtered = df[df['Name'].str.startswith('A')]

print(df\_filtered)

String Ends with:

python

Copy code

# Select rows where 'Name' ends with 'son'

df\_filtered = df[df['Name'].str.endswith('son')]

print(df\_filtered)

3. Selecting Specific Columns

Selecting a Single Column:

python

Copy code

# Select a single column 'Name'

name\_column = df['Name']

print(name\_column)

Selecting Multiple Columns:

python

Copy code

# Select multiple columns 'Name' and 'Age'

df\_selected = df[['Name', 'Age']]

print(df\_selected)

4. Using .loc[] and .iloc[] for Selection

Using .loc[] (Label-based selection):

python

Copy code

# Select row by label/index 3 and columns 'Name' and 'Age'

df\_loc = df.loc[3, ['Name', 'Age']]

print(df\_loc)

Using .iloc[] (Integer-location based selection):

python

Copy code

# Select the first 3 rows and the first 2 columns

df\_iloc = df.iloc[:3, :2]

print(df\_iloc)

Using .loc[] for condition-based selection:

python

Copy code

# Select all rows where 'Age' > 30 and show 'Name' and 'Salary' columns

df\_loc\_filtered = df.loc[df['Age'] > 30, ['Name', 'Salary']]

print(df\_loc\_filtered)

5. Using .query() Method for Filtering

Filter using a SQL-like syntax:

python

Copy code

# Use .query() to filter rows where 'Age' > 30 and 'Salary' > 50000

df\_filtered = df.query('Age > 30 and Salary > 50000')

print(df\_filtered)

Filter with multiple conditions:

python

Copy code

# Filter rows where 'Age' > 30 OR 'Department' is 'HR'

df\_filtered = df.query('Age > 30 or Department == "HR"')

print(df\_filtered)

6. Selecting Rows with .isin()

Filter rows where a column has specific values:

python

Copy code

# Select rows where 'Department' is either 'HR' or 'IT'

df\_filtered = df[df['Department'].isin(['HR', 'IT'])]

print(df\_filtered)

Filter rows NOT in a list of values:

python

Copy code

# Select rows where 'Department' is NOT 'HR' or 'IT'

df\_filtered = df[~df['Department'].isin(['HR', 'IT'])]

print(df\_filtered)

7. Filter Rows Based on Range of Values

Using .between() for range selection:

python

Copy code

# Select rows where 'Age' is between 25 and 40 (inclusive)

df\_filtered = df[df['Age'].between(25, 40)]

print(df\_filtered)

Non-inclusive range selection:

python

Copy code

# Select rows where 'Age' is between 25 and 40, exclusive

df\_filtered = df[df['Age'].between(25, 40, inclusive='neither')]

print(df\_filtered)

8. Handling Missing Values During Selection

Filter rows with missing values in a column:

python

Copy code

# Select rows where 'Salary' is missing (NaN)

df\_missing = df[df['Salary'].isna()]

print(df\_missing)

Filter rows where column is NOT missing:

python

Copy code

# Select rows where 'Salary' is NOT missing

df\_not\_missing = df[df['Salary'].notna()]

print(df\_not\_missing)

-----------------------------------

import openpyxl

# Function to delete a sheet with a given name from an Excel file

def delete\_sheet\_by\_name(excel\_file, sheet\_name):

# Load the workbook

wb = openpyxl.load\_workbook(excel\_file)

# Check if the sheet exists in the workbook

if sheet\_name in wb.sheetnames:

# Remove the sheet

wb.remove(wb[sheet\_name])

# Save the workbook after removing the sheet

wb.save(excel\_file)

print(f"Sheet '{sheet\_name}' deleted from {excel\_file}")

else:

print(f"Sheet '{sheet\_name}' not found in {excel\_file}")

# List of Excel files

excel\_files = ['file1.xlsx', 'file2.xlsx', 'file3.xlsx']

sheet\_name\_to\_delete = 'Sheet1' # Specify the sheet name you want to delete

# Iterate through each file and delete the specified sheet

for file in excel\_files:

delete\_sheet\_by\_name(file, sheet\_name\_to\_delete)

def my\_function(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key} = {value}")

my\_function(name="Alice", age=25, job="Engineer")

name = Alice

age = 25

job = Engineer

How it works:

The \*\*kwargs collects all the extra keyword arguments into a dictionary.

