CoGrammar

OOP: Components & Best Practices

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Software Engineering Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(FBV: Mutual Respect.)**
- 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 Open Classes. You can submit these questions here: Open Class Questions

Software Engineering Lecture 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

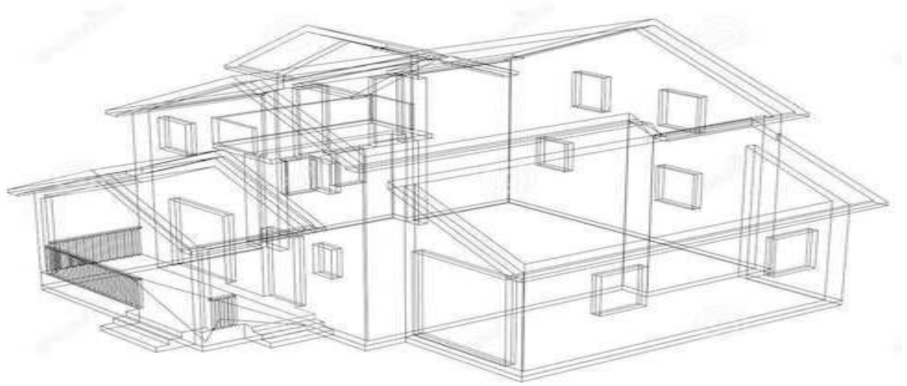
Lecture Objectives

1. Identify common best practices when working with classes and objects and apply these practices to your code.

CoGrammar

Classes in OOP

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



Attributes (Properties)

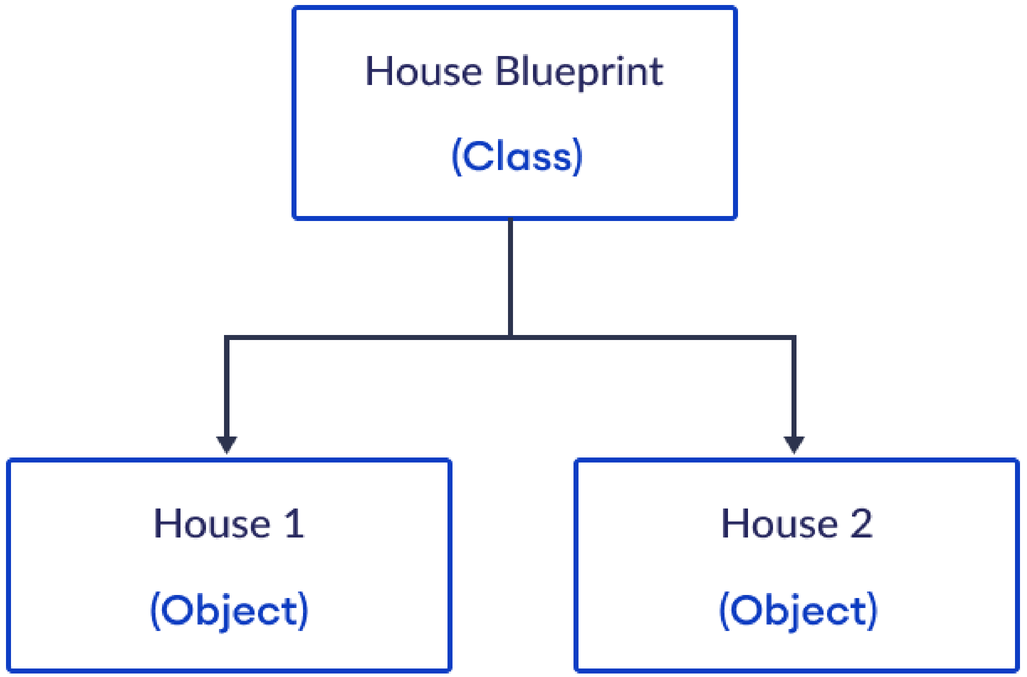
- Attributes, also known as properties or data members are the characteristics associated with an object.
- Attributes define the state of an object and provide information about its current condition.
- For a class named ‘House’, some relevant attributes could be:
  - number of bedrooms
  - year built

Methods (Behaviours)

- Methods, also known as behaviours, define the actions or behaviours 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


House Blueprint
(Class)
House 1
(Object)
House 2
(Object)

What is object-oriented programming (OOP)?

A. A programming paradigm that emphasizes creating objects to represent real-world entities.

B. A programming language that is specifically designed for object-oriented programming.

C. A programming methodology that focuses on code reusability and maintainability.

What is the difference between a class and an object?

A. A class is a blueprint or template for creating objects, while an object is an instance of a class.
B. A class is a set of instructions that define the behaviour of an object, while an object is a physical entity that exists in memory.
C. A class is a high-level abstraction that represents a general concept, while an object is a specific instance of that concept.

What is an attribute?

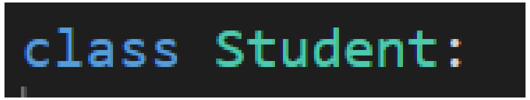
A. A variable within an object that holds data.
B. A procedure or function associated with an object that encapsulates data and behaviour.
C. A collection of data and the methods that operate on that data.

