SIMULATION OF A SPACE MODULE IN THE ARTEMIS MISSION CONTEXT

Tutorial goal:

The **Artemis mission**, led by NASA, represents a new era of space exploration with the goal of returning astronauts to the Moon and establishing a sustainable human presence beyond Earth's orbit. This mission aims to lay the groundwork for future journeys to Mars, focusing on technological innovation and international collaboration.

In this **fun and educational project**, you will be immersed in the Artemis mission context, where you will design and simulate the behavior of a **space module --** either a **habitat** module intended to house astronauts or a **cargo** module transporting vital equipment. The simulations must account for several key factors, including **resource constraints** such as fuel and payload capacity, as well as **physical dynamics** like lunar gravity and orbital mechanics.

Acting as mission engineers, you will also need to optimize the module's resources, considering **industrial management** and **space logistics**, including trajectory planning and resource allocation. This project will integrate and apply concepts from **physics**, **mechanics**, **mathematics**, and **industrial engineering**, while developing skills in **programming**, **modeling**, and **solving complex problems**.

So let's start !

Part 1: Basic Concepts - Structures, Classes, Inheritance, Polymorphism

ⓘ Context:

In this section, you will learn how to model a space module for the Artemis mission using object-oriented programming (OOP) concepts. You will create classes, specialize them through inheritance, and use polymorphism techniques. These foundational skills will enable you to build more complex simulations in later sections.

Step 1: Creating Basic Classes

Learning Objective:

- Understand the concept of **class** and **object**.
- Learn how to define a class in Python with attributes and methods.
- Be able to initialize an object from a class.

1.1 Creating the ModuleSpatial Class

The first step is to create a base class that models a generic space module. You should understand that this class serves as a **template** for creating concrete objects, with properties (attributes) and behaviors (methods).

- **Attributes**: Define the properties of a space module.
    - name: The name of the space module (e.g., "Orion").
    - weight: The total weight of the module (in kilograms).
    - fuel: The amount of fuel available (in liters).
    - speed: The maximum speed of the module (in km/h).
- **Methods**: Define the behaviors of a space module.
    - launch(): Simulates the launch of the module, reducing fuel and displaying a launch message.
    - land(): Simulates a landing, consuming fuel.
    - status(): Displays the current state of the module (remaining fuel, etc.).

1.2 Using the ModuleSpatial Class

You will create instances of ModuleSpatial, simulate launches and landings, and check the module's status.

Exercise:

1. Create an instance of ModuleSpatial for a module named "Orion" with a weight of 25,000 kg, 500 liters of fuel, and a maximum speed of 28,000 km/h.
2. Simulate the launch and landing of the module, and display its status.

Step 2: Inheritance and Specialization of Modules

Learning Objective:

- Understand the concept of **inheritance**: how to create subclasses that share attributes and methods of the parent class while having their own specific characteristics.
- Learn how to **specialize** a class by adding custom behaviors.

2.1 Specialized Subclasses: ModuleCargo and ModuleHabitat

To enhance the simulation, you will create **subclasses** of ModuleSpatial that model specific types of modules, such as cargo and habitat modules.

- **ModuleCargo**: Transports materials.
    - **Additional Attribute**: payload_capacity (the load capacity in kilograms).
    - **Specific Method**: load_material(weight) to load materials into the cargo module.
- **ModuleHabitat**: Transports astronauts.

- **Additional Attribute**: num_astronauts (the number of astronauts on board).
- **Specific Method**: board_astronauts(count) to board astronauts into the habitat module.

2.2 Using the Subclasses

You will create objects from ModuleCargo and ModuleHabitat, then test their functionalities.

Exercise:

1. Create a cargo module cargo_alpha with an initial payload capacity of 1,000 kg and a habitat module habitat_beta with 4 astronauts.
2. Simulate loading and boarding operations for each module.

Step 3: Polymorphism and Interfaces

Learning Objective:

- Introduce the concept of **polymorphism**, allowing methods to be called generically on objects of different types.
- Use **interfaces** to enforce certain methods in subclasses.

3.1 Creating the InterfaceModuleSpatial Interface

An **interface** is an abstract class containing methods without implementation. It requires all implementing classes to define certain methods. In this case, modules must be able to **deploy solar panels** and **separate modules** during missions.

3.2 Implementing the Interface in Subclasses

The ModuleCargo and ModuleHabitat subclasses must implement the methods defined in the interface.

3.3 Polymorphism: Generic Method Usage

Polymorphism allows you to handle objects of different types in a uniform way. For example, you can manage different types of space modules (cargo, habitat) within the same function.

Exercise:

1. Write a function that simulates a space mission by calling the launch(), deploy_solar_panels(), and separate_modules() methods on any module.
2. Use this function to simulate missions for both the cargo and habitat modules.

ⓘ Conclusion:

By the end of this section, you will have acquired:

- The ability to create and manipulate **classes and objects** in Python.
- A solid understanding of **inheritance** to specialize objects.
- Skills in **polymorphism** to treat objects generically.
- Knowledge of using **interfaces** to enforce method implementation in subclasses.

These foundational skills are essential for tackling the more advanced parts of the project, which will include algorithms, computational mechanics, and other complex concepts.