Welcome to this CoGrammar tutorial: Classes and Methods

The session will start shortly...

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

Software Engineering Session Housekeeping

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- We would love your **feedback** on lectures: Feedback on Lectures

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** midway and throughout the 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

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

Rafiq Manan

Charlotte Witcher

Nurhaan Snyman

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

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

CoGrammar HyperionDev

Skills Bootcamp Progression Overview

To be eligible for a certificate of completion, students must fulfil three specific criteria. These criteria ensure a high standard of achievement and alignment with the requirements for the successful completion of a Skills Bootcamp.

✓ Criterion 1 - Meeting Initial Requirements

Criterion 1 involves specific achievements within the first two weeks of the program. To meet this criterion, students need to:

- Attend a minimum of 7-8 hours per week of guided learning (lectures, workshops, or mentor calls) within the initial two-week period, for a total minimum of 15 guided learning hours (GLH), by no later than 15 September 2024.
- Successfully complete the Initial Assessment by the end of the first 14 days, by no later than 15 September 2024.

Skills Bootcamp Progression Overview

✓ Criterion 2 - Demonstrating Mid-Course Progress

Criterion 2 involves demonstrating meaningful progress through the successful completion of tasks within the first half of the bootcamp.
To meet this criterion, students should:

- Complete 42 guided learning hours and the first half of the assigned tasks by the end of week 7, no later than 20 October 2024.

