



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** 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

**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Classes and Methods

Learning Outcomes

- Distinguish between classes and objects.
- Create classes with attributes and methods.
- Use the dot operator to interact with objects.

The Building Blocks of OOP: Classes and Objects



Classes

- A *class* is a blueprint or template for creating *objects*. Think of a house blueprint – it describes the house but isn't a house itself.
- A *class* defines the type of object. It specifies the data (*attributes*) objects will hold and the actions (*methods*/behaviors) they can perform.
- Classes organize code and model real-world entities.

Objects

- An *object* is an *instance* of a class – an actual house built from the *blueprint*.
- Each object has its own data (*attributes*) and can perform the actions (*methods*) defined by its class.
- Objects are the active elements in OOP, interacting to perform tasks.

Interacting with Objects: The Dot Operator

- The *dot operator* (.) is how we interact with objects.
- It's used for:
 - **Accessing attributes:** `object.attribute` (e.g., `my_dog.name`).
 - **Calling methods:** `object.method()` (e.g., `my_dog.bark()`).
- The dot operator is how you "tell an object to do something" or "ask for information."

Actions and Behaviors: Methods



Instance Methods

- *Instance methods* operate on a *specific object's data*.
- `self` refers to the *current object* within the method, allowing *access* and *modification* of its attributes.
- **Example:** `my_dog.bark()` – the dot operator calls the `bark()` method on `my_dog`.

Class Methods

- *Class methods* operate on the *class itself*.
- Use the `@classmethod` decorator. `cls` (by convention) represents the class.
- **Use cases:** class-level information tracking.
- **Example:** `Dog.get_num_dogs()` – the dot operator calls the class method.

Static Methods

- *Static methods* are related to the class but *don't need* access to the *class* or any *object*.
- Use the `@staticmethod` decorator. No `self` or `cls`.
- **Use cases:** utility functions logically grouped with the class.
- **Example:** `Dog.is_mammal()` – the dot operator calls the static method.

Data Management and Protection



Instance vs. Class Attributes

- *Instance attributes* belong to each object; *class attributes* are shared by all objects.
- Instance attributes are defined in `__init__` using `self`; class attributes are defined at the class level.
- **Example:** `my_dog.name` (instance), `Dog.num_dogs` (class).

Encapsulation and Data Hiding

- *Encapsulation* bundles data (attributes) and methods within a class.
- *Data hiding* restricts direct access to internal data for data integrity.
- `_` prefix indicates "internal use" (convention, not strict privacy).
- *Getters* (access) and *setters* (modify) provide controlled access and data validation.
- **Example:** `my_dog.get_age()`, `my_dog.set_age(5)`.

The Power of OOP: Principles and Practices



Review of Key Concepts

- **Classes:** Blueprints.
- **Objects:** Instances.
- **Attributes (Instance/Class):** Data.
- **Methods (Instance/Class/Static):** Actions.
- **Dot Operator:** Accessing attributes and calling methods.
- **Encapsulation:** Data protection.

Core OOP Principles

- **Abstraction:** Hiding complexity, showing only essential information (like driving a car).
- **Encapsulation:** (Recap) Bundling and protecting data.
- **Modularity:** Breaking down programs into self-contained modules (classes).
- **Reusability:** Using classes multiple times.

Conclusion and Recap



Conclusion and Recap

- **Classes are blueprints, objects are instances:** Classes define the structure (attributes) and behavior (methods) for creating objects, which are concrete instances of those classes.
- **Methods define object behavior:** Instance methods operate on specific object data using **self**, while class and static methods offer class-level or utility functions.
- **Attributes store object data:** Instance attributes hold individual object data, while class attributes are shared across all objects of a class.
- **Encapsulation protects data:** Encapsulation bundles data and methods, using getters and setters to control access and maintain data integrity.

Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