In the example above, kwargs becomes {'name': 'Alice', 'age': 25, 'job': 'Engineer'}, which you can loop through to process the key-value pairs.

When to use \*\*kwargs:

When you want to allow a function to handle any number of named keyword arguments, or when you don't know the argument names in advance.

This is useful when a function might be called with different sets of keyword arguments, like in configuration settings or APIs.

def my\_function(\*args, \*\*kwargs):

print("Positional arguments:", args)

print("Keyword arguments:", kwargs)

my\_function(1, 2, 3, name="Alice", age=25)

Positional arguments: (1, 2, 3)

Keyword arguments: {'name': 'Alice', 'age': 25}

In this case:

args will be (1, 2, 3) (the positional arguments).

kwargs will be {'name': 'Alice', 'age': 25} (the keyword arguments).

DeCOOOOO

def my\_decorator(func):

def wrapper(name):

print("Before the greeting")

result = func(name) # Call the original function

print("After the greeting")

return result

return wrapper

@my\_decorator

def greet(name):

return f"Hello, {name}!"

print(greet("Alice"))

Before the greeting

Hello, Alice!

After the greeting

def log\_decorator(func):

def wrapper(\*args, \*\*kwargs):

print(f"Calling {func.\_\_name\_\_} with arguments: {args}, {kwargs}")

result = func(\*args, \*\*kwargs)

print(f"{func.\_\_name\_\_} returned: {result}")

return result

return wrapper

@log\_decorator

def add(a, b):

return a + b

print(add(5, 3))

Calling add with arguments: (5, 3), {}

add returned: 8

8

####

def requires\_permission(func):

def wrapper(user):

if user == "admin":

return func(user)

else:

return "Access Denied"

return wrapper

@requires\_permission

def view\_admin\_page(user):

return f"Welcome to the admin page, {user}!"

print(view\_admin\_page("admin")) # Allowed

print(view\_admin\_page("guest")) # Denied

Welcome to the admin page, admin!

Access Denied

#####

import time

def timer\_decorator(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

print(f"{func.\_\_name\_\_} took {end\_time - start\_time} seconds to execute")

return result

return wrapper

@timer\_decorator

def long\_task():

time.sleep(2) # Simulate a long-running task

return "Task Completed!"

print(long\_task())

long\_task took 2.0023415088653564 seconds to execute

Task Completed!

####

def requires\_permission(func):

def wrapper(user):

if user == "admin":

return func(user)

else:

return "Access Denied!"

return wrapper

@requires\_permission

def view\_admin\_page(user):

return f"Welcome to the admin page, {user}!"

print(view\_admin\_page("admin")) # Output: Welcome to the admin page, admin!

print(view\_admin\_page("guest")) # Output: Access Denied!

-----------------------------------

--------------------------------------------------------

string

Reversing a String:

python

Copy code

reversed\_string = string[::-1] # Output: "dlroW olleH"

Splitting a String:

python

Copy code

sentence = "Python is fun"

words = sentence.split(" ") # Output: ['Python', 'is', 'fun']

Joining a List of Strings:

python

Copy code

words\_list = ['Python', 'is', 'fun']

sentence = " ".join(words\_list) # Output: "Python is fun"

Replacing Substrings:

python

Copy code

string = "I love Python"

new\_string = string.replace("Python", "coding") # Output: "I love coding"

###########################33

String ADV

1. Filtering Specific Words from a String

Suppose you have a string with some unwanted words, and you want to filter them out.

Using split(), list comprehension, and join():

python

Copy code

sentence = "Python is awesome but sometimes difficult"

unwanted\_words = ["but", "difficult"]

# Split the sentence into words

words = sentence.split()

# Use list comprehension to filter out unwanted words

filtered\_words = [word for word in words if word not in unwanted\_words]

# Join the filtered words back into a sentence

filtered\_sentence = " ".join(filtered\_words)

print(filtered\_sentence) # Output: "Python is awesome sometimes"

2. Replace Words and Filter Others

In this example, we will replace certain words and filter out specific ones in the same process.

python

Copy code

sentence = "I enjoy coding in Python but JavaScript is challenging"

replace\_words = {"Python": "Java", "JavaScript": "C++"}

filter\_words = ["but"]

# Split the sentence into words

words = sentence.split()

# Replace words and filter at the same time

processed\_words = [

replace\_words.get(word, word) for word in words if word not in filter\_words

]

# Join the result back into a string

processed\_sentence = " ".join(processed\_words)

print(processed\_sentence) # Output: "I enjoy coding in Java C++ is challenging"

3. Remove Punctuation and Specific Words, then Rejoin

You may want to clean up punctuation from a string, remove certain words, and then reassemble the string.

python

Copy code

import string

sentence = "Python, is; very! fun but sometimes, tough."

unwanted\_words = ["but", "sometimes"]

# Remove punctuation using translation table

cleaned\_sentence = sentence.translate(str.maketrans('', '', string.punctuation))

# Split the cleaned sentence into words

words = cleaned\_sentence.split()

# Filter out unwanted words

filtered\_words = [word for word in words if word not in unwanted\_words]

# Join the remaining words back into a sentence

final\_sentence = " ".join(filtered\_words)

print(final\_sentence) # Output: "Python is very fun tough"

4. Replacing and Filtering Numbers

You might want to replace numbers with specific values while filtering out certain other words.

python

Copy code

sentence = "I have 2 apples and 5 bananas but no oranges"

replace\_numbers = {"2": "three", "5": "ten"}

filter\_words = ["but", "no"]

# Split the sentence

words = sentence.split()

# Replace numbers and filter unwanted words

processed\_words = [

replace\_numbers.get(word, word) for word in words if word not in filter\_words

]

# Rejoin the processed words

result\_sentence = " ".join(processed\_words)

print(result\_sentence) # Output: "I have three apples and ten bananas oranges"

5. Using replace() and Filtering Substrings

This example combines replace() and filtering out specific substrings:

python

Copy code

sentence = "The weather is very sunny and bright today, but it's too hot."

replace\_phrase = {"sunny": "cloudy", "bright": "dull"}

filter\_phrases = ["but", "too hot"]

# Replace and split the sentence into words

for old, new in replace\_phrase.items():

sentence = sentence.replace(old, new)

# Now split and filter out unwanted words

filtered\_sentence = " ".join(

[word for word in sentence.split() if word not in filter\_phrases]

)

print(filtered\_sentence) # Output: "The weather is very cloudy and dull today, it's"

6. Removing Extra Whitespace and Filtering Certain Words

You might have a string with extra whitespace and unwanted words. Here's how to clean it up:

python

Copy code

sentence = " Python is amazing, but sometimes tricky "

unwanted\_words = ["but", "tricky"]

# Strip leading/trailing whitespace and split the sentence

words = sentence.strip().split()

# Filter and remove unwanted words

filtered\_words = [word for word in words if word not in unwanted\_words]

# Join the words back into a sentence with single spaces

final\_sentence = " ".join(filtered\_words)

print(final\_sentence) # Output: "Python is amazing sometimes"

7. Replace Specific Characters and Filter Words in One Go

This example replaces certain characters (like commas and periods) and removes certain words.

python

Copy code

sentence = "Hello, world. Python is great but has its quirks."

replace\_characters = {",": "", ".": ""}

filter\_words = ["but", "quirks"]

# Replace characters

for old, new in replace\_characters.items():

sentence = sentence.replace(old, new)

# Split the cleaned sentence and filter words

filtered\_words = [word for word in sentence.split() if word not in filter\_words]

# Join the result into a string

final\_sentence = " ".join(filtered\_words)

print(final\_sentence) # Output: "Hello world Python is great has its"

8. Remove Duplicate Words and Filter

You may want to remove duplicate words and filter specific words out.

python

Copy code

sentence = "Python is fun fun but sometimes sometimes tricky tricky"

filter\_words = ["but"]

# Split sentence and remove duplicates by converting to set, then filter

words = sentence.split()

unique\_words = []

[unique\_words.append(word) for word in words if word not in unique\_words]

# Filter unwanted words

filtered\_words = [word for word in unique\_words if word not in filter\_words]

# Rejoin the words into a string

final\_sentence = " ".join(filtered\_words)

print(final\_sentence) # Output: "Python is fun sometimes tricky"

----------------------------------------------

pyutils:

Basic datetime Objects

python

Copy code

from datetime import datetime

# Get current date and time

now = datetime.now()

print(now) # Output: 2024-09-25 12:34:56.789123 (example)

# Get just the current date

today = now.date()

Formatting Date and Time

You can format datetime objects into readable strings using strftime().

python

Copy code

# Format current date and time

formatted = now.strftime('%Y-%m-%d %H:%M:%S')

print(formatted) # Output: 2024-09-25 12:34:56

Common strftime() format codes:

%Y: Year (e.g., 2024)

%m: Month (01-12)

%d: Day of the month (01-31)

%H: Hour (00-23)

%M: Minute (00-59)

%S: Second (00-59)

Timedelta: Differences Between Dates

The timedelta object represents the difference between two dates or times.

python

Copy code

from datetime import timedelta

# Create a timedelta of 7 days

delta = timedelta(days=7)

# Add or subtract timedelta to a date

next\_week = now + delta

print(next\_week) # Output: 2024-10-02 12:34:56

# Subtracting dates

date\_diff = next\_week - now

print(date\_diff) # Output: 7 days, 0:00:00

Extracting Date and Time Components

You can access specific parts of a datetime object.

python

Copy code

print(now.year) # Output: 2024

print(now.month) # Output: 9

print(now.day) # Output: 25

print(now.hour) # Output: 12

print(now.minute) # Output: 34

print(now.second) # Output: 56

Handling Time

The time class represents time objects without a date.

python

Copy code

from datetime import time

# Create a time object

t = time(14, 30, 0)

print(t) # Output: 14:30:00

Handling Only Date

The date class can be used for operations involving only dates.

python

Copy code

from datetime import date

# Today's date

today = date.today()

print(today) # Output: 2024-09-25

# Create a specific date

d = date(2024, 12, 31)

print(d) # Output: 2024-12-31

Getting the Day of the Week

You can get the day of the week in both numeric and string form.

python

Copy code

# Monday is 0, Sunday is 6

print(now.weekday()) # Output: 2 (Wednesday)

# Get the day name

day\_name = now.strftime("%A")

print(day\_name) # Output: Wednesday

Handling datetime in Pandas

Pandas offers excellent support for datetime operations, which is particularly useful for data analysis.

python

Copy code

import pandas as pd

# Convert a date column to datetime

data = {'date': ['2024-09-25', '2024-09-26', '2024-09-27']}

df = pd.DataFrame(data)

df['date'] = pd.to\_datetime(df['date'])

print(df)

Adding or Subtracting Days in Pandas

python

Copy code

# Add 7 days to each date in the column

df['new\_date'] = df['date'] + pd.DateOffset(days=7)

print(df)

. Calculating Date Differences in Pandas

python

Copy code

# Calculate the difference in days between two dates

df['days\_diff'] = (df['new\_date'] - df['date']).dt.days

print(df)

Formatting Dates in Pandas

You can format a datetime column in Pandas the same way you do with strftime.

python

Copy code

# Convert datetime to a string in the desired format

df['formatted\_date'] = df['date'].dt.strftime('%Y/%m/%d')

print(df)

Handling Missing Dates

If your dataset has missing or null dates, you can use pd.NaT to fill or check for missing datetime values.

python

Copy code

# Fill missing dates with a default date

df['date\_filled'] = df['date'].fillna(pd.Timestamp('2024-01-01'))

# Check for missing dates

missing\_dates = df['date'].isna()

print(missing\_dates)

Time Differences Between Rows

You can calculate the time difference between two datetime columns or rows.

python

Copy code

# Create a new column to simulate a second datetime

df['date2'] = df['date'] + pd.DateOffset(days=3)

# Calculate the difference between 'date2' and 'date' in days

df['date\_diff'] = (df['date2'] - df['date']).dt.days

print(df)

------------------------------------------

regex:

Basic Regex Functions in Python

The main functions provided by the re module are:

re.search(): Search for a match anywhere in the string.

re.match(): Match the regex pattern at the beginning of the string.

re.findall(): Find all substrings that match the pattern.

re.sub(): Replace occurrences of the pattern with something else.

import re

# re.search() example

result = re.search(r'\d+', 'The number is 123')

print(result.group()) # Output: 123

# re.match() example

result = re.match(r'\w+', 'Hello world')

print(result.group()) # Output: Hello (matches at the beginning)

# re.findall() example

result = re.findall(r'\d+', 'The numbers are 123 and 456')

print(result) # Output: ['123', '456']

# re.sub() example (replace all digits with '\*')

result = re.sub(r'\d+', '\*', 'The number is 123')

print(result) # Output: The number is \*

Special Regex Characters

Here are a few common regex symbols:

Character Description Example

. Matches any single character h.t → hat, hot, hit

\d Matches any digit (0-9) \d → 7, 123

\w Matches any word character (A-Z, a-z, 0-9) \w → abc123

\s Matches any whitespace character \s → (space)

\* Matches 0 or more occurrences a\* → a, aa, aaa

+ Matches 1 or more occurrences a+ → a, aa, aaa

[] Matches any character inside brackets [abc] → a, b, c

^ Matches the start of the string ^Hello → Matches "Hello" at the beginning

$ Matches the end of the string end$ → Matches "end" at the end

. Regex Groups

Groups allow you to extract specific parts of a match.

python

Copy code

result = re.search(r'(\d{3})-(\d{2})-(\d{4})', 'Phone: 123-45-6789')

print(result.group(1)) # Output: 123 (first group)

print(result.group(2)) # Output: 45 (second group)

print(result.group(3)) # Output: 6789 (third group)

Filtering Rows Using Regex

You can use str.contains() or str.match() to filter rows in pandas DataFrames using regex.

python

Copy code

import pandas as pd

# Create a sample DataFrame

data = {'name': ['Alice', 'Bob', 'Charlie', 'David'],

'email': ['alice123@gmail.com', 'bob\_smith@yahoo.com', 'charlie@company.org', 'david@mycompany.com']}

df = pd.DataFrame(data)

# Filter rows where email ends with 'gmail.com'

filtered\_df = df[df['email'].str.contains(r'gmail\.com$')]

print(filtered\_df)

Extracting Substrings Using Regex

Use str.extract() to pull out matching substrings from columns.

python

Copy code

# Extract domain from email

df['domain'] = df['email'].str.extract(r'@([a-zA-Z0-9.-]+)')

print(df)

5.3 Replacing Substrings Using Regex

You can replace patterns using str.replace().

python

Copy code

# Replace numbers in the email with an empty string

df['clean\_email'] = df['email'].str.replace(r'\d+', '', regex=True)

print(df)

Splitting Columns Based on Regex

You can use str.split() to split a column based on a regex pattern.

python

Copy code

# Split the email into two columns: username and domain

df[['username', 'domain']] = df['email'].str.split('@', expand=True)

print(df)

6.2 Advanced Filtering with Regex

You can use more complex regex patterns to filter rows.

python

Copy code

# Filter rows where email contains a digit

filtered\_df = df[df['email'].str.contains(r'\d')]

print(filtered\_df)

# Filter rows where email contains either 'gmail' or 'yahoo'

filtered\_df = df[df['email'].str.contains(r'gmail|yahoo')]

print(filtered\_df)