Welcome to the Lecture

CoGrammar

Classes II

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

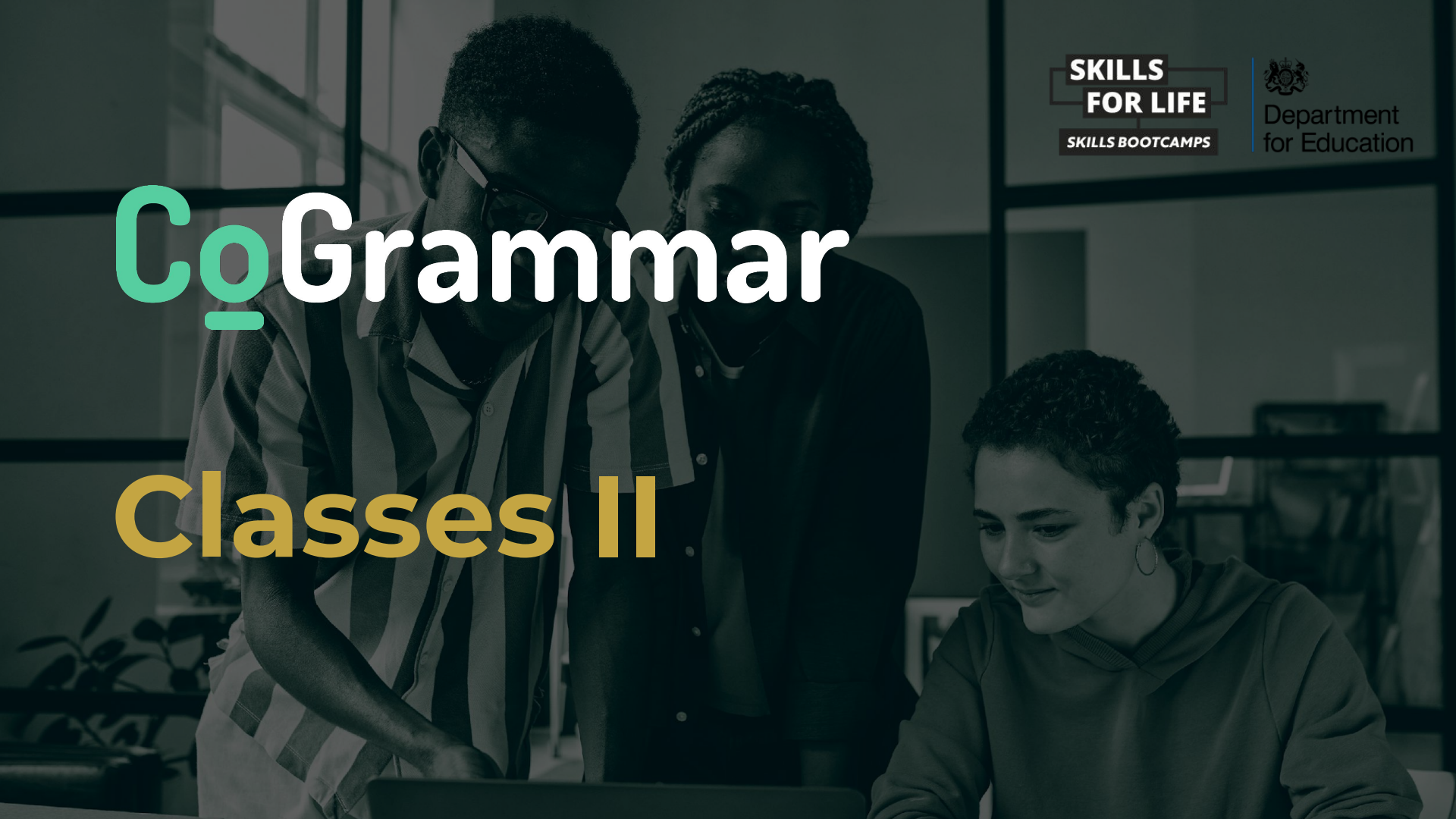
Ronald Munodawafa

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
Classes II

Poll

What is the role of the `__init__` method in a Python class?

