



Welcome to this **CoGrammar** Lecture: Classes Part 2

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

60 Guided Learning Hours

Data Science - **13 tasks**

Software Engineering - **13 tasks**

Web Development - **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

Attributes, Instance, Static and Class Methods

April 2024

Learning objectives

Classes (Recap), Attributes

Three types of methods for a class

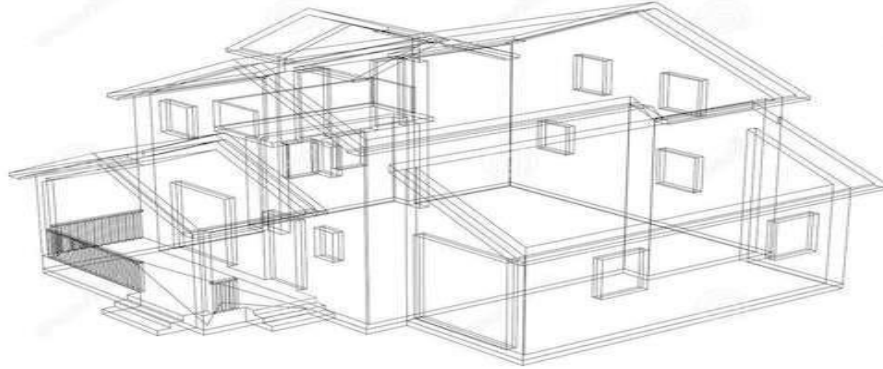
- ❖ Instance
- ❖ Static
- ❖ Class Methods

Classes



Classes Recap

A class is a blueprint or template for creating objects. It defines the attributes and methods that all objects of that class will have.



Attributes

- Attributes are **values** that define the characteristics associated with an object.
- They define the **state** of an object and provide information about its **current condition**.
- For a class named 'House', some relevant attributes could be:
 - **number_of_bedrooms**
 - **year_built**

Methods (Behaviours)

- Methods, also known as behaviours, define **the actions or behaviors** that objects can perform
- They encapsulate the functionality of objects and allow them to **interact with each other** and the outside world.
- For a class named 'House', some relevant method could be:
 - **set_location()**: Allows updating the location of the house

Methods

In a Python class, you can define three different types of methods:

- ❖ **Instance methods**, which take the current instance, `self`, as their first argument (most common)
- ❖ **Class methods**, which take the current class, `cls`, as their first argument
- ❖ **Static methods**, which take neither the class nor the instance

Instance Methods



What are Instance Methods ?

Instance methods are like *actions or behaviours that specific objects can perform*. They 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 Method Example

```
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()
```


What are Static Methods ?

Static methods are like *standalone functions* that are associated with a class. They're useful for organizing *utility functions* that are related to the class but *don't depend on individual object instances*. By using static methods, we can *group* together *related functionalities within a class* and make our code more organized and modular.

Static Method Example

```
class MathUtil:  
    @staticmethod  
    def add(x, y):  
        return x + y  
  
# Calling the static method  
result = MathUtil.add(3, 5)  
print(result) # Output: 8
```

Class Methods



What are Class Methods ?

- ❖ In object-oriented programming, a class method is like a *special function that belongs to the class itself*, rather than to any specific object created from the class. It's a way for classes to *define functions that operate on the class itself*, rather than on individual objects.
- ❖ They're useful for *defining functionalities that affect the entire class*, such as modifying class attributes or performing operations that involve the class as a whole.

Class Methods Examples

```
class MathUtil:
    @classmethod
    def calculate_average(cls, num_list):
        total = sum(num_list)
        return total / len(num_list)

# Using the class method
numbers = [10, 20, 30, 40, 50]
average = MathUtil.calculate_average(numbers)
print(average) # Output: 30.0
```

Best Practices



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 users 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 that can have a pet we won't add all the pet attributes to the person class. We will rather create a new 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 its attributes.
- ❖ A method docstring will contain a short description of the methods followed by its parameters and what will be returned.

Docstrings cont.

```
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
```

Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

