Welcome to this CoGrammar lecture: Instance, Static and Class Methods

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators answering questions.

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** throughout this session, should you wish to ask any follow-up questions.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: **Feedback on Lectures**
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

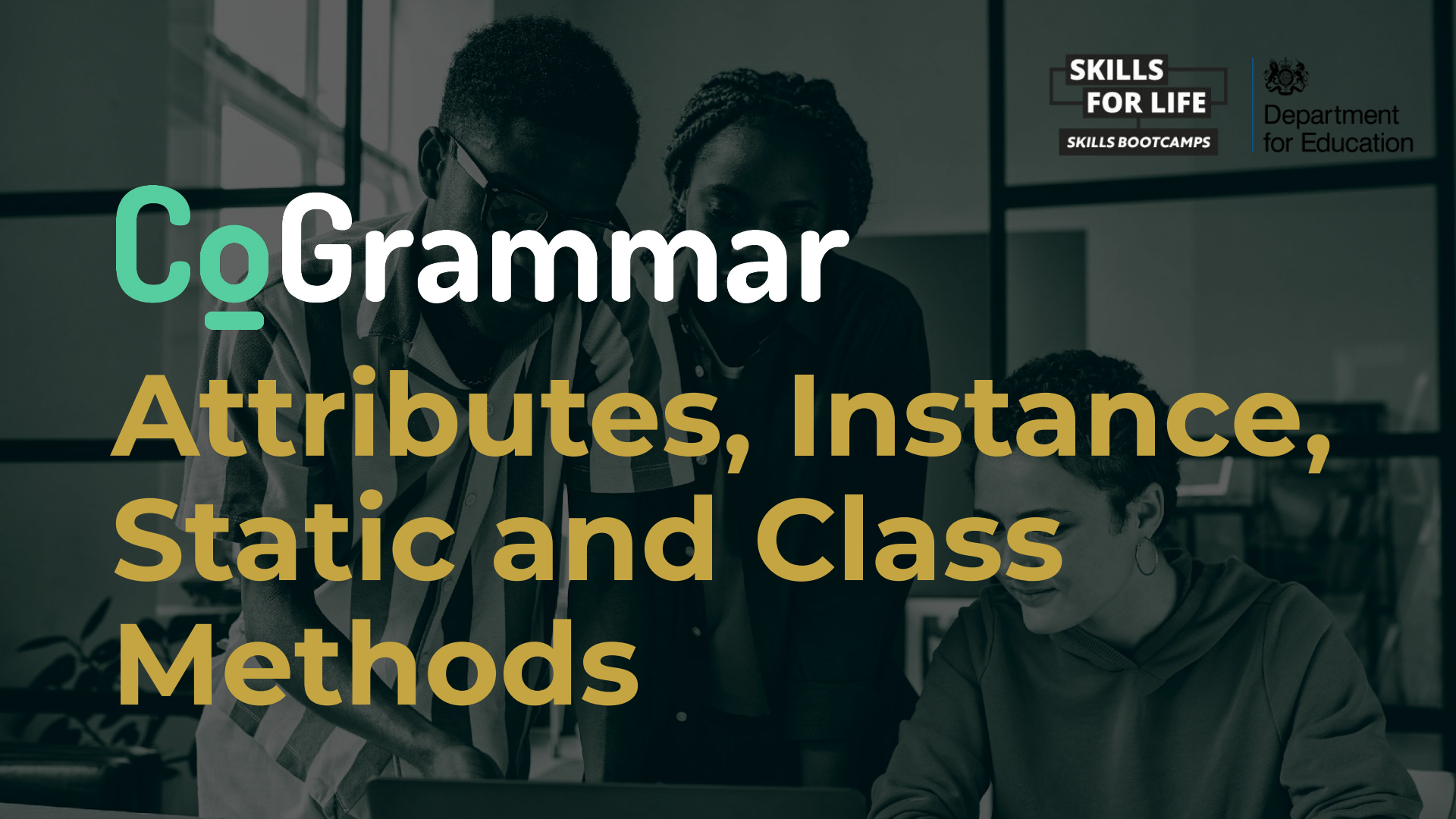
Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev


SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
Attributes, Instance, Static and Class Methods

Poll

```
class Sample:
    def   init  (self):
        self.number = 5

    def update(self, value):
        self.number = value

obj = Sample()
obj.update(10)
print(obj.number)
```

1. What will happen if you run the following code?

1. 5
2. **Error:**update() takes no arguments
3. 10

CoGrammar

Poll

```
class Counter:
    def   init  (self, start=0):
        self.count = start

    def increment(self):
        self.count += 1

obj1 = Counter()
obj2 = Counter(10)

obj1.increment()
obj2.increment()
print(obj1.count, obj2.count)
```

2. Which statement is true about self in this context?

A. self.count is shared between all instances of the class.

B. self.count will be reset to 0 after each increment call.

C. Each instance has its own separate value for self.count.

CoGrammar

Learning Outcomes

- Differentiate between class attributes and instance attributes.
- Explain and use static and class methods in Python.

Instance of a Class

CoGrammar

Class - Instance - Object

CoGrammar

Class - Instance - Object

CoGrammar

Class - Instance - Object

In programming terms:

- **A class** is a blueprint for creating objects. It defines the attributes (characteristics) and methods (actions) that objects of that type will have. It's like a template or a factory.

- **An instance (or object)** is a specific, individual creation of that class. It's a concrete realization of the blueprint. It's like a particular cookie made from the cookie cutter.

Class vs Instance Attributes

CoGrammar

What Are Class Attributes?

Definition: Class attributes are variables defined within a class, but outside of any methods. They are shared by all instances of the class.

Key Points:

- They belong to the class, not the individual instance.
- If modified, the change affects all instances of the class.

Example of Class Attributes

```
class MyClass:
    class_attribute = 0  # Class attribute

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

# Accessing class attribute
obj1 = MyClass(10)
print(MyClass.class_attribute) # Access through the
class
print(obj1.class_attribute)  # Access through an
instance
```

What Are Instance Attributes?

- **Definition**: Instance attributes are variables that are bound to a particular instance of a class.
- **Key Points**:
  - They are unique to each object.
  - Defined inside the `__init__` method.
  - Do not affect other instances of the class.

Example of Instance Attributes

```
class MyClass:
    def __init__(self, value):
        # Instance attribute
        self.instance_attribute = value

# Creating instances
obj1 = MyClass(10)
obj2 = MyClass(20)

print(obj1.instance_attribute)   # 10
print(obj2.instance_attribute)   # 20
```

Class Attributes vs Instance Attributes

Attribute Type	Scope	Access Through	Shared Between Instances?
Class Attribute	The entire class	Class or instance	Yes
Instance Attribute	Specific to each object	Instance only	No

- **Key Difference:** Class attributes are shared across all instances, while instance attributes are unique to each instance.

Instance Methods

CoGrammar

What are Instance Methods?

- **Definition**: Instance methods are functions defined within a class and **bound** to an instance of the class. They operate on data (**attributes**) specific to that instance.
- **Key Points**:
  - They have access to instance attributes via `self`.
  - They can modify instance attributes and return instance-specific information.

Instance Methods - Example

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

    def study(self):
        print(f"{self.name} is studying hard!")

# Creating a student object
student1 = Student("Alice")

# Calling the instance method
print(student1.study()) # Alice is studying hard!
```

Static Methods

CoGrammar

Purpose of Static Methods

- **Definition**: Static methods do not depend on the class or instance data. They are self-contained and perform operations that do not modify the state of the object or class.
- **Key Use Case**: Utility functions that are related to the class, but don't need to access or modify class/instance attributes.

Static Methods - Example

```
class Calculator:

    @staticmethod
    def add(x, y):
        return x + y

    def multiply(self, x):
        self.value *= x

    def __init__(self, initial_value=0):
        self.value = initial_value

print(Calculator.add(5, 3))  # Static method call
calc = Calculator(10)
calc.multiply(2)
print(calc.value)
```

Static Methods vs Instance Methods

Method Type	Access to Instance	Access to Class	Example
Static Method	No	No	Utility function
Instance Method	Yes	No	Operates on instance data

- **Key Difference:** Static methods cannot access or modify instance or class data.

Class Methods

CoGrammar

Purpose of Class Methods

- **Definition**: Class methods operate on the class itself, rather than on instances of the class. They have access to the class-level attributes.
- **Key Use Case**: Often used for methods that modify the class-level data.

Defining and Calling Class Methods

```
class MyClass:
    class_attribute = 0

    @classmethod
    def increment_class_attribute(cls):
        cls.class_attribute += 1

# Calling class method
MyClass.increment class attribute()
print(MyClass.class_attribute)  # 1
```

Class Methods vs Instance Methods

Method Type	Access to Instance	Access to Class	Example
Class Method	No	Yes	Modifies class-level data
Instance Method	Yes	No	Operates on instance data

- **Key Difference:** Static methods cannot access or modify instance or class data.

Best Practices

CoGrammar

Naming Conventions

- Python classes use the CamelCase naming convention
- Each word within the class name will start with a capital letter.
- **E.g.** Student, WeightExercise

```
class Student:
```

```
class WeightExercise:
```

Naming Conventions...

- Give your classes meaningful and descriptive names
- Other developers should already have an idea what your class is for from the name.

BAD

```
class CNum:
```

GOOD

```
class ContactNumber:
```

Single Responsibility

- Make sure your classes represent a single idea.
- If we have a person class where the person can have a pet, we don't want to add all the pet attributes to the person class. We will rather create a new pet class.

```
class Person:

    def __init__(self, name, surname, pet_name, pet_type):
        self.name = name
        self.surname = surname
        self.pet_name = pet_name
        self.pet_type = pet_type
```

Single Responsibility...

```
class Person:
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname

class Pet:
    def __init__(self, name, type):
        self.name = name
        self.type = type
```

Docstrings

- We can document our classes and class methods using docstrings in the same manner we used them with functions.
- Our class docstrings will contain a short description of the class and it's attributes.
- A method docstring will contain a short description of the methods followed by it's parameters and what will be returned.

Docstrings...

```
class Person:
    """
    Class representing a person.

    Attributes:
        name (str): Name of person
        surname (str): Surname of person
    """
    def __init__(self, name, surname):
        """
        Initialise class attributes.

        Parameters:
            name (str): Name of person
            surname (str): Surname of person
        """
        self.name = name
        self.surname = surname
```

Let’s take a short break

CoGrammar

Demo Time!

CoGrammar

Poll

```
class Counter:
    count = 0

    def __init__(self):
        Counter.count += 1

c1 = Counter()
c2 = Counter()
print(Counter.count)
```

1. What will be the output of the following code?

A. 0
B. 1
C. 2

CoGrammar

Poll

```
class Calculator:
    tax_rate = 0.2

    def __init__(self, value):
        self.value = value

    @staticmethod
    def add_tax(amount):
        return amount + amount * tax_rate

calc = Calculator(100)
print(Calculator.add_tax(100))
```

2. What will happen when this code is run?

1. It will print 120.0 because it can access Calculator.tax_rate
2. It will raise a NameError because tax_rate isn't accessible
3. It will print 120

CoGrammar

Conclusion and Recap

- Class attributes are shared by all instances, while instance attributes are unique to each instance.
- Instance methods operate on instance-specific data.
- Static methods don't access or modify class or instance data.
- Class methods access and modify class-level data.

Practical: Simple Car Management System

Objective: Create a Car class to manage car information. The class should:

- Keep track of the total number of cars created.
- Store information about each car (VIN, color, model).
- Have a default manufacturer but allow it to be changed for all cars.
- Validate VIN numbers.
- Display car details.

Resources

- Reading
  - Think Python, 2nd edition
- Official Python Documentation:
  - 9. Classes — Python 3.13.1 documentation
- Online Tutorials:
  - Python's Instance, Class, and Static Methods Demystified
- Indently YouTube Channel
  - Class Methods, Static Methods, & Instance Methods EXPLAINED in Python

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar