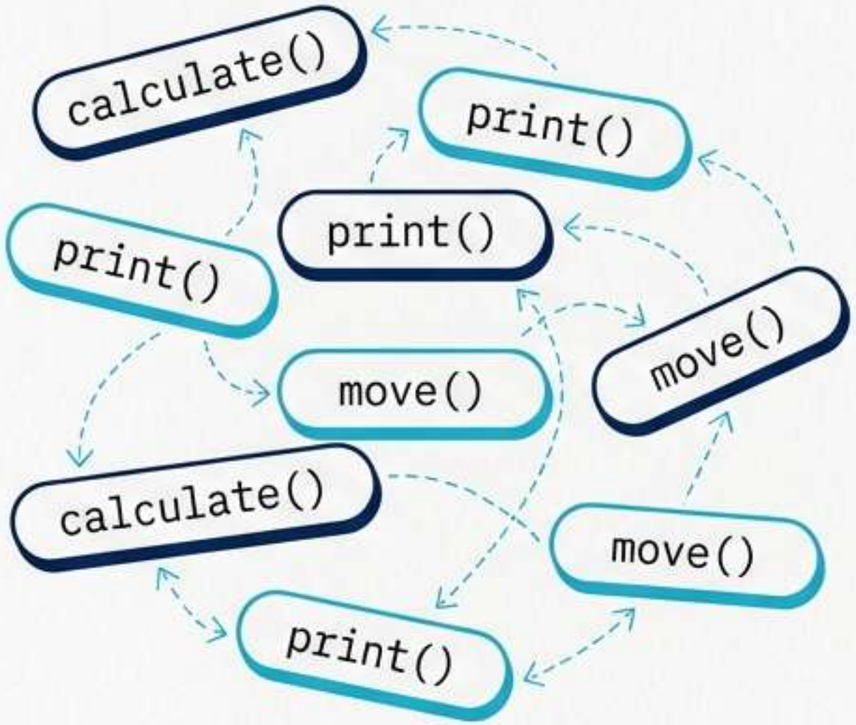
Classes & Objects: The Foundations of Object-Oriented Programming

Unit 11.4A · Python OOP

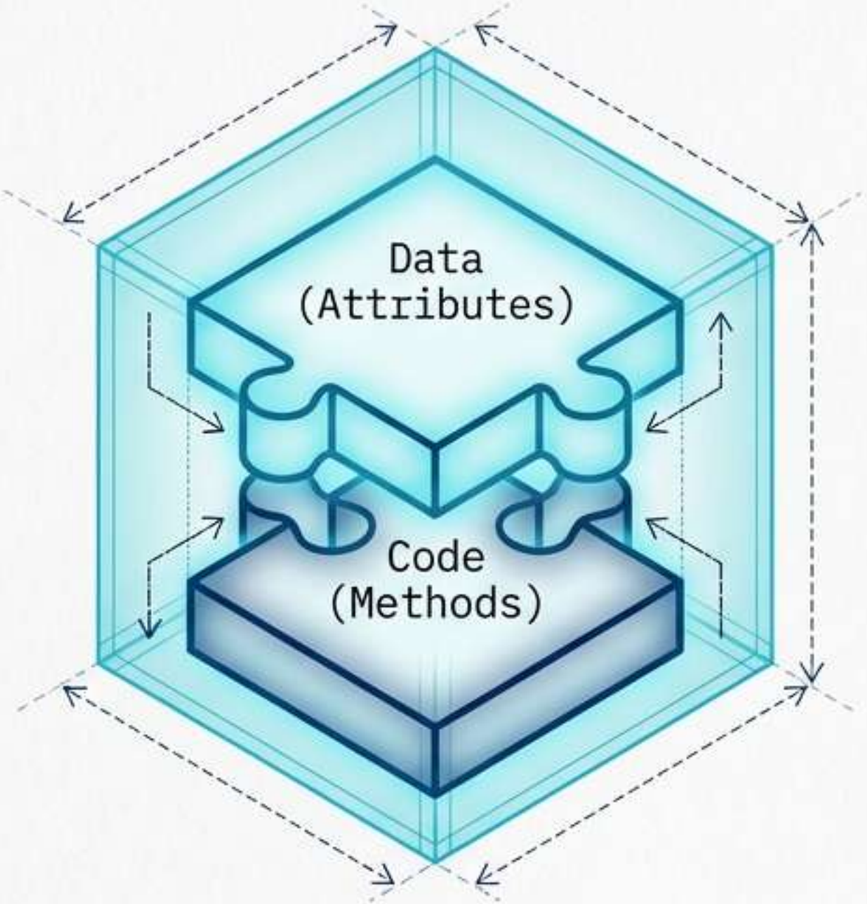
Learning Objectives

- **Topic:** Classes and Objects in Python.
- **Target 11.4.1.1:** Create classes and instances of classes.
- **Target 11.4.1.3:** Use the special method __init__ to set default properties.
- **Target 11.4.1.2:** Develop methods for the class.

Traditional Programming

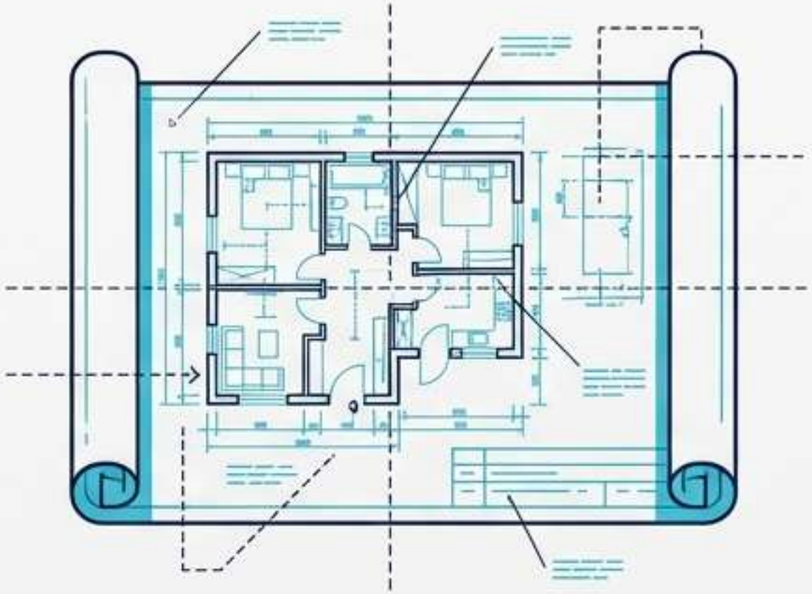


Object-Oriented Programming (OOP)



OOP is a paradigm where code is organized around ‘objects’ rather than actions. An object securely binds both data and behavior into a single entity.

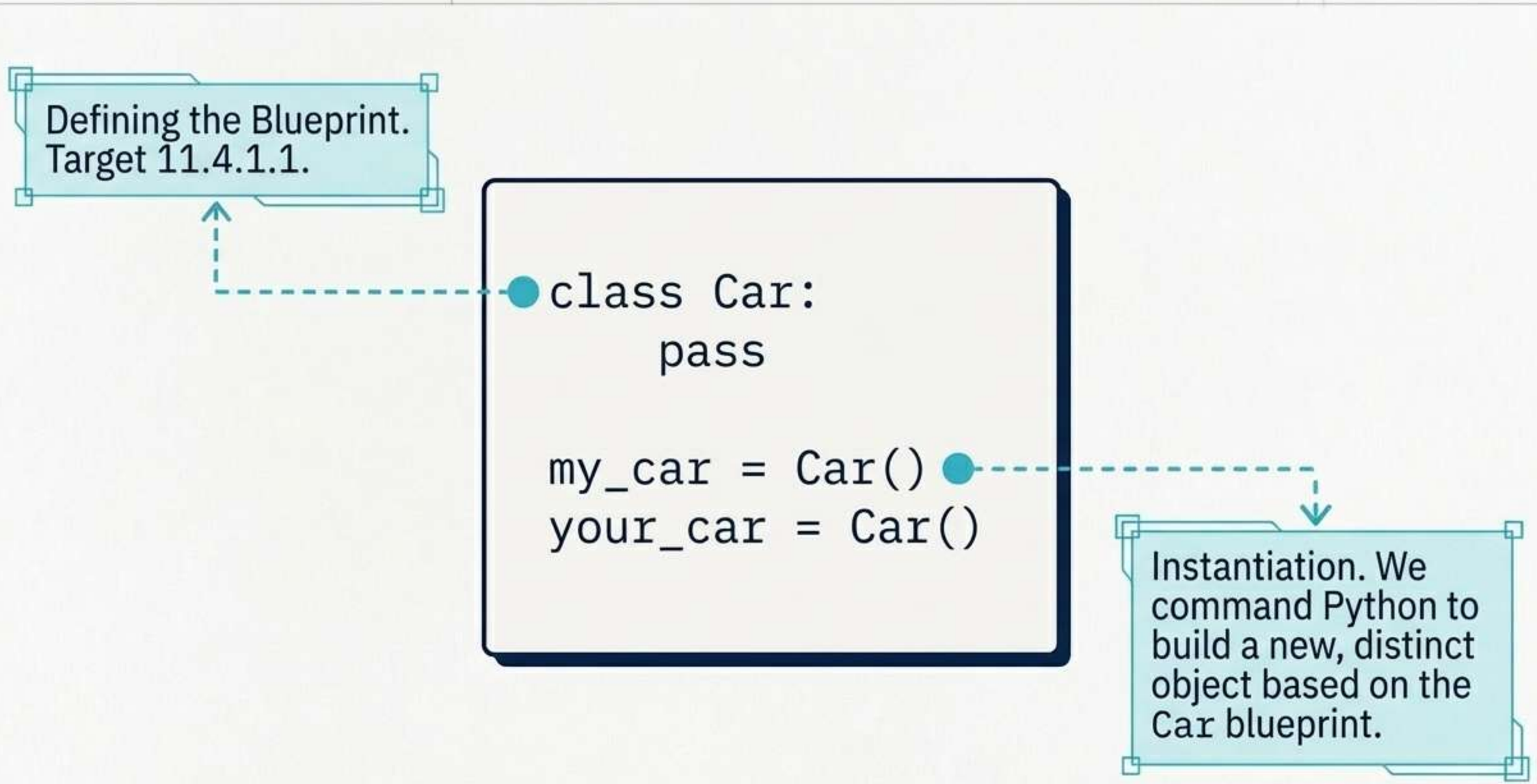
Class (The Blueprint)



- It is the conceptual plan.
- Describes what the entity SHOULD have (walls, windows).
- Describes what it can DO (open doors).
- A single set of structural rules.

Object (The Instance)

- The ACTUAL house built from the blueprint.
- Unlimited instances can be built from one class.
- Each instance holds unique data (different colors).
- Strictly follows the blueprint's structure.

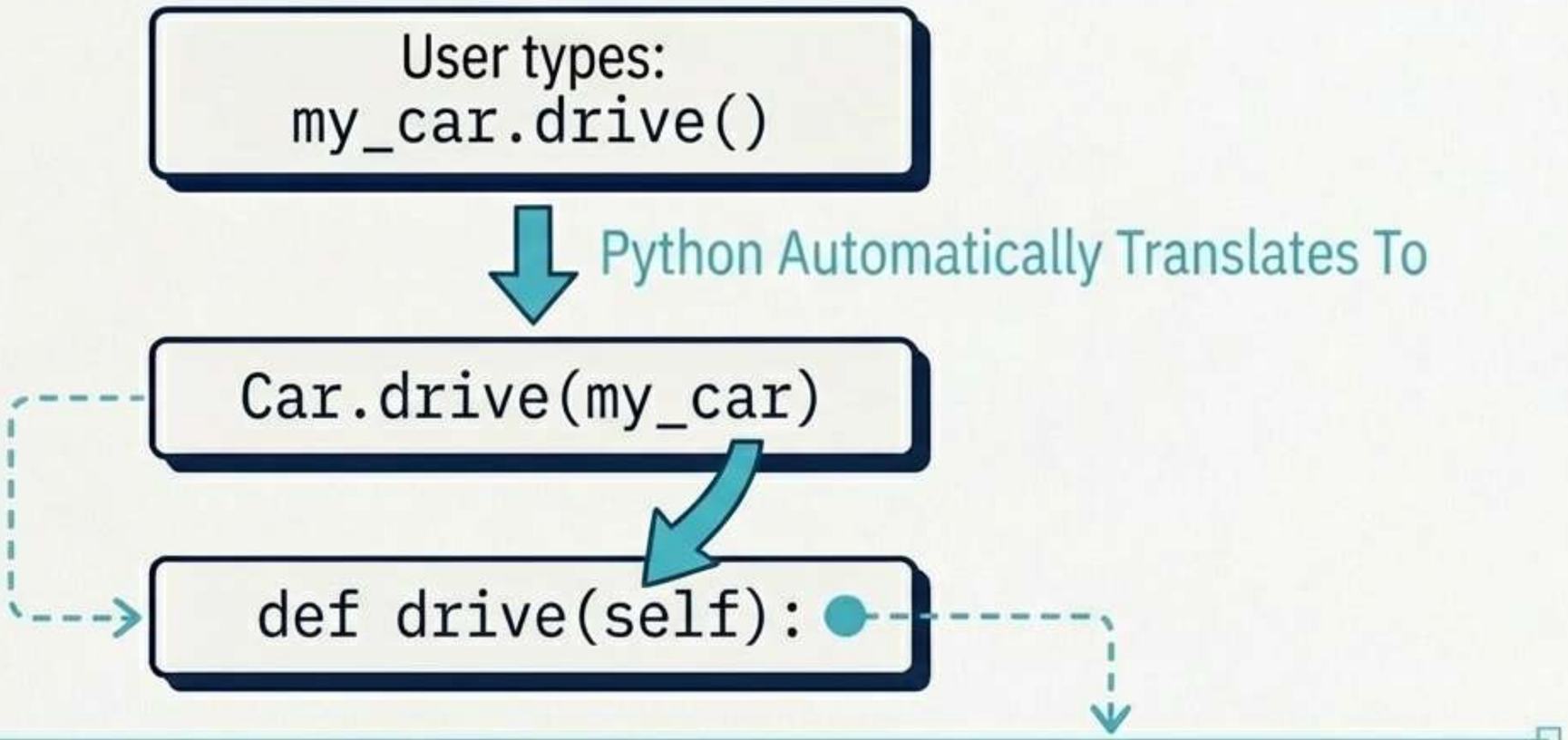

Defining the Blueprint. Target 11.4.1.1.
class Car:
pass
my_car = Car()
your_car = Car()
Instantiation. We command Python to build a new, distinct object based on the Car blueprint.

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

c1 = Car('Toyota', 'Camry')
c2 = Car('BMW', 'X5')
```

The Constructor (Target 11.4.1.3). The __init__ method runs automatically the exact moment a new object is created. It acts as the initialization protocol, setting up the object's default attributes (data).

Demystifying ‘self’



What is self? It is simply a reference to the CURRENT object being used. When you call a method on `my_car`, Python automatically passes `my_car` in as the `self` argument so the blueprint knows exactly which "house" to modify.

Adding Behavior (Methods)

```
# Target 11.4.1.2: Methods
def drive(self):
    self.speed += 10
    print(f'{self.brand} is
driving at {self.speed} km/h')

def stop(self):
    self.speed = 0
```

```
my_car = Car('Tesla', 'Model S')
```

```
my_car.drive()
```

```
my_car.drive()
```

```
my_car.stop()
```

Methods define behavior. They are functions locked inside a class that change the internal state (attributes) of the specific object.

Worked Example 1: The Student

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def promote(self):
        self.grade += 1
        print(f'{self.name} is now in
Grade {self.grade}')

ali = Student('Ali', 10)
ali.promote()
```

Terminal/Console

Console Output

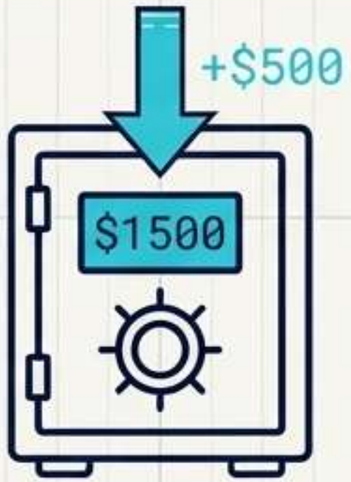
```
> 10
> Ali is now in Grade 11
```

State Change in Action. Calling the promote() method directly increments self.grade without requiring external variables.

Worked Example 2: The Bank Account

```
acct = BankAccount(
    'Dana', 1000)
```


```
acct.deposit(500)
```




```
acct.withdraw(2000)
```

Methods Protect Data. We use if/else logic inside methods to ensure an object's internal state cannot be corrupted by bad inputs.

Worked Example 3: Dynamic Calculations

```
def get_age(self, current_year):
    print(f'Age - {current_year - self.year_of_birth}')
```

User Profile

```
user1 = Person('Azamat',
       'Arystanov', 1984)
```

User Profile

```
user2 = Person('Karina',
       'Mahmudova', 2004)
```

Age - 40

Beyond self. Methods can take external arguments (like current_year) to calculate outputs dynamically based on the object's static attributes.

Pitfalls & Common Errors

Hazard 1: The Missing self

```
def drive():
```

```
def drive(self):
```

Impact: Causes a fatal TypeError when the method is called, because Python tries to pass the object into a function that has no room for it.

Hazard 2: The Underscore Trap

```
def _init__(self):
```

Single underscores

```
def __init__(self):
```

Double underscores / Dunder

Impact: A single underscore creates a normal method that Python will ignore during object creation. Attributes will fail to initialize.

Independent Practice: Formative Assessment

1. Navigate to: app.formative.com/join
2. 2. Enter the Join Code below.

446PDV

Complete the assigned practical tasks to test your understanding of class creation, initialization, and method development.

Graded Tasks Matrix

Phase 3: Apply

Write code: Create a Rectangle class with attributes length and width. Implement methods for area() and perimeter().

Phase 4: Create

Design a system: Build a Book class (title, author, page count). Engineer a read(pages) method that persistently tracks reading progress.

Phase 1: Remember

Define the terms: What is the exact difference between a class and an object? Explain the role of self.

Phase 2: Understand

Explain the mechanics: Why is the __init__ method required, and what happens to an object if we fail to define it?