**Python Basics Guided Exercises: Functions, Classes, and OOP Concepts**

This student handbook contains guided exercises that will help you understand and apply Python’s key concepts, including functions, classes, and object-oriented programming (OOP). By following these exercises, you will build a solid foundation for your Python skills.

**1. Introduction to Functions**

A **function** is a reusable block of code that performs a specific task. Functions are used to avoid repetition and to organize code in a more readable way.

**Exercise 1: Basic Function**

Create a function greet() that prints out "Hello, welcome to Python!" when called.

def greet():

print("Hello, welcome to Python!")

# Call the function

greet()

**Objective:**

* Understand how to define and call a function.

**Exercise 2: Function with Parameters**

Modify the greet() function to accept a name as a parameter and greet the user by name.

def greet(name):

print(f"Hello, {name}! Welcome to Python!")

# Call the function with an argument

greet("Alice")

**Objective:**

* Learn how to pass parameters to functions.

**Exercise 3: Function with Return Values**

Write a function add\_numbers() that takes two numbers as parameters and returns their sum.

def add\_numbers(a, b):

return a + b

# Call the function and print the result

result = add\_numbers(3, 5)

print(result)

**Objective:**

* Understand how to return a value from a function.

**2. Introduction to Classes and Objects**

In Python, **classes** are templates for creating objects. **Objects** are instances of classes, and classes encapsulate both data (attributes) and methods (functions).

**Exercise 4: Creating a Class**

Create a class Car with two attributes: make and model. Add a method display\_info() that prints the car's make and model.

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

def display\_info(self):

print(f"This car is a {self.make} {self.model}.")

# Create an object of the Car class

my\_car = Car("Toyota", "Corolla")

# Call the method on the object

my\_car.display\_info()

**Objective:**

* Understand how to define a class and create an object.

**Exercise 5: Using \_\_init\_\_ Method (Constructor)**

In the Car class, add a constructor method \_\_init\_\_() to initialize the make and model of the car when a new object is created.

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

# Create an object of the Car class

my\_car = Car("Honda", "Civic")

# Access attributes directly

print(f"My car is a {my\_car.make} {my\_car.model}.")

**Objective:**

* Learn how to initialize object attributes using the constructor.

**3. Object-Oriented Programming (OOP) Concepts**

**Exercise 6: Inheritance**

Create a class ElectricCar that inherits from the Car class. Add an additional attribute battery\_size and a method describe\_battery() that prints the battery size.

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

def display\_info(self):

print(f"This car is a {self.make} {self.model}.")

class ElectricCar(Car):

def \_\_init\_\_(self, make, model, battery\_size):

super().\_\_init\_\_(make, model)

self.battery\_size = battery\_size

def describe\_battery(self):

print(f"This car has a {self.battery\_size}-kWh battery.")

# Create an ElectricCar object

my\_electric\_car = ElectricCar("Tesla", "Model S", 75)

# Call methods from both the ElectricCar and the parent Car classes

my\_electric\_car.display\_info()

my\_electric\_car.describe\_battery()

**Objective:**

* Understand inheritance, where one class inherits properties and methods from another class.

**Exercise 7: Encapsulation (Private Attributes)**

Encapsulation is the concept of restricting access to certain details of an object. In Python, private attributes can be defined using an underscore.

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

self.\_\_mileage = 0 # Private attribute

def drive(self, miles):

self.\_\_mileage += miles

def get\_mileage(self):

return self.\_\_mileage

# Create a Car object

my\_car = Car("Ford", "Mustang")

# Attempting to access private attribute directly will result in an error

# print(my\_car.\_\_mileage) # Uncommenting this will raise an AttributeError

# Using the method to access the mileage

my\_car.drive(100)

print(f"Mileage: {my\_car.get\_mileage()} miles.")

**Objective:**

* Learn about encapsulation and how to create private attributes.

**Exercise 8: Polymorphism**

Polymorphism allows different classes to define the same method, but with different behaviors.

class Dog:

def sound(self):

print("Woof!")

class Cat:

def sound(self):

print("Meow!")

# Polymorphism in action

animals = [Dog(), Cat()]

for animal in animals:

animal.sound() # The appropriate method is called for each animal

**Objective:**

* Understand polymorphism and how objects of different classes can implement the same method with different behaviors.

**4. Putting It All Together**

**Exercise 9: Car Dealership System**

Create a simple **CarDealership** class that stores a list of Car objects. Implement methods to add a car, remove a car, and list all cars available in the dealership.

class Car:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

def display\_info(self):

print(f"{self.make} {self.model}")

class CarDealership:

def \_\_init\_\_(self):

self.inventory = []

def add\_car(self, car):

self.inventory.append(car)

def remove\_car(self, car):

if car in self.inventory:

self.inventory.remove(car)

else:

print("Car not found.")

def list\_cars(self):

if not self.inventory:

print("No cars in inventory.")

else:

for car in self.inventory:

car.display\_info()

# Create dealership and cars

dealership = CarDealership()

car1 = Car("Ford", "Focus")

car2 = Car("Toyota", "Camry")

# Add cars to inventory

dealership.add\_car(car1)

dealership.add\_car(car2)

# List all cars in the dealership

dealership.list\_cars()

# Remove a car from inventory

dealership.remove\_car(car1)

# List again to see the updated inventory

dealership.list\_cars()

**Objective:**

* Combine all OOP concepts: classes, objects, methods, and inheritance.

implement a CRUD (Create, Read, Update, Delete) API for a shopping cart system using **FastAPI**. This example assumes that we are working with a list of items in the cart, where each item has an ID, name, quantity, and price.

