Practical Task - SAQA 14909

# OOP vs Procedural Programming - Project-Based Answers

## SECTION 1: Object-Oriented Programming – Using the Driver Subsystem

### 1. Basic principles of a class

In the Mhlanga-eats system the focus was both procedural and object oriented with, the class `Driver` in `models/Driver.js` explaining OOP as it defines a blueprint for all driver entities. It encapsulates properties like `email`, `phoneNumber`, and `vehicleModel`. Classes help organize code and define how driver data is structured and saved.

class Driver {  
 constructor(data) {  
 this.email = data.email;  
 // ...  
 }  
}

### 2. Basic principles of an object

An object is an instance of a class. When a driver submits the registration form, we create a `Driver` object with their information:

const driver = new Driver({ ...formData });

### 3. Basic principles of information hiding and encapsulation

Encapsulation is shown by grouping all driver attributes inside the `Driver` class and exposing only the `save()` method to interact with the database. Internal logic and structure remain hidden from the rest of the application, promoting secure and modular code.

### 4. Basic principles of inheritance

Polymorphism between `User` and `Driver` where `Driver` shares some of the properties with other `Users` in our system:

class User {  
 constructor(email, username) {  
 this.email = email;  
 this.username = username;  
 }  
}  
  
class Driver extends User {  
 constructor(data) {  
 super(data.email, data.username);  
 this.vehicleModel = data.vehicleModel;  
 // ...  
 }  
}

### 5. Principles of polymorphism

Polymorphism allows us to treat different classes (`Driver`, `Admin`, `Customer`) as a common type, like `User`. For example, if both `Driver` and `Admin` extend `User`, we could store them in a common array and call a shared method like `getProfile()` without knowing the exact class.

## SECTION 2: Procedural Programming – Using the Delivery System

### 1. Functions and variables in structured programming

In the earlier stages of your Delivery system, you used procedural patterns — plain functions and global/local variables.

Example:

let deliveries = [];  
  
function createDelivery(orderId, address) {  
 const delivery = { id: orderId, address: address, status: 'pending' };  
 deliveries.push(delivery);  
 return delivery;  
}

- Parameters: `orderId`, `address` are passed to the function.

- Local variables: `delivery` is local to the function.

- Global variable: `deliveries` holds all deliveries and is accessed throughout the file.

### 2. Difference between OOP and Procedural Programming

|  |  |  |
| --- | --- | --- |
| Feature | Object-Oriented (Driver Subsystem) | Procedural (Delivery System) |
| Structure | Organized using classes and objects | Organized using functions and procedures |
| Data & Behavior | Data and methods bundled in classes | Data is separate from functions |
| Reusability | Supports inheritance and polymorphism | Requires function duplication or globals |
| Modularity | High – through encapsulation and abstraction | Lower – often harder to manage complexity |
| Real-world Modeling | Easy to represent real entities (e.g., Driver) | More abstract, better for small utilities |