Ecosystem Enviroment : Object Oriented Programming Example in Python

Leonidas Ntrekos s6210058
Email: s6210058@aueb.gr
GitHub: LNtrekos

December 2025

Abstract

The scope of this project is to implement object oriented programming to create an ecosystem (main object) consisting of various species (second object). In addition, a "menu" program is developed that enables the user to interact in many different ways with the ecosystem.

1 Overview

The whole project consists of 3 different scripts (which can be found in the **ecosystem** file) and it's main which is the final implementation.

1. **utils.py** consists of 3 helper functions that take user input and have proper error handling for each case, eg. numeric, float, and character input.
2. **classes.py** consists of the two classes used for the project, the ecosystem and the species class, designed to communicate with each other.
3. **menu.py** includes all the functions that allow the user to interact with the ecosystem, the backbone of the entire program.

For convenience of the user, an executable has been added (which is essential, main.py)

2 utils

Throughout this project, input from the user is often required. These helper functions ensure a smooth workflow with consistent error handling, value checks, and confirmation prompts.

- **user_input_int**
- **user_input_int**

The functions above handle integer/float input from the user with type validation, lower/upper bound checks, and custom prompts for incorrect entries. They share common parameters for the desired prompt at each point in time, which include the main message displayed to the user and the message shown when the value is out of range or of an incorrect type. These functions return validated input from the user.

- **user_input_character**

This function ensures non-empty and user-approved text input. It's only parameter is the main message and returns a validated, confirmed string.

3 classes

The orientation of this project as they are the objects:

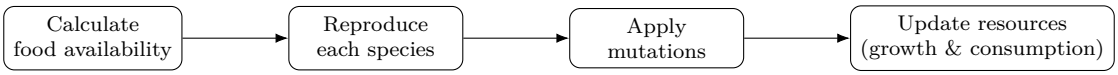
- **Species Class**

Represents an individual Genus (singular species) of the ecosystem. It's parameters are **name**, (initial) **population**, (initial) **growth rate** and (fixed) **mutation rate**. Growth rate represents the "multiplier" for each next generation based on the current population. Each next generation will be the integer of the multiplication of growth rate and current population. Mutation rate is a probability, eg. 0.2, that the specie's growth rate changes to either 0.7 or 1.1 and that procedure is called **mutate** inside the class. Additionally, the class is equipped with the function **reproduce** which depends on the food availability of the ecosystem. If ecosystem's resources are less **in half** (total number) from the ecosystem's total population then we are in short of food (food_availability == 0) and so each species' growth rate is reduced to half, otherwise species' growth rate stays as is. Lastly, inside the class the function **display_info** is designed to (what a surprise) display information for each genus, in the style of:

```
Species Name: Lion
Population: 30
Growth rate: 1.2
Mutation rate: 0.3
```

- **Ecosystem class** As the name already suggests, this class represents the ecosystem environment that contains multiple species. Its parameters are the (initial) **resources**, the (fixed) **growth rate**, and an (initially empty) **species list**, which will be populated with instances of the class `species`. It provides methods to **add**, **search**, **remove**, and **display (all) species**. The ecosystem's resources and growth rate can be updated or modified through corresponding methods defined within the class. Lastly, the ecosystem includes a method that simulates the next generation of all species, but only proceeds if sufficient resources are available for each of them.

  1. **Calculate food availability** based on current resource levels.
  2. **Reproduce each species** according to their reproductive strategy.
  3. **Apply mutations** with probability determined by each species' mutation rate.
  4. **Update resources** after population growth and consumption.



It is evident that this does not fully represent how reproduction actually occurs in an ecosystem, but it serves as an appropriate simplified example that is sufficient for the purposes of this project.

4 menu

This file contains the functions that connect user input with the Ecosystem and Species classes. It provides main menu interface, input-driven creation and modification of ecosystems and species Simulation controls (automated and interactive) and utility functions for listing and

displaying ecosystem status. The main idea behind the menu system is that it gives the user a simple, controlled interface to all functionalities of the ecosystem. It acts as the connecting layer between the Ecosystem and Species classes and the utility functions described earlier, effectively providing a complete and user-friendly program. More detailed, the menu allows the user to:

1. **Initialize and inspect an ecosystem**

Create a new ecosystem from scratch using **create_new_ecosystem** and Display the current state of the ecosystem using **print_ecosystem_info**.

- **Manage species interactively**

Add new species, search for existing species, update their attributes, remove species, or list all species in the ecosystem using the corresponding menu options.

- **Run simulations with or without user interaction**

Simulate resource dynamics and species evolution either automatically across generations, or in step-by-step mode where the user must confirm each transition before continuing.