Day 4: Advanced Python Data Structures and Functions

- Tuples vs. Lists: Immutability of tuples, list modification capabilities.
- Sets: Unique elements, adding/updating elements, set operations.
- Unpacking Techniques: Skipping items, advanced unpacking with tuples and lists.
- Set Comprehension: Filtering and set operations for data handling.
- Python Functions: Basic definitions, default parameters, keyword/positional arguments.
- Library System Tasks: Adding books, searching by author, book checkout functionality.
- Lambda Functions: Introduction, mapping, squaring values, high-order functions.
- Sequence Manipulation: Utilizing `map`, `filter`, and sequence functions like `sum`, `max`, `min`.

Tuples vs. Lists in Python

Exploring the distinctions between tuples and lists in Python, emphasizing immutability, utility methods, and unpacking techniques.

Tuples: Immutable Sequences

```python
person = ("Alex", "SA", 20, 20)
print(person, type(person))
# Trying to change a tuple value (e.g., person[0] = "Thato") raises an error.
print(person.count(20))  # Count occurrences of 20
print(person.index("Alex"))  # Find the index of "Alex"
# Attempting to find an index not present (e.g., person.index(80)) raises a ValueError.
```

Lists: Mutable Sequences

```
# Lists, being mutable, allow modifications to their elements.
tech_gadgets_list = ['Smartphone', "Laptop", "Smartwatch", "Tablet", "Tablet"]
print(tech_gadgets_list)
```

Lists Allow Modification

Sets: Unique Elements

```
tech_gadgets = {'Smartphone', "Laptop", "Smartwatch", "Tablet", "Tablet"}
print(tech_gadgets)
# Demonstrating set operations: addition and update
tech_gadgets.add('E-reader')
tech_gadgets.update(['Drone', "Selfiestick"])
print(tech_gadgets)
# Demonstrating safe removal with discard (removes 'Drone' without errors)
tech_gadgets.discard('Drone')
print(tech_gadgets)
```

Sets Ensure Uniqueness and Lack Order

Skipping Items with Unpacking

```
# Skipping multiple items using unpacking and storing the last item separately
t1, t2, *_, t3 = [100, 200, 300, 400, 60, 40, 30]
print(t1, t2, t3)
# Applying the same technique to a tuple
t1, t2, *z1, t3 = (100, 200, 300, 400, 60, 40, 30)
print(t1, t2, t3)
```

Python Set Operations and Set Comprehension

A comprehensive guide to manipulating sets in Python and leveraging set comprehension for efficient data handling.

Introduction to Sets

```
tech_gadgets = {'Smartphone', "Laptop", "Smartwatch", "Tablet", "Tablet"}
print(tech_gadgets)
```

Adding Elements to a Set

```
tech_gadgets.add('E-reader')
print(tech_gadgets)
```

Updating Sets with Multiple Elements

```
more_gadgets = ['Drone', "Selfiestick"]
tech_gadgets.update(more_gadgets)
print(tech_gadgets)
```

Deleting Elements from Sets

```
tech_gadgets.discard('Drone')
tech_gadgets.discard('Gimble')  # 'Gimble' does not exist, but no error will be raised
print(tech_gadgets)
```

Safe Deletion from Sets Using discard

Set Operations: Intersection, Union, and Difference

```
outdoor_activities = {'Hiking', 'Cycling', 'Swimming'}
indoor_activities = {'Gaming', 'Reading', 'Cycling'}
print(indoor_activities.intersection(outdoor_activities))  # Intersection
print(indoor_activities.union(outdoor_activities))  # Union
print(indoor_activities.difference(outdoor_activities))  # Difference
```

Performing Intersection, Union, and Difference Operations

Simplifying Tasks with Set Comprehension

```
activity_gadgets = {
    'Smartwatch': 'Hiking',
    'VR Headset': 'Gaming',
    'Smartphone': 'Reading',
    'Drone': 'Cycling'
}

def getGadgetsSet(activity_gadgets, activities):
  return {gadget for gadget, activity in activity_gadgets.items() if activity in activities}

print(getGadgetsSet(activity_gadgets, indoor_activities))
print(getGadgetsSet(activity_gadgets, outdoor_activities))
```

Using Set Comprehension for Filtering Gadgets

Python Functions: Simplifying and Abstracting Logic

A step-by-step exploration of defining and utilizing Python functions for more readable and maintainable code.

Basic Function Definition and Usage

```
def sum(a, b):
  return a + b

print("The sum is: ", sum(30, 50))
print("The sum is: ", sum(50, 40))
```

Defining and Using a Simple Sum Function

Implementing Default Parameters

```
def driving_test(name, age=15, car="Toyota Tazz"):
  if age >= 18:
    return f"{name} eligible for driving. You will be tested with {car}"
  else:
    return "Try again after a few years 👶🍼"

print(driving_test("Gemma"))
print(driving_test(age=21, name="Dhara", car="GR Corolla"))
```

Driving Test Function with Default Parameters

Keyword and Positional Arguments

```
print(driving_test("Gemma", 20, "GR Corolla"))  # Positional arguments
print(driving_test(age=21, name="Dhara"))  # Keyword arguments
```

Demonstrating Keyword and Positional Argument Usage

Managing Library System with Python

Illustrating Python's capabilities in handling library system tasks like adding books, searching by author, and checking out books.

Adding a New Book to the Library

```python
def add_book(library, new_book):
  library.append(new_book)

# Example book to add
book = {
    "title": "Fluent Python II",
    "author": "Alex",
    "year": 2016,
    "available": True
}

# Adding the book to the library
add_book(library_list, book)
```

Function to Add a New Book

Searching Books by Author

```
def search_by_author(library, author_name):
  return [book for book in library if author_name == book["author"]]

# Searching for books by 'Mark Lutz'
search_results = search_by_author(library_list, 'Mark Lutz')
print(search_results)
```

Function to Search Books by Author

Checking Out a Book

```python
def check_out_book(library, title):
  for book in library:
    if (book['title'] == title):
      if (book['available']):
        book['available'] = False
        return f"Yes, {title} is available and successfully checked out ✅"
      else:
        return f"No, {title} is unavailable ❌"

  return f"{title} is not present in the library ☠️"

# Checking out 'Fluent Python' and attempting to check out 'Hardy Boys'
print(check_out_book(library_list, "Fluent Python"))
print(check_out_book(library_list, "Hardy Boys"))
print(check_out_book(library_list, "Learning Python I"))
```

```
        if (book['title'] == title):
            if (book['available']):
                book['available'] = False
                return f"Yes, {title} is available and successfully checked out ✅"
            else:
                return f"No, {title} is unavailable ❌"

    return f"{title} is not present in the library ☠️"

# Checking out 'Fluent Python' and attempting to check out 'Hardy Boys'
print(check_out_book(library_list, "Fluent Python"))
print(check_out_book(library_list, "Hardy Boys"))
print(check_out_book(library_list, "Learning Python I"))
```

Exploring Python Functions and Sequences

A concise guide to lambda functions, high-order function patterns, and working with sequences in Python.

Introduction to Lambda Functions

```
add = lambda a, b: a + b
print(add(6, 7))
```

Mapping with Lambda Functions

```
player_stats = [10, 30, 60]
boosted_stats = map(lambda x: x * 2, player_stats)
print(list(boosted_stats))
```

Using map with Lambda for List Transformation

Lambda Functions for Squaring Values

```
super_boosted_stats = map(lambda x: x * x, [10, 30, 60])
print(list(super_boosted_stats))
```

Squaring List Values with Lambda and map

Utilizing High-Order Functions

```
def map_own(fn, arr):
  return [fn(val) for val in arr]

print(map_own(lambda x: x * 2, [10, 30, 60]))
```

Implementing Currying with Lambda

```
mul = lambda x: (lambda y: x * y)
mul_5 = mul(5)
print(mul_5(10))
```

High-Order Functions in Use: map and filter

```
result1 = map(lambda x: x * 2, [10, 30, 60])
result2 = filter(lambda x: x > 10, [10, 50, 60, 100, 6, 8, 30])
print(list(result1))
print(list(result2))
```

Applying map and filter Functions

Pythonic Ways with Sequences

```
print(sum([10, 30, 60]))
print(max([10, 80, 30, 60]))
print(min([10, -1, 30, 60]))
print(all([10, 0, 30, -1]))
```

That's all folks 🙋‍♂️