Object-Oriented Python

Your Trusty Instructor

May 5, 2025

Introduction to Object-Oriented Programming (OOP)

Object-Oriented Programming is a paradigm focused on modeling software as a collection of objects. Objects can represent real-world entities, concepts, or even abstract ideas.

Each object has:

- **State**: The data or attributes the object holds (e.g., name, age).
- **Behavior**: The actions or methods it can perform (e.g., drive, jump).

1. Key Concepts

1.1. Classes and Objects

- A **class** is like an architect's blueprint.
- An **object** is an actual house built using that blueprint.

Example naming convention: `PascalCase` (e.g., `MyAwesomeClass`). Always start each word with a capital letter.

1.2. The `self` Keyword and `__init__` Method

The `self` keyword refers to the instance itself.

The `__init__` method initializes the attributes of the object.

```python
class Car:
    def __init__(self, brand, model, max_speed=0):
        self.brand = brand
        self.model = model
        self.max_speed = max_speed
```

1.3. Attributes and Default Values

You can set default values in `__init__` parameters.

Example:

```
class Car:
    def __init__(self, speed=0):
        self.speed = speed
```

2. The Data Model

Special attributes (called dunder methods) allow interaction with Python internals.
Examples:

- `__str__`: Defines string representation (for `print()`)
- `__repr__`: Defines unambiguous string representation
- `__eq__`: Defines equality comparison

2.1. Example

```
class Car:
    def __init__(self, brand, model, max_speed, min_speed):
        self.brand = brand
        self.model = model
        self.max_speed = max_speed
        self.min_speed = min_speed

    def __str__(self):
        return f"{self.brand}␣{self.model}␣{self.max_speed}␣{self.min_speed

    def __eq__(self, other):
        return self.brand == other.brand and self.model == other.model
```

3. Inheritance

Inheritance lets you define a new class based on an existing one.

```
class Vehicle:
    def __init__(self, type_of_vehicle):
        self.type_of_vehicle = type_of_vehicle

class Car(Vehicle):
    def __init__(self, brand, model):
        super().__init__("Car")
        self.brand = brand
        self.model = model
```

Summary

OOP in Python helps you build modular, scalable, and reusable code. Get comfortable with classes, objects, encapsulation, dunder methods, and inheritance to unlock powerful design capabilities.

4. Day 02 — Object-Oriented Exercises

Exercise 00 — Create a Professor Class

Directory: `day02/ex00/`
Files to submit: `professor.py`
Goal: Define your first simple class with attributes.

Task:

- Create a class `Professor`.
- The class must have attributes: `first_name`, `last_name`, and `age`.
- Define an `__init__` method to set these attributes.

Example:

```
prof = Professor("John", "Doe", 45)
print(prof.first_name)   # John
print(prof.last_name)    # Doe
print(prof.age)          # 45
```

Extra questions:

- What happens if you forget to use `self` in your class methods?
- How can you add default values to some attributes?

Exercise 01 — Car Class with Acceleration and Braking

Directory: `day02/ex01/`
Files to submit: `car.py`
Goal: Build a class with attributes and methods to modify its internal state.

Task:

- Create a class `Car`.
- Attributes: `brand`, `model`, `max_speed`, `min_speed`, `speed` (default 0).
- Methods:
  - `accelerate`: Increases the speed by 50, without exceeding `max_speed`.
  - `brake`: Decreases the speed by 50, without going below `min_speed`.

Example:

```
car = Car("Peugeot", "308", 180, -30)
print(car.speed)  # 0
car.accelerate()
print(car.speed)  # 50
car.accelerate()
print(car.speed)  # 100
car.brake()
print(car.speed)  # 50
```

Extra questions:

- Why is it important to add limits on acceleration and braking?
- How could you make the increase or decrease amounts adjustable?

Exercise 02 — The Book

Directory: `day02/ex02/`
Files to submit: `book.py`, `recipe.py`, `test.py`
Goal: Get familiar with creating multiple classes and interacting with their objects.

Task:

- Implement two classes: `Book` (in `book.py`) and `Recipe` (in `recipe.py`).
- Create a `test.py` file to test your classes.

Recipe Class Attributes:

- `name` (str): Name of the recipe.
- `cooking_lvl` (int): Cooking level (1 to 5).
- `cooking_time` (int): In minutes (must not be negative).
- `ingredients` (list): List of strings.
- `description` (str): (Optional) Description of the recipe.
- `recipe_type` (str): Either `"starter"`, `"lunch"`, or `"dessert"`.

Recipe Class Requirements:

- Validate input on creation.
- Implement the `__str__` method to format the recipe info.

Book Class Attributes:

- `name` (str): Name of the book.
- `last_update` (datetime): Date of the last update.
- `creation_date` (datetime): Creation date.
- `recipes_list` (dict): Dictionary with keys `"starter"`, `"lunch"`, and `"dessert"`.

Book Class Methods:

- `get_recipe_by_name(name)`: Prints and returns a recipe matching the name.
- `get_recipes_by_types(recipe_type)`: Returns all recipe names of the given type.
- `add_recipe(recipe)`: Adds a recipe and updates `last_update`.

Testing Instructions:

- Create your test file `test.py`.
- Import the classes using:

```python
from book import Book
from recipe import Recipe
```

- Test your classes and methods thoroughly.

Important: Validate inputs carefully and exit gracefully on errors. The description in `Recipe` can be empty, but all other fields must be correctly validated.

Exercise 03 — Family Tree

Directory: `day02/ex03/`
Files to submit: `game.py`
Goal: Practice inheritance between classes.

Task:

- Create a class `GotCharacter` with attributes:
  - `first_name`
  - `is_alive` (default `True`)
- Create a child class for a GoT house (example: `Stark`) that inherits from `GotCharacter`.
- Add attributes to the child class:
  - `family_name` (default value matching the class name)
  - `house_words` (e.g., `"Winter is Coming"`)
- Implement two methods:
  - `print_house_words`: Prints the House words.
  - `die`: Sets `is_alive` to `False`.

Example:

```
from game import Stark
arya = Stark("Arya")
print(arya.__dict__)
# {'first_name': 'Arya', 'is_alive': True, 'family_name': 'Stark', 'house_w
arya.print_house_words()
# Winter is Coming
print(arya.is_alive)
# True
arya.die()
print(arya.is_alive)
# False
```

Additional Instructions:

- Provide a docstring for the child class (example: "A class representing the Stark family. Or when bad things happen to good people.").
- Feel free to create other child classes from `GotCharacter`.

Exercise 04 — Watch out for the sale people!

Directory: `day01/ex10/`
Files to submit: `sales_energy.py`
Goal: Model and simulate a chain reaction of excited salespeople in the open office ... you know how it can be. Ding ding ding goes the bell.

It is your presales day and all the sales are therein their 10x10 open space. Each salesperson has an "energy level" — when they get too excited (energy > 9), they scream and energize their neighbors (including diagonals). This continues until no one new screams.

At each step:

- Each salesperson gains 1 energy.
- Any person whose energy goes above 9 "screams".
- Screaming gives +1 energy to each adjacent neighbor (including diagonals).
- A salesperson can only scream once per step.
- Once the step ends, all screamers reset their energy to 0.

Your program should:

- Simulate one or more steps of this behavior.
- Return the number of "screams" at each step.
- Detect if there's a moment when all salespeople scream at once.
- All of this using objects where you see it convenient

Input: A 10x10 grid of digits (as strings), where each digit is the initial energy level of one salesperson.

Input:

```
5483143223
2745854711
5264556173
6141336146
6357385478
4167524645
2176841721
6882881134
4846848554
5283751526
```

To make it more understandable:

Before any steps:

```
11111
19991
19191
19991
11111
```

After step 1:

```
34543
40004
50005
40004
34543
```

After step 2:

```
45654
51115
61116
51115
45654
```

Given the starting energy levels of the salepeoples in your open office, simulate 100 steps. How many total screams are there after 100 steps?