A. It is a destructor method used to clean up resources when an object is deleted.
B. It is used to define class-level methods that can modify class attributes.
C. It initialises object attributes when an object is created.

Poll

What is the purpose of the `@staticmethod` in the following code?

```
1   class Math:
2       @staticmethod
3       def add(a, b):
4           return a + b
5
6   print(Math.add(3, 5))
```

a. It allows the method to modify instance attributes.
b. It allows the method to be called without creating an instance of the class.
c. It allows the method to access class attributes.

Learning Objectives & Outcomes

- Define inheritance and its role in object-oriented programming.
- Define inheritance key components (base class, subclass, superclass).
- Implement and utilise the principles of inheritance within Python projects.
- Explain and implement method overriding.
- Utilise `super()` to access parent class methods.

Introduction

CoGrammar

Introduction

Basic Vehicle

All Derived from **Basic Vehicle**










1st Gen → 2nd Gen → 3rd Gen → 4th Gen

What is Inheritance

CoGrammar

What is Inheritance?

- Intuition and Definition:
  - Sometimes we require a class with the same attributes and properties as another class but we want to extend some of the behaviour or add more attributes.
  - **Inheritance** is a is a fundamental concept in object-oriented programming where a new class (the subclass) is derived from an existing class (the superclass or base class), inheriting its attributes and methods.
- Real-world analogy:
  - Think of biological inheritance - children inheriting traits from their parents.

Building Better Software: The Advantages of Inheritance

- Why Use Inheritance?
  - **Code Reuse**: Inheritance allows for reuse of code from parent classes.
  - **Organized Structure**: Creates a clear hierarchical relationship between classes.
  - **Extensibility**: Enables easy extension and modification of existing classes without altering their code.

Building Better Software: The Advantages of Inheritance

- Real-world Example: Animal Kingdom
  - **Animal**: Parent class to all animals with common traits (e.g., eat, sleep).
  - **Dog**: Inherits from Animal, adding specific Dog traits (e.g., bark, breed).
- Advantages of Inheritance:
  - **Code Reuse**: Shared behaviors like eating and sleeping are defined once in Animal.
  - **Organized Structure**: Clear hierarchy (Animal → Dog) makes relationships easy to follow.
  - **Extensibility**: Dog adds unique traits without changing Animal.

Key Terms in Inheritance

CoGrammar

Key Terminology in Inheritance

- **Superclass** (Parent/Base Class): The class being inherited from.
- **Subclass** (Child/Derived Class): The class inheriting from the superclass.
- Inheritance Hierarchy: The structure that shows the relationship between the parent and child classes.

Basic Inheritance Syntax in Python

CoGrammar

Basic Inheritance Syntax

```
class ParentClass:
    # Parent class code

class ChildClass(ParentClass):
    # Child class code
```

Basic Inheritance Example

```
class Animal:  # Superclass
    def __init__(self, number_of_legs):
        self.number_of_legs = number_of_legs

    def speak(self):
        return "Animal sound"
    def eat(self):
        return "Animal is eating"

class Dog(Animal):  # Subclass
    pass            # Nothing Explicitly implemented

my_dog = Dog(4)
print(my_dog.speak())  # Inherited speak() method
print(my_dog.eat())    # Inherited eat() method
print(f"My dog has {my_dog.number_of_legs} legs.") # Inherited
attribute
```

Understanding Method Overriding

CoGrammar

What is Method Overriding?

- Intuition and Definition:
  - We can override methods in our subclass to either extend or change the behaviour of a method.
  - **Method Overriding** The process of re-implementing a method already implemented in a superclass.
  - To apply method overriding, you simply need to define a method with the same name as the method you would like to override.
- Real-world analogy:
  - Think of the Dog subclass overrides the speak() method from the Animal superclass to provide a specific sound, like barking, instead of a generic sound.

Example of Method Overriding

```python
class Animal:
    def speak(self):
        # Base class method: generic sound for any animal
        return "makes a sound"

class Dog(Animal):
    def speak(self):
        # Overridden method: specific sound for dogs
        return "barks"

# Example usage:
generic_animal = Animal()  # Create an instance of the base class
print(generic_animal.speak())  # Output: makes a sound (generic sound for animals)

my_dog = Dog()  # Create an instance of the subclass
print(my_dog.speak())  # Output: barks (specific sound for dogs)
```

Unleashing the Power of `super()` in Python

CoGrammar

What is `super()`?

- Intuition and Definition:
  - When extending functionality of a method we would first want to call the base class method and then add the extended behaviour.
  - When adding more instance attributes to the subclass we have to call to the base class `__init__()` (**the Constructor**) to avoid having to redeclare all our base class attributes.
  - `super()` is a built-in function that allows a subclass to call methods from its superclass.

super(): Constructor Example

```python
class Animal:
    def __init__(self, name, age):
        self.name = name  # Initializes the 'name' attribute
        self.age = age    # Initializes the 'age' attribute

class Dog(Animal):
    def __init__(self, name, age, breed):
        super().__init__(name, age) # Calls Animal's __init__ to initialize 'name' and 'age'
        self.breed = breed  # Initializes the 'breed' attribute

# Example usage
my_dog = Dog("Buddy", 5, "Golden Retriever")
print(my_dog.name)    # Output: Buddy
print(my_dog.age)     # Output: 5
print(my_dog.breed)   # Output: Golden Retriever
```

When to Use `super()`?

- Beyond Constructors:
  - While `super()` is often used to call a parent class's constructor, its functionality extends beyond this. You can also use super() to call any parent class method that has been overridden in the subclass.
- Why use `super()` with other functions?
  - Maintaining Parent Class Behavior: Call the parent class's method to keep its original functionality.
  - Avoiding Redundancy:Reuse common code from the parent class.

super(): Example

```
class Animal:
    def speak(self):
        return "makes a sound"

class Dog(Animal):
    def speak(self):
        # Calls the parent class's speak method and extends it
        return super().speak() + " and barks"

# Example usage:
my_dog = Dog()
print(my_dog.speak())   # Output: makes a sound and barks
```

Extending from the base class functionality

Calling from the base class

CoGrammar

Poll

What will be the output of the following code?

```
class Vehicle:
    def start(self):
        return "Vehicle starting"

class Car(Vehicle):
    def start(self):
        return "Car starting"

my_car = Car()
print(my_car.start())
```

1. Vehicle starting
2. Error: start() method is not defined
3. None
4. Car starting

CoGrammar

Poll

What will be the output of the following code?

```
1   class Animal:
2       def __init__(self, name):
3           self.name = name
4
5   class Dog(Animal):
6       def __init__(self, name, breed):
7           super().__init__(name)
8           self.breed = breed
9
10  my_dog = Dog("Max", "Beagle")
11  print(my_dog.name)
12  print(my_dog.breed)
```

1. Max and Dog
2. Beagle and Dog
3. Max and Beagle
4. Error: super() cannot be used in the constructor

Lesson Conclusion and Recap

Recap the key concepts and techniques covered during the lesson.

- **Inheritance** allows a subclass to inherit attributes and methods from a superclass, enabling code reuse and structured organisation.
- **Superclass and Subclass**: The superclass (parent) provides the inherited properties, while the subclass (child) extends or modifies them.
- **Method Overriding**: Subclasses can override inherited methods to provide specific implementations, allowing customization.
- **`super()`**: The `super()` function allows subclasses to call methods from the superclass, often used in constructors or overridden methods.
- **Benefits**: Inheritance simplifies code by reusing functionality, enhancing extensibility, and maintaining a clear hierarchy.

Follow-up Activity: Vehicle Rental System

- **Objective:** To create a Vehicle Rental Service using inheritance, where each vehicle type (Car, Bike) can calculate its rental cost based on the number of days rented. The system should handle exceptions such as negative days and invalid input types.
- **Steps to Implement:**
  - **Base Class: Vehicle**
    - **Attributes:** make, model, year
    - Methods
      - `get_details()`
      - `rent()`
      - `calculate_rental_cost(days)`
  - **Subclasses: Car, Bike**
    - Inherit attributes and methods from `Vehicle`
    - Override `rent()` and `calculate_rental_cost()` for specific implementations
    - Handle exceptions for invalid input

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar