Methods, __str__ & Class vs Instance Attributes

AiCE Software Engineering Foundations — Python Programming (Advanced)

A class with only data isn't yet an object in the full sense — it needs behavior too. This lesson adds methods, the functions that give an object the ability to act on its own data, controls how an object displays itself with __str__, and distinguishes data that belongs to one object from data shared by every object of a class.

1. Methods — Functions That Belong to a Class

A method is a function defined inside a class. Methods define what objects of that class can do — their behavior. Like __init__, every method's first parameter is self, giving the method access to the object's own attributes.

```
class Student:
    def __init__(self, name, age, country, math, english,
science):
        self.name = name
        self.age = age
        self.country = country
        self.scores = {
            "Math"   : math,
            "English": english,
            "Science": science
        }

    def get_average(self):
        """Calculate and return the student's average
score."""
        scores = list(self.scores.values())
        return round(sum(scores) / len(scores), 2)

    def get_grade(self):
        """Return the letter grade based on the average
score."""
        avg = self.get_average()
        if avg >= 70: return "A"
```

```
        elif avg >= 60: return "B"
        elif avg >= 50: return "C"
        elif avg >= 40: return "D"
        else: return "F"

    def get_status(self):
        """Return PASS or FAIL based on the average score."""
        return "PASS" if self.get_average() >= 50 else "FAIL"

    def display_profile(self):
        """Print a formatted profile for this student."""
        print("=" * 40)
        print(f"  Name    : {self.name}")
        print(f"  Age     : {self.age}")
        print(f"  Country : {self.country}")
        print(f"  Average : {self.get_average()}")
        print(f"  Grade   : {self.get_grade()}")
        print(f"  Status  : {self.get_status()}")
        print("=" * 40)

# Create student objects
amara = Student("Amara Diallo", 22, "Senegal", 88, 76, 91)
kofi = Student("Kofi Mensah", 19, "Ghana", 45, 52, 38)

# Call methods on each object
amara.display_profile()
kofi.display_profile()
```

Each student object calls its own methods using its own data. amara.get_average() calculates from amara's scores; kofi.get_average() calculates from kofi's scores. The method code is written once inside the class and works correctly for every object.

> **Methods Calling Methods**
> *Notice that get_grade() calls self.get_average() — a method calling another method on the same object. This is perfectly normal and is how you build behavior from smaller, focused pieces, exactly like functions calling other functions from an earlier unit.*

2. The __str__ Method — Controlling How Objects Display

When you print an object directly, Python shows something like <__main__.Student object at 0x7f3a...> — not very informative. The __str__ dunder method lets you define exactly how an object should look when printed or converted to a string:

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

    def __str__(self):
        """Return a readable string representation of this
student."""
        return f"Student({self.name}, age {self.age},
{self.country})"

amara = Student("Amara Diallo", 22, "Senegal")
print(amara)         # Calls __str__ automatically
print(str(amara))
```

This produces:

```
Student(Amara Diallo, age 22, Senegal)
Student(Amara Diallo, age 22, Senegal)
```

Any time Python needs a string representation of your object — printing it, including it in an f-string, or passing it to str() — it calls __str__ automatically. Defining __str__ makes your objects readable and your debugging dramatically easier.

3. Class Attributes vs. Instance Attributes

Instance attributes (set in __init__ with self.attribute_name) are unique to each object. Class attributes are defined directly in the class body — outside any method — and are shared by all instances of the class:

```
class Student:
    # Class attributes — shared by ALL Student objects
    institution = "AiCE"
    passing_mark = 50

    def __init__(self, name, score):
        # Instance attributes — unique to each object
        self.name = name
        self.score = score

    def get_status(self):
        # Class attribute accessed via self or Student
        if self.score >= Student.passing_mark:
            return "PASS"
        return "FAIL"

amara = Student("Amara", 88)
kofi = Student("Kofi", 42)

# Class attributes accessible from any instance
print(amara.institution)     # AiCE
print(kofi.institution)      # AiCE
print(Student.institution)   # AiCE — accessed directly from
the class

# Instance attributes are unique
print(amara.name)            # Amara
print(kofi.name)             # Kofi
print(amara.get_status())    # PASS
print(kofi.get_status())     # FAIL
```

Class attributes are useful for values that apply to all objects of a type — a passing mark, a platform name, a version number, a configuration constant. They are defined once and shared, rather than duplicated in every object's __init__.

When to Use Which

Ask: is this value the same for every object of this class, or does it vary from object to object? Same for every object → class attribute. Varies per object → instance attribute, set in __init__.

Key Takeaways

A method is a function defined inside a class, always taking self as its first parameter, and methods routinely call other methods on the same object to build behavior from smaller pieces. __str__ controls what appears when an object is printed, converted to a string, or embedded in an f-string — defining it is standard professional practice. Class attributes, defined directly in the class body, are shared identically by every instance; instance attributes, set inside __init__ with self., are unique to each object. The next lesson brings everything from this unit together into one complete, well-designed Student class — including patterns for converting objects to and from dictionaries, which prepares you directly for JSON storage later in this program.