CoGrammar

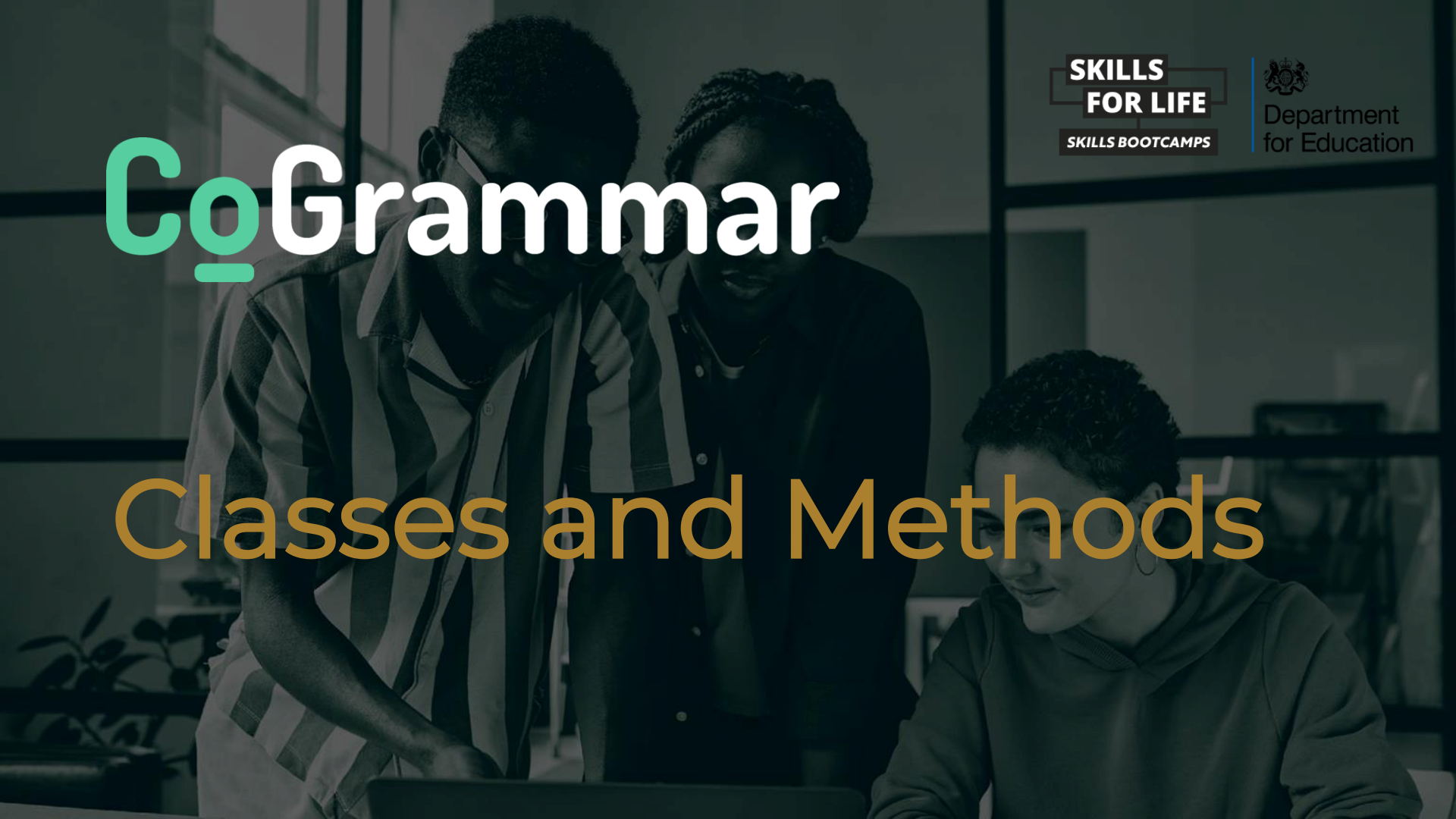
Skills Bootcamp Progression Overview

✓ Criterion 3 - Demonstrating Post-Course Progress

Criterion 3 involves showcasing students' progress after completing the course.
To meet this criterion, students should:

- Complete all mandatory tasks before the bootcamp's end date. This includes any necessary resubmissions, no later than 22 December 2024.
- Achieve at least 84 guided learning hours by the end of the bootcamp, 22 December 2024.

CoGrammar


SKILLS
FOR LIFE
SKILLS BOOTCAMPS
Department
for Education
CoGrammar
Classes and Methods

Learning Outcomes

- Define a class and create instances (objects) of that class.
- Define and use attributes and methods in a class.
- Write constructors to set initial values for object attributes.
- Implement encapsulation by using private attributes and providing public methods to access and modify them.

Learning Outcomes

- Define and access class attributes.
- Define and access instance attributes.
- Define and call static methods using the @staticmethod decorator .
- Define and call class methods using the @classmethod decorator.

Classes

CoGrammar

Classes

- Classes are blueprints for creating objects. They define the properties and behaviours that objects of the class will have.
- Classes encapsulate data (attributes) and functionality (methods) into a single unit, facilitating code organisation and reuse.

Classes...

```
# Define the Car class
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    def drive(self):
        return f"The {self.color} {self.brand} is driving."
```

Attributes

- Attributes represent the state or characteristics of objects. They are the data associated with instances of the class and define what an object of that class looks like.
- Attributes can be variables that store data (instance variables) or methods (instance methods) that define behaviors.

```
# Define the Car class
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color
```

Methods

- Methods are functions defined within a class and they define the behaviours or actions that objects or instances of the class can perform.
- Methods operate on the data (attributes) associated with the class and provide the functionality to manipulate that data.
- Methods can be instance methods, static methods, or class methods.

