Day 3: Advanced Python Data Handling

- String Methods: Case conversion, content checking, manipulation
- List Operations: Adding, removing elements, sorting, copying
- Common Operations: Shared methods between strings, lists, tuples, and sets
- Tuples vs. Lists: Understanding immutability and mutability
- Variable Assignment: Multiple assignments, unpacking techniques
- List Comprehension: Filtering, transforming, and classifying data
- Dictionaries: Basics, iteration, nested structures, comprehension
- Inventory Management: Product additions, updates, and enhancements
- Data Filtering and Manipulation: Guest list filtering, employee updates
- Copying Techniques: Dictionaries and lists with unpacking operators

Summarized String Methods

Case Conversion

- `capitalize()`: Uppercase first character.
- `lower()`: Lowercase all characters.
- `swapcase()`: Swap case of each character.
- `title()`: Uppercase the first character of each word.
- `upper()`: Uppercase all characters.

Checking Content

- `endswith(suffix[, start[, end]])`: Checks if string ends with suffix.
- `startswith(prefix[, start[, end]])`: Checks if string starts with prefix.
- `islower()`: Checks for all lowercase characters.
- `isdigit()`: Checks for all digit characters.

Manipulation and Transformation

- `lstrip([chars])`: Removes leading characters.
- `rstrip([chars])`: Removes trailing characters.
- `strip([chars])`: Removes leading and trailing characters.
- `split(sep=None, maxsplit=-1)`: Splits string.
- `join(iterable)`: Joins elements with string as separator.
- `replace(old, new[, count])`: Replaces occurrences of substring.

Searching and Replacing

- `count(sub[, start[, end]])`: Counts substring occurrences.
- `find(sub[, start[, end]])`: Searches for substring, returns index or `-1`.
- `index(sub[, start[, end]])`: Searches for substring, raises `ValueError` if not found.

List Methods

Adding Elements

- `append(x)`: Adds item to end.
- `extend(iterable)`: Appends items from iterable.
- `insert(i, x)`: Inserts item at position.

Removing Elements

- `remove(x)`: Removes first occurrence of `x`.
- `pop([i])`: Removes and returns item at position.
- `clear()`: Clears list.

Finding and Counting

- `index(x[, start[, end]])`: Returns index of first occurrence of `x`.
- `count(x)`: Counts occurrences of `x`.

Sorting and Reversing

- `sort(*, key=None, reverse=False)`: Sorts items.
- `reverse()`: Reverses items.

Copying

- copy(): Returns shallow copy.

Miscellaneous

- `len(list)`: Returns item count.
- `list[i]`: Accesses item.
- `list[i] = x`: Sets item.
- `del list[i]`: Removes item.
- `list[i:j]`: Returns slice.
- `list[i:j] = t`: Replaces slice.
- `del list[i:j]`: Removes slice.
- `list += t`: Extends list with `t`.
- `list * n`: Repeats list n times.

Common methods between String and List

#	Operation/Method	String Example	List Example
1.	Indexing	`my_str[0]`	`my_list[0]`
2.	Slicing	`my_str[1:3]`	`my_list[1:3]`
3.	+ (Concatenation)	`my_str1 + my_str2`	`my_list1 + my_list2`
4.	* (Repetition)	`my_str * 2`	`my_list * 2`
5.	`len()`	`len(my_str)`	`len(my_list)`
6.	`in` (Membership Test)	`'a' in my_str`	`3 in my_list`
7.	Iteration (`for` loop)	`for char in my_str:`	`for item in my_list:`

Understanding Tuples vs. Lists in Python

Tuples: Immutable Sequences

```python
# Introducing tuples as immutable collections
person = ("Alex", "SA", 20, 20)
print(person, type(person))
# Attempting to modify a tuple will result in an error
# person[0] = "Thato"  # Uncommenting this line will raise a TypeError

# Tuples support operations like count and index
print(person.count(20))  # Counts the occurrences of 20
print(person.index("Alex"))  # Finds the index of "Alex"
```

Tuples: Immutable Sequences

```
# Exploring the index method in tuples
# The index method returns the first index of a value
# If the value is not found, a ValueError is raised

# Unlike some other languages or data structures, there's no 'find' method that returns -1 if not found
# print(person.index(80))  # Uncommenting this will raise a ValueError as 80 is not in the tuple
```

Lists: Mutable Sequences

```
# Highlighting the mutability of lists
people_list = ["Alex", "SA", 20, 20]
print(people_list, type(people_list))
# Lists allow modification of their elements
people_list[0] = "Thato"
print(people_list)  # Shows the updated list with "Thato" replacing "Alex"
```

Error Handling with Tuple Methods

```
# Demonstrating error handling for tuple methods
try:
    print(person.index(80))
except ValueError as e:
    print(f"Error: {e}")  # Showcases handling the error when value is not found
```

Common Operations between List and Tuple

#	Operation/Method	List Example	Tuple Example
1	Indexing	`my_list[0]`	`my_tuple[0]`
2	Slicing	`my_list[1:3]`	`my_tuple[1:3]`
3	+ (Concatenation)	`my_list1 + my_list2`	`my_tuple1 + my_tuple2`
4	* (Repetition)	`my_list * 2`	`my_tuple * 2`
5	`len()`	`len(my_list)`	`len(my_tuple)`
6	`in` (Membership Test)	`3 in my_list`	`'a' in my_tuple`
7	Iteration (for loop)	`for item in my_list:`	`for item in my_tuple:`
8	`count()`	`my_list.count('a')`	`my_tuple.count('a')`
9	`index()`	`my_list.index('a')`	`my_tuple.index('a')`

Common Operations between List and Set

#	Operation/Method	List Example	Set Example
1	`len()`	`len(my_list)`	`len(my_set)`
2	`in` (Membership Test)	`'a' in my_list`	`'a' in my_set`
3	Iteration (`for` loop)	`for item in my_list:`	`for item in my_set:`
4	Conversion	`my_set = set(my_list)`	`my_list = list(my_set)`

Common Operations between String, List, Tuple and Set

SNo.	Operation/Method	Applicable To	Example
1	Indexing	String, List, Tuple	`my_seq[0]`
2	Slicing	String, List, Tuple	`my_seq[1:3]`
3	+ (Concatenation)	String, List, Tuple	`seq1 + seq2`
4	* (Repetition)	String, List, Tuple	`my_seq * 2`
5	`len()`	All	`len(my_seq)`
6	`in` (Membership Test)	All	`'a' in my_seq`
7	Iteration (`for` loop)	All	`for item in my_seq:`
8	`count()`	String, List, Tuple	`my_seq.count('a')`
9	`index()`	String, List, Tuple	`my_seq.index('a')`
10	Conversion	List ⇔ Set	`set(my_list)`, `list(my_set)`

Variable Assignment and Unpacking

Efficient Variable Assignment

```python
# Assigning the same value to multiple variables in a single line
a = b = c = 10
print(a, b, c)  # Outputs: 10 10 10

# Assigning different values to multiple variables in a single line
lilitha, Dhara, Quinlan = "Chocolate", "Lime", 'Vanilla'
print(lilitha, Dhara, Quinlan)  # Outputs: Chocolate Lime Vanilla
```

Exploring Unpacking

```
# Unpacking a list into variables
movies = ["Remember the Titans", "Juno", "Lucy"]
caleb, gemma, yolanda = movies
print(gemma)  # Outputs: Juno
```

Basic Unpacking from a List

Exploring Unpacking

```
# Skipping items during unpacking using underscore (_)
t1, t2, _, _ = [100, 200, 300, 400]
print(t1, t2)  # Outputs: 100 200

# Skipping selective items
t1, _, t2, t3 = [100, 200, 300, 400]
print(t1, t2, t3)  # Outputs: 100 300 400
```

Skipping Items with Underscore

Exploring Unpacking

```
# Using asterisk (*) to capture remaining items
t1, t2, *t3 = (100, 200, 300, 400, 60, 40, 30)
print(t1, t2, t3)  # Outputs: 100 200 [300, 400, 60, 40, 30]
```

Unpacking with Asterisk

Calculating Distances

```python
# Calculating distances from the origin without unpacking
coordinates = [(5, 4), (1, 1), (6, 10), (9, 10)]
distances = [(x**2 + y**2) ** 0.5 for x, y in coordinates]
print(distances)  # Inline demonstration of Task 3's final approach
```

Calculating Distances

```
# Utilizing unpacking for clearer and more concise code
distances = []
for cord in coordinates:
  x, y = cord
  distances.append((x**2 + y**2) ** 0.5)

print(distances)  # Same output as Task 1, with unpacking for clarity
```

Calculating Distances

```
# Streamlining the calculation with unpacking and list comprehension
distances = [(x**2 + y**2) ** 0.5 for x, y in coordinates]
print(distances)  # Showcases the elegance and efficiency of Python's list comprehension
```

Classifying Numbers as Even or Odd with List Comprehension

Introduction to List Comprehension

```
# List comprehension provides a concise way to create lists.
# It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.
# The expressions can be anything, meaning you can put in all kinds of objects in lists.

# The basic syntax is:
# [expression for item in list if conditional]
```

The Task: Even or Odd Classification

```
# Given a list of numbers, classify each number as 'Even' or 'Odd'.
numbers = [8, 5, 7, 4, 6, 2]

# The goal is to produce a new list indicating the classification.
```

Solution Without List Comprehension

```python
# Traditional loop-based approach to classify numbers
numbers_copy = []
for num in numbers:
  numbers_copy.append("Even" if num % 2 == 0 else "Odd")
print(numbers_copy)  # Output: ['Even', 'Odd', 'Odd', 'Even', 'Even', 'Even']
```

Solution With List Comprehension

```
# Simplifying the task with list comprehension
result = ["Even" if num % 2 == 0 else "Odd" for num in numbers]
print(result)  # Output: ['Even', 'Odd', 'Odd', 'Even', 'Even', 'Even']
```

Filtering and Transforming Lists with Comprehension

Introduction to Filtering with List Comprehension

```
# List comprehension can be used not just for transformation but also for filtering.
# The syntax includes an 'if' condition to filter items from the original list based on a criterion.
```

The Task: Filtering Avenger Names

```
# Given a list of Avenger names, the goal is to filter out names based on their length.
avengers = [
    "Hulk",
    "Iron man",
    "Black widow",
    "Captain america",
    "Spider man",
    "Thor",
]

# Specifically, we want to select names where the length is greater than 10 characters.
```

Traditional Approach

```
# Traditional method using for-loop and if-statement
filtered_names = []
for avenger in avengers:
  if len(avenger) > 10:
    filtered_names.append(avenger)

print(filtered_names)  # Expected Output: ['Black widow', 'Captain america', 'Spider man']
```

List Comprehension for Filtering

```python
# Achieving the same result with list comprehension
filtered_names = [avenger for avenger in avengers if len(avenger) > 10]
print(filtered_names)  # Output: ['Black widow', 'Captain america', 'Spider man']
```

Adding Transformation

```python
# Extending list comprehension to include transformation
# Here, filtering names longer than 10 characters and converting them to uppercase
filtered_names1 = [
    avenger.upper() for avenger in avengers if len(avenger) > 10
]
print(filtered_names, filtered_names1)  # Output: ['Black widow', 'Captain america', 'Spider man'] ['BLACK WIDOW', 'CAPTAIN AMERICA', 'SPIDER MAN']
```

Mastering Python Dictionaries

An in-depth exploration of dictionaries in Python: from basics to advanced operations and dictionary comprehension.

Introduction to Dictionaries

```python
# Dictionaries in Python are collections of key-value pairs.
person = {
    "name": "Siya Kolisi",
    "age": 32,
    "country": "South Africa",
    "sport": "Rugby"
}

# Accessing and printing a dictionary and its type
print(person, type(person))
print(person["name"])  # Accessing a value by its key
```

Updating Dictionary Values

```
# Updating dictionary values
person['age'] += 1  # Incrementing age
print(person)
```

Updating Dictionary Values

```
# Exploring dictionary methods for keys, values, and items
print(person.keys())   # Outputs the keys
print(person.values())  # Outputs the values
print(person.items())  # Outputs key-value pairs as tuples
```

Iterating Over Dictionaries

```
# Looping through a dictionary's items
for key, value in person.items():
    print(key, value)  # Prints each key-value pair
```

Safely Accessing Values

```
# Using .get() for safe access to dictionary values
print(person.get('height'))  # Returns None if 'height' is not a key
print(person.get('height', 175))  # Returns 175 as a default value if 'height' is not found
```

Nested Dictionaries

```
# Working with nested dictionaries
print(person['address']['city'])  # Accessing a nested value
```

Nested Dictionaries

```
# Avoiding errors with nested missing data
print(person.get('stats', {}).get('points', 'No points data'))
```

Dictionary Comprehension

```python
# Creating a new dictionary with comprehension
nums = {x: x**2 for x in range(10) if x % 2 == 0}
print(nums)
```

Inventory Management

Working with Book Inventories

```
# Task 1: Filtering books based on rating
books = [
    {"title": "Infinite Jest", "rating": 4.5, "genre": "Fiction"},
    {"title": "The Catcher in the Rye", "rating": 3.9, "genre": "Fiction"},
    {"title": "Sapiens", "rating": 4.9, "genre": "History"},
    {"title": "A Brief History of Time", "rating": 4.8, "genre": "Science"},
    {"title": "Clean Code", "rating": 4.7, "genre": "Technology"},
]
```

Identifying Highly Rated Books

Working with Book Inventories

```
# Using list comprehension for concise filtering
top_rated_books = [book['title'] for book in books if book['rating'] >= 4.7]
print(top_rated_books)
```

Working with Book Inventories

```
# Task 3: Sorting the filtered list of book titles
sorted_top_rated_books = sorted(top_rated_books)
print(sorted_top_rated_books)
```

Mini Inventory Management System

```
# Adding new products to the inventory
inventory = [
    {"name": "Apple", "quantity": 30, "price": 0.50},
    {"name": "Banana", "quantity": 20, "price": 0.20},
]
```

Mini Inventory Management System

```
# Simulating user input for a new product
product_name = "Orange"
quantity = 10
price = 0.60

new_product = {"name": product_name, "quantity": quantity, "price": price}
inventory.append(new_product)
print(inventory)
```

Mini Inventory Management System

```
# Task 2: Updating quantity and price if the product exists
for item in inventory:
  if (item['name'] == product_name):
    item['quantity'] += quantity
    item['price'] = price
    break
else:
  inventory.append(new_product)

print(inventory)
```

Enhancing Code Quality

```
# Utilizing for...else for efficient inventory updates
product_name = "Apple"
quantity = 40
price = 0.4
```

Enhancing Code Quality

```
for item in inventory:
  if (item['name'] == product_name):
    item['quantity'] += quantity
    item['price'] = price
    break
else:
  inventory.append({"name": product_name, "quantity": quantity, "price": price})

print(inventory)
```

Advanced Data Filtering and Manipulation in Python

Filtering Guest Lists

```
# Filtering guests based on age, code, and blacklist
guests = [
    {"name": "Alice", "age": 25, "code": "VIP123"},
    {"name": "Bob", "age": 17, "code": "VIP123"},
    {"name": "Charlie", "age": 30, "code": "VIP123"},
    {"name": "Dave", "age": 22, "code": "GUEST"},
    {"name": "Eve", "age": 29, "code": "VIP123"}
]
```

Filtering Guest Lists

```
blacklist = ["Dave", "Eve"]
PASS_CODE = "VIP123"

guestlist = [guest['name'] for guest in guests if guest['age'] >= 21
             and guest['code'] == PASS_CODE and guest['name'] not in blacklist]
print(guestlist)  # Outputs filtered guest names
```

Initial Filtering Logic

Enumerating Lists

```python
# Demonstrating enumerate for list indexing
nums = [90, 50, 80]
for idx, num in enumerate(nums):
    print(idx, num)  # Prints index and number
```

Using enumerate for Indexing

Employee Experience Update

```
# Incrementing experience and updating status based on years
employees = [
    {"name": "Alex", "experience": 2},
    {"name": "Gemma"},
    {"name": "Rashay", "experience": 4},
    {"name": "Thato"}
]
```

Employee Experience Update

```
for employee in employees:
    employee['experience'] = employee.get('experience', 0) + 1
    # Update status based on experience
    experience = employee['experience']
    if experience >= 5:
        employee['status'] = "Senior"
    elif 3 <= experience < 5:
        employee['status'] = "Mid-Level"
    else:
        employee['status'] = "Junior"

print(employees)
```

Dictionary and List Copying Techniques

```python
# Copying and merging dictionaries with unpacking
movie = {"name": "Mr Bones", "year": 2001}
detail = {"actor": "Leon Schuster", "director": "Gray Hofmeyr"}

# Merging two dictionaries
movie_details = {**movie, **detail}
print(movie_details)  # Merged dictionary

# Copying and extending lists with unpacking
t1 = [80, 90]
t2 = [50, 60]
t3 = [*t1, *t2]  # Combines t1 and t2 into t3
print(t3)  # Combined list
```

That's all folks 🙋🏽‍♂️