



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

- 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 at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- 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](#)

Skills Bootcamp

8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✓ Criterion 1: Initial Requirements

- **Timeframe:** First 2 Weeks
- **Guided Learning Hours (GLH):**
Minimum of 15 hours
- **Task Completion:** First four tasks

Due Date: 24 March 2024

✓ Criterion 2: Mid-Course Progress

- **Guided Learning Hours (GLH):** 60
- **Task Completion:** 13 tasks

Due Date: 28 April 2024

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

Completion: All mandatory tasks,
including Build Your Brand and
resubmissions by study period end
Interview Invitation: Within 4 weeks
post-course
Guided Learning Hours: Minimum of
112 hours by support end date
(10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship
Outcome: Document within 12
weeks post-graduation
Relevance: Progression to
employment or related
opportunity



**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Tutorial: Classes and Methods

April 2024

Learning objectives

- ❖ Classes
- ❖ Attributes
- ❖ Instance
- ❖ Static and
- ❖ Class Methods

Classes



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 organization 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 behaviours.

```
# 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 that define the behaviors or actions that objects of the class can perform.
- ❖ They 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.

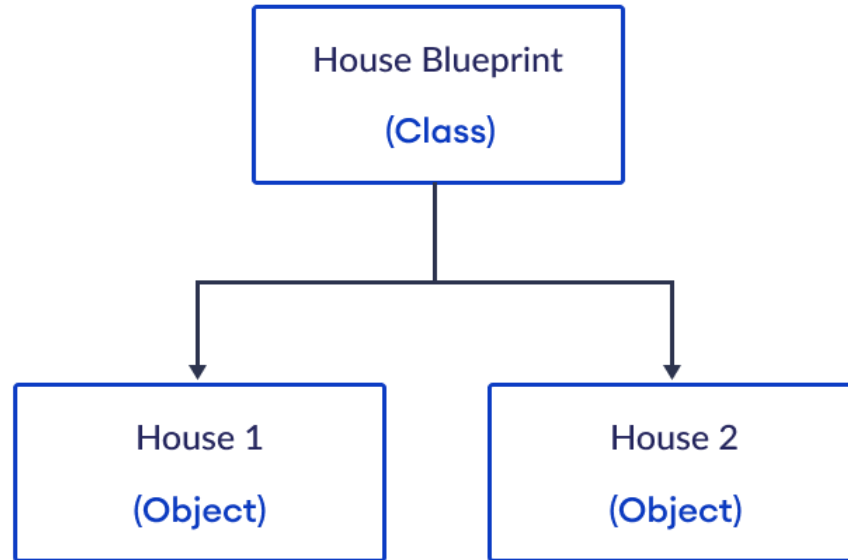
```
class Car:  
    def drive(self):  
        print("The car is driving.")
```

Objects

- ❖ An object is an instance of a class. It is a concrete realization 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 behavior.

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

Objects cont.



Pillars of OOP

Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.

Polymorphism

- Polymorphism allows us to access these overridden methods and attributes that share the same name as the base class.

Pillars of OOP

Encapsulation

- Encapsulation in Python is the concept of wrapping data (variables) and methods (functions) into a single unit (class) and restricting access to the direct modification of an object's attributes.

Abstraction

- Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object. In a way, it is an extension of encapsulation, focusing on hiding the complexity and only exposing a high-level interface.

Static Methods



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

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

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

- 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 Methods



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

```
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

- ❖ 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.

Summary



Summary

- ❖ **Classes** provide a way to structure and organize code, attributes represent the state of objects, and instance, static, and class methods define behaviours and operations associated with classes and objects.
- ❖ **Static methods** are self-contained functions within a class that do not require access to instance or class variables.
- ❖ **Class methods** operate on the class itself and receive the class as their first parameter, allowing them to access and modify class variables.
- ❖ Both static and class methods provide ways to encapsulate functionality within a class and promote code organization and reusability.

Summary: Class vs Static Method

- ❖ A class method takes `cls` as the first parameter while a static method needs no specific parameters.
- ❖ A class method can access or modify the class state while a static method can't access or modify it.
- ❖ Static methods know nothing about the class state, are utility-type methods that take some parameters and work upon those parameters. Class methods must have class as a parameter.

Summary: Class vs Static Method

- ❖ We use `@classmethod` decorator in python to create a **class method** and we use `@staticmethod` decorator to create a **static method** in python.
- ❖ We generally use the **class method** to create **factory methods**. Factory methods return class objects (similar to a constructor) for different use cases.
- ❖ We generally use **static methods** to create **utility functions**.

**Let's take a short
break**

CoGrammar



Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