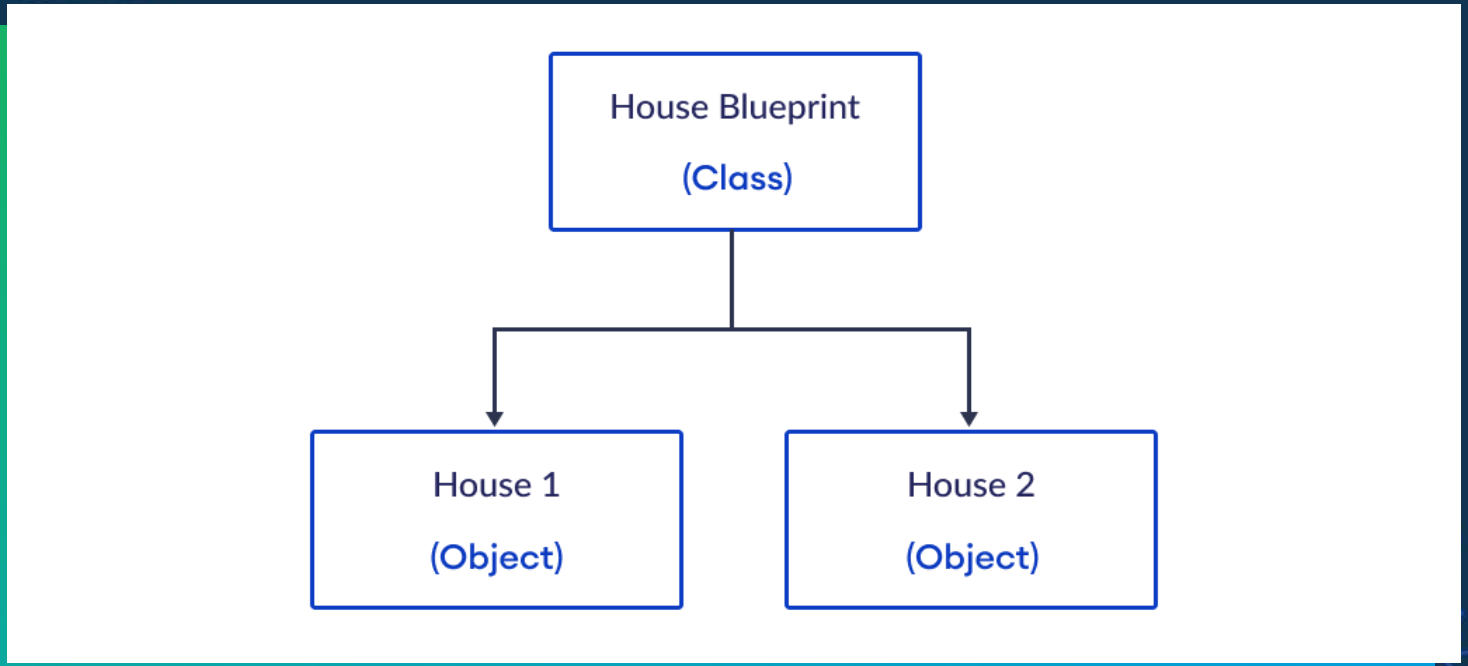
```
class Car:
    def drive(self):
        print("The car is driving.")
```

Objects

- An object is an instance of a class. It is a concrete realisation of the class blueprint, possessing its own unique set of attributes and methods.
- When you create an object, you are essentially creating a specific instance of that class with its own data and behaviour.

```
# Create an object (instance) of the Car class
my_car = Car("Toyota", "red")
```

Objects...



Access Control

CoGrammar

Encapsulation in OOP

- Encapsulation is the concept of bundling the data (attributes) and methods (functions) that operate on that data into a single unit, or class.
- By implementing encapsulation, we can restrict direct access to some of an object's components, promoting controlled access and data protection.
- The core principle here is that we want to hide the internal state of an object and only allow modification through well-defined methods (getters and setters).
- Getters refer to methods that read data and setters refer to methods that update data.

Access Control - Attributes

- Access control mechanisms (public, protected, private) restrict or allow the access of certain attributes within a class.

```python
class MyClass:
    def __init__(self):
        # Public attribute
        self.public_attribute = "I am public"

        # Protected attribute (by convention)
        self._protected_attribute = "I am protected"

        # Private attribute
        self.__private_attribute = "I am private"
```

Access Control - Methods

- Access control mechanisms (public, protected, private) can also restrict or allow the access of certain methods with in a class.

```
def public_method(self):
    return "This is a public method"

def _protected_method(self):
    return "This is a protected method"

def __private_method(self):
    return "This is a private method"
```

Applying the Access Control

```
# Create an instance of MyClass
obj = MyClass()

# Accessing public attributes and methods
print(obj.public_attribute)     # Output: I am public
print(obj.public_method())      # Output: This is a public method

# Accessing protected attributes and methods (not enforced, just a convention)
print(obj._protected_attribute)  # Output: I am protected
print(obj._protected_method())   # Output: This is a protected method

# Accessing private attributes and methods (name mangling applied)
# Note: It's still possible to access, but it's discouraged
print(obj._MyClass__private_attribute)  # Output: I am private
print(obj._MyClass__private_method())   # Output: This is a private method
```

Instance Methods

CoGrammar

Instance Methods

- Instance methods are like actions or behaviours that specific objects can perform.
- These methods have access to the object's data and are defined within the class.
- By using instance methods, we can model how objects interact and behave in our programs, making object-oriented programming a powerful way to structure our code.

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
student1.study()
```

Instance Methods - Example

- We define a Student class with an instance variable name to store the student's name.
- Inside the __init__ method (constructor), when a new Student object is created, we assign the provided name to the name instance variable.
- We define an instance method study() that uses the instance variable name to print a message indicating that the student is studying.
- We create an instance of the Student class (student1) with the name "Alice".
- Finally, we call the study() method on the student1 instance, and it prints "Alice is studying hard!".

Static Methods

CoGrammar

Static Methods

- Static methods are like standalone functions that live within a class.
- They're handy for grouping together related functionality without needing to access specific instance or class data.
- You mark them with the '@staticmethod' decorator to let Python know they're special.

Static Methods - Example

- We define a Car class with a static method honk().
- The honk() method doesn't require access to any specific instance or class variables, so it's marked as a static method using the @staticmethod decorator.
- We can call the static method directly on the class itself (Car.honk()), and it returns "Beep beep!", simulating the sound of a car horn.

```
class Car:
    @staticmethod
    def honk():
        return "Beep beep!"

# Calling the static method
print(Car.honk())  # Output: Beep beep!
```

CoGrammar

Class Methods

CoGrammar

Class Methods

- Class methods are like special functions that belong to the class itself.
- They're not tied to any particular instance but can do cool stuff with the class as a whole.
- You mark them with the '@classmethod' decorator and they get this fancy 'cls' parameter which stands for the class itself. It's a neat way to work with class-level stuff.

Class Methods - Example

```python
class Car:
    num_cars_sold = 0  # Class variable to keep track of the number of cars sold

    def __init__(self, brand):
        self.brand = brand
        Car.num_cars_sold += 1  # Increment the number of cars sold when a new car is created

    @classmethod
    def get_num_cars_sold(cls):
        return cls.num_cars_sold

# Creating instances of the Car class
car1 = Car("Toyota")
car2 = Car("Honda")

# Accessing the class method to get the number of cars sold
print(Car.get_num_cars_sold())  # Output: 2
```

Class Methods - Example

- We define a Car class with a class variable num_cars_sold to keep track of the number of cars sold.
- Inside the __init__ method (constructor), every time a new car object is created, we increment the num_cars_sold class variable.
- We define a class method get_num_cars_sold() using the @classmethod decorator, which returns the current number of cars sold.
- We create two instances of the Car class (car1 and car2).
- We then call the class method get_num_cars_sold() using the class name Car, and it returns the total number of cars sold, which is 2 in this case.

Let’s take a short break

CoGrammar

Demo Time!

CoGrammar

Conclusion and Recap

CoGrammar

Conclusion and Recap

- Classes: Blueprints for creating objects that encapsulate both data (attributes) and functionality (methods).
- Encapsulation: Classes group related attributes and methods, promoting code organisation and reusability.
- Instance Methods: Operate on individual objects and can access and modify instance attributes.
- Class Methods: Operate on the class itself and can modify class-level data. Defined using the @classmethod decorator.
- Static Methods: General utility methods that don't depend on class or instance data. Defined using the @staticmethod decorator.

Conclusion and Recap

Where You'll Use Classes and Methods:

- Web Development: Defining user accounts, products, and transactions as objects with related methods.
- Data Analysis: Classes represent data structures, and methods perform operations on them (e.g., filtering or calculating).
- Game Development: Characters, levels, and rules are modelled using classes, with methods dictating their actions and behaviours.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar