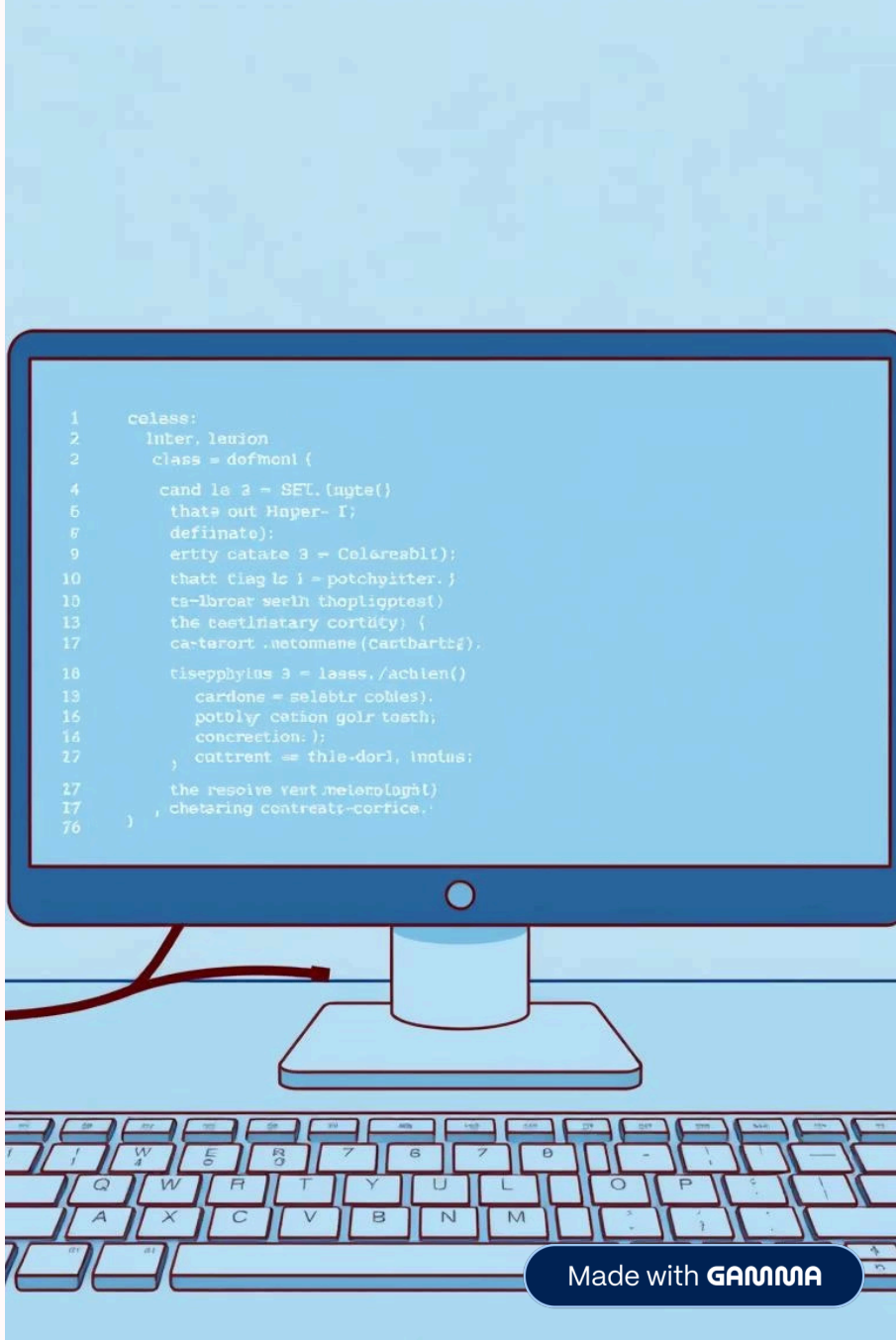
Classes & Objects vs. Functions



Agenda of Today's Class

- Classes and Objects
- Understanding _init_ and self in classes
- Class and Object Attributes
- Class and functions difference
- **LeetCode 1480 running sum**

What is a Function in Python?

A function is a reusable block of code designed to perform a specific task when called.

- Defined with the `def` keyword
- Can accept inputs (parameters/arguments)
- Can return outputs back to the caller
- Functions focus on **actions**: "Do this!"

```
def calculate_area(length, width):
 """Calculate the area of a rectangle"""
 area = length * width
 return area

# Function call
room_area = calculate_area(12, 15)
print(f"Room area: {room_area} sq ft")
```

What is a Class in Python?

Blueprint for Objects

A blueprint from which objects are created. It bundles related data and functions together in a logical unit.

Attributes & Methods

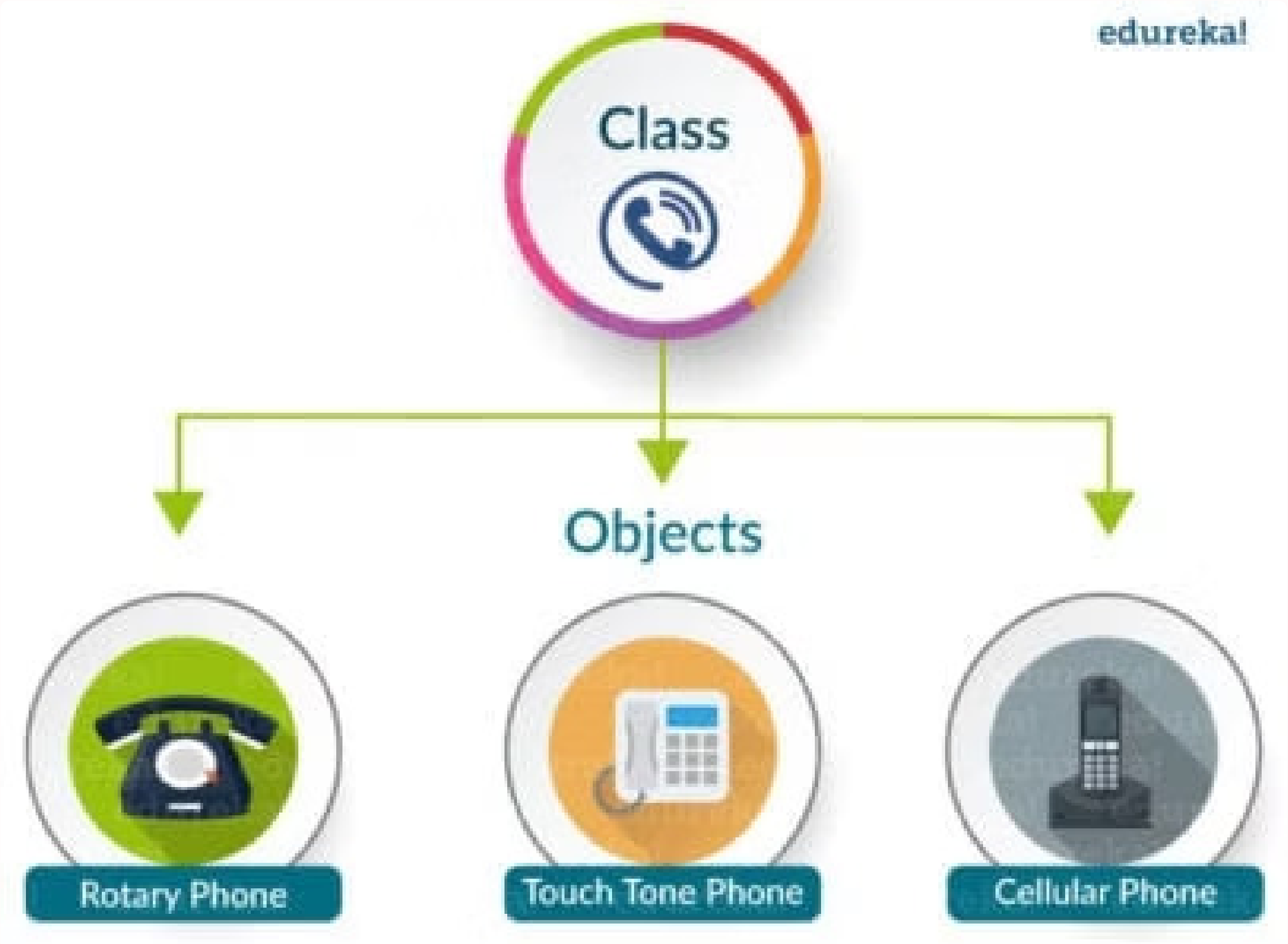
Classes contain attributes (data) and methods (functions). The `__init__` method initializes new object instances with specific attributes.

Real-world Modeling

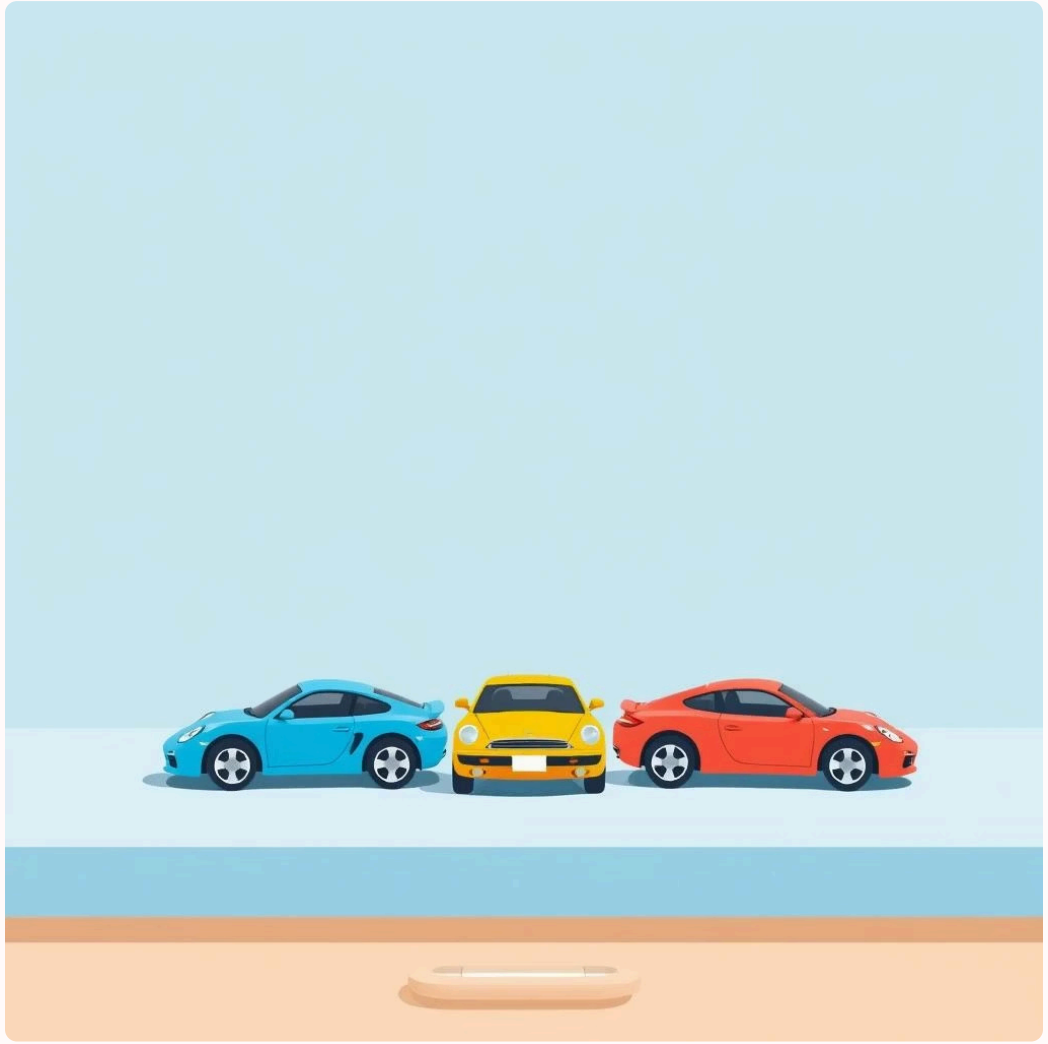
Classes enable developers to model real-world entities by combining state (data) and behavior (methods) in a single structure.

```
class Car:
 def __init__(self, color, model):
 self.color = color # attribute
 self.model = model # attribute
 self.speed = 0 # attribute with default value

 def accelerate(self): # method
 self.speed += 5
```



Objects: Instances of Classes



Objects are concrete instances created from a class blueprint:

- Each object **maintains its own state** (attribute values)
- **Multiple objects** can be created from a single class
- Objects interact through their methods
- Each instance is **independent** of others

```
# Creating multiple Car objects
tesla = Car("red", "Model 3")
toyota = Car("blue", "Corolla")

# Each has its own state
tesla.accelerate() # tesla.speed is now 5
print(toyota.speed) # still 0
```

Functions vs. Classes: Key Differences

State Management

Functions: Stateless - don't remember anything between calls

Classes: Stateful - objects maintain data between method calls

Organization

Functions: Individual units performing specific tasks

Classes: Group related data and multiple functions together

Rule of thumb: **Classes → PascalCase**, **functions/variables → snake_case**.

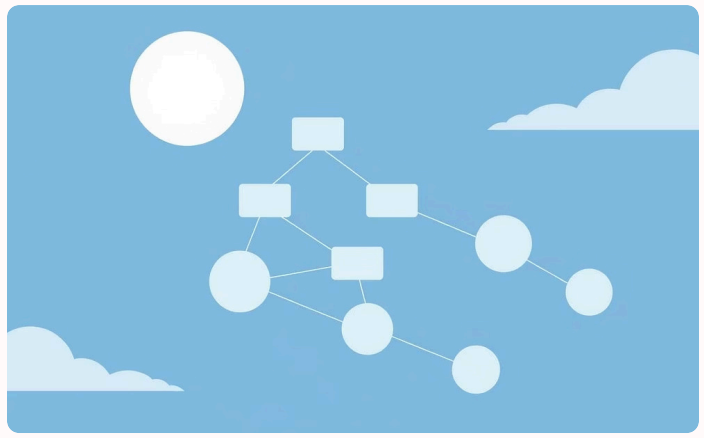
When to Use Functions?

Functions are ideal for:

- Simple operations that transform inputs to outputs
- Utility operations used across different parts of code
- **Pure computational tasks** (math operations, text formatting)

Functions follow the **Single Responsibility Principle** - each function should do one thing well.

When to Use Classes & Objects?



Entity Modeling

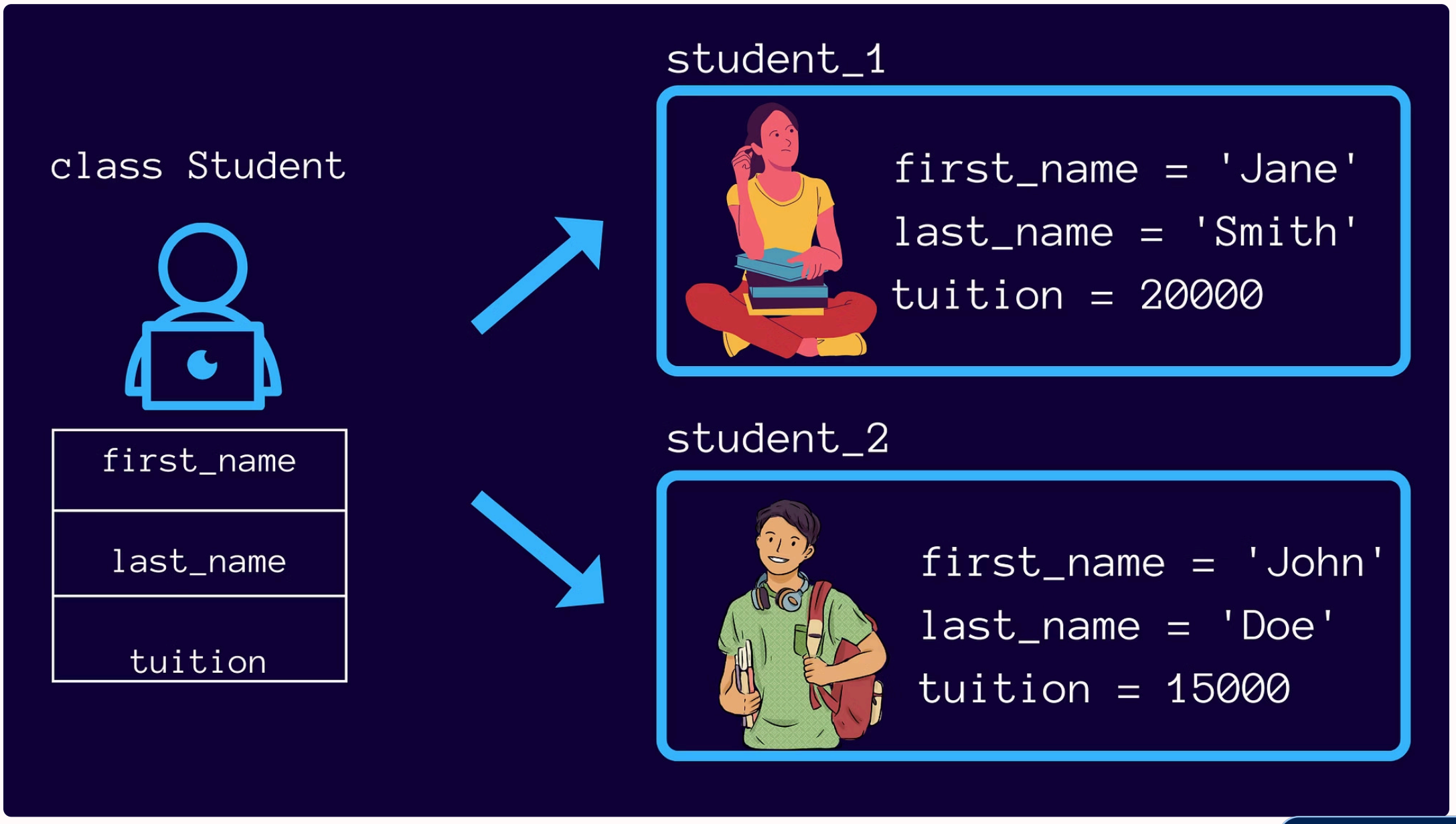
When representing real-world entities with both attributes and behaviors (bank accounts, users, products).

Complex Systems

When building larger applications where organization and structure are crucial.

Code Reuse

When leveraging inheritance or composition to reuse code across similar objects.



Class and Object Attributes

Python distinguishes between attributes that belong to the class itself and those that belong to individual instances of the class. This distinction is crucial for managing shared data and unique object states.

Class Attributes

Defined directly inside the class but outside any methods. These attributes are shared by all instances of the class. They are useful for constants, default values, or properties that apply universally to all objects of that type.

```
class Planet:
GALAXY = "Milky Way" # Class attribute

def __init__(self, name):
self.name = name # Instance attribute


# Accessing a class attribute
print(Planet.GALAXY)
```

Object (Instance) Attributes

These are unique to each object (instance) of a class. They are typically defined within the `__init__` method using the `self` keyword. Instance attributes store data that is specific to an individual object's state.

```
earth = Planet("Earth")
mars = Planet("Mars")

# Each object has its own 'name'
print(earth.name) # "Earth"
print(mars.name) # "Mars"
```

Practical Example: Age Calculator

Function Approach

```
def calculate_age(birth_year, current_year):
    return current_year - birth_year

# Usage
age = calculate_age(1990, 2024)
print(f"You are {age} years old")
```

Simple, focused on calculation only. No memory of past calculations.

Class Approach

```
class Person:
    def __init__(self, name, birth_year):
        self.name = name
        self.birth_year = birth_year
        self.age = None

    def calculate_age(self, current_year):
        self.age = current_year - self.birth_year
        return self.age

    def display_info(self):
        return f"{self.name} is {self.age} years old"

# Usage
john = Person("John", 1990)
john.calculate_age(2024)
print(john.display_info())
```

Maintains person's data and offers multiple related methods.

Choose the approach that best fits your specific problem and code organization needs!