**Step 1: Install Dependencies**

To get started, you'll need FastAPI and Uvicorn.

bash

Copy

pip install fastapi uvicorn

**Step 2: Create the FastAPI application**

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from typing import List

# Initialize FastAPI app

app = FastAPI()

# Data models using Pydantic

class CartItem(BaseModel):

item\_id: int

name: str

quantity: int

price: float

class ShoppingCart(BaseModel):

student\_id: int

items: List[CartItem]

# In-memory storage (could be replaced with a database)

carts\_db = {}

# CRUD Operations

@app.post("/cart/{student\_id}/add\_item/")

def add\_item\_to\_cart(student\_id: int, item: CartItem):

if student\_id not in carts\_db:

carts\_db[student\_id] = ShoppingCart(student\_id=student\_id, items=[])

# Check if item already exists in cart

for existing\_item in carts\_db[student\_id].items:

if existing\_item.item\_id == item.item\_id:

existing\_item.quantity += item.quantity

return {"message": "Item quantity updated successfully"}

carts\_db[student\_id].items.append(item)

return {"message": "Item added to cart successfully"}

@app.get("/cart/{student\_id}/", response\_model=ShoppingCart)

def get\_cart(student\_id: int):

if student\_id not in carts\_db:

raise HTTPException(status\_code=404, detail="Cart not found")

return carts\_db[student\_id]

@app.put("/cart/{student\_id}/update\_item/")

def update\_item\_in\_cart(student\_id: int, item: CartItem):

if student\_id not in carts\_db:

raise HTTPException(status\_code=404, detail="Cart not found")

cart = carts\_db[student\_id]

for cart\_item in cart.items:

if cart\_item.item\_id == item.item\_id:

cart\_item.quantity = item.quantity

return {"message": "Item updated successfully"}

raise HTTPException(status\_code=404, detail="Item not found")

@app.delete("/cart/{student\_id}/remove\_item/{item\_id}/")

def remove\_item\_from\_cart(student\_id: int, item\_id: int):

if student\_id not in carts\_db:

raise HTTPException(status\_code=404, detail="Cart not found")

cart = carts\_db[student\_id]

item\_to\_remove = None

for cart\_item in cart.items:

if cart\_item.item\_id == item\_id:

item\_to\_remove = cart\_item

break

if item\_to\_remove:

cart.items.remove(item\_to\_remove)

return {"message": "Item removed from cart"}

raise HTTPException(status\_code=404, detail="Item not found")

@app.delete("/cart/{student\_id}/clear/")

def clear\_cart(student\_id: int):

if student\_id not in carts\_db:

raise HTTPException(status\_code=404, detail="Cart not found")

carts\_db[student\_id].items.clear()

return {"message": "Cart cleared successfully"}

**Explanation**

1. **Data Models:**
   * CartItem: Defines an item in the shopping cart, with attributes like item\_id, name, quantity, and price.
   * ShoppingCart: A shopping cart associated with a student\_id containing a list of CartItem objects.
2. **In-Memory Storage:**
   * We use carts\_db as a simple in-memory dictionary to store carts where the key is the student\_id and the value is the shopping cart object.
3. **CRUD Endpoints:**
   * POST /cart/{student\_id}/add\_item/: Adds an item to a student's shopping cart. If the item already exists, it updates the quantity.
   * GET /cart/{student\_id}/: Retrieves the student's shopping cart.
   * PUT /cart/{student\_id}/update\_item/: Updates the quantity of an item in the cart.
   * DELETE /cart/{student\_id}/remove\_item/{item\_id}/: Removes an item from the cart.
   * DELETE /cart/{student\_id}/clear/: Clears all items from the cart.

**Step 3: Run the application**

To run the application, use the command:

uvicorn main:app --reload

Replace main with the name of your Python file if it's different.

**Step 4: Example Requests**

1. **Add Item to Cart (POST request)**:
   * URL: POST /cart/1/add\_item/
   * Payload:

json

Copy

{

"item\_id": 101,

"name": "Laptop",

"quantity": 1,

"price": 1200.50

}

1. **Get Cart (GET request)**:
   * URL: GET /cart/1/
   * This will return the cart for student\_id=1.
2. **Update Item in Cart (PUT request)**:
   * URL: PUT /cart/1/update\_item/
   * Payload:

json

Copy

{

"item\_id": 101,

"name": "Laptop",

"quantity": 2,

"price": 1200.50

}

1. **Remove Item from Cart (DELETE request)**:
   * URL: DELETE /cart/1/remove\_item/101/
   * This will remove the item with item\_id=101 from the cart.
2. **Clear Cart (DELETE request)**:
   * URL: DELETE /cart/1/clear/
   * This will clear all items in the student's cart.