Best Practices

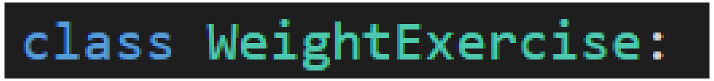
CoGrammar

Naming Convention

- 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:
```

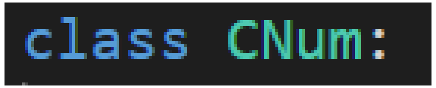


```
class WeightExercise:
```

Use Meaningful Names

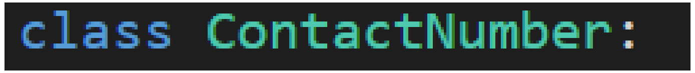
- Give your classes meaningful and descriptive names.
- Other developers should already have an idea what your class is used for when they read the class name.

Not Good



```
class CNum:
```

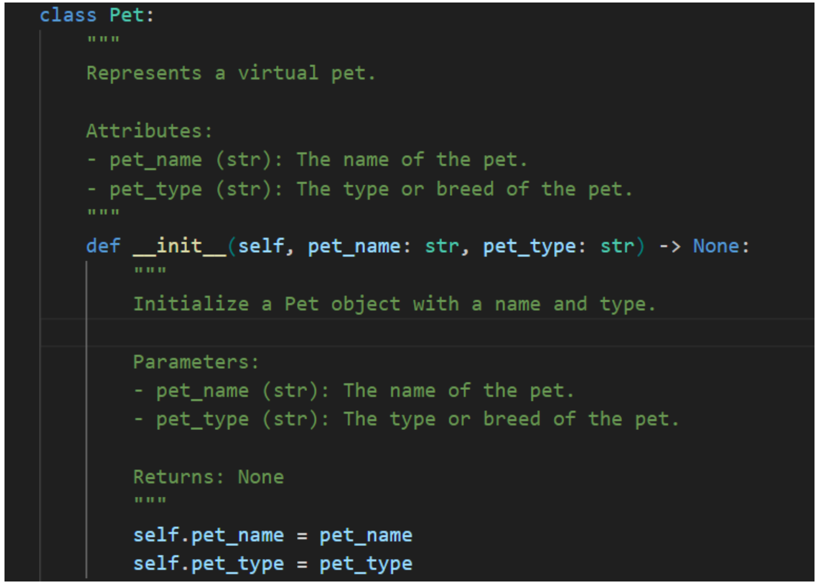
Good



```
class ContactNumber:
```

Docstrings

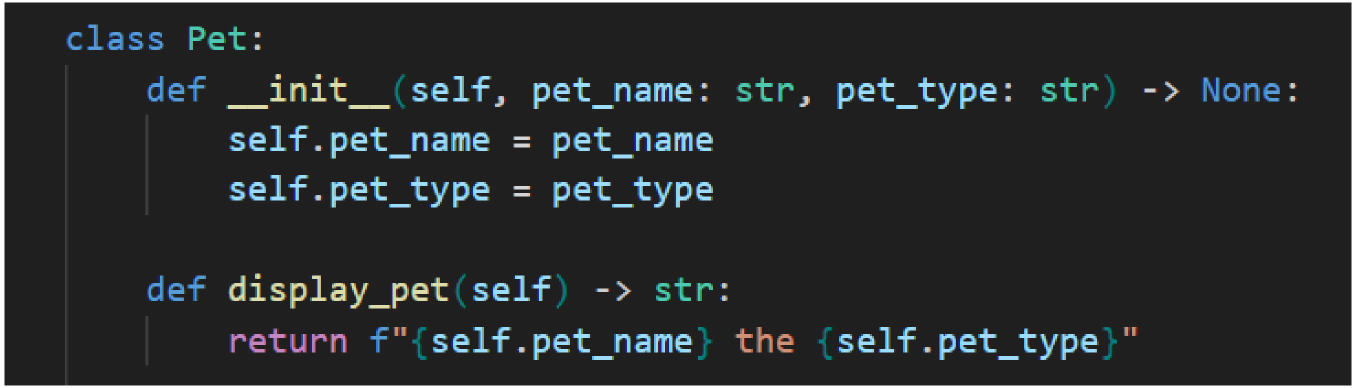
- A form of documenting your classes and methods. Think of these as user manuals for each function.



```python
class Pet:
    """
    Represents a virtual pet.

    Attributes:
    - pet_name (str): The name of the pet.
    - pet_type (str): The type or breed of the pet.
    """
    def __init__(self, pet_name: str, pet_type: str) -> None:
        """
        Initialize a Pet object with a name and type.

        Parameters:
        - pet_name (str): The name of the pet.
        - pet_type (str): The type or breed of the pet.

        Returns: None
        """
        self.pet_name = pet_name
        self.pet_type = pet_type
```

Type Annotations

- Type hints make your code more understandable by offering developers to see what types of arguments a class or method expects while giving them an idea of what will be returned.



```python
class Pet:
    def __init__(self, pet_name: str, pet_type: str) -> None:
        self.pet_name = pet_name
        self.pet_type = pet_type

    def display_pet(self) -> str:
        return f"{self.pet_name} the {self.pet_type}"
```

CoGrammar

Encapsulation

- Encapsulate data within your classes and limit access to the data through methods.
- We want to hide the internal state and implementation of the object from the outside world.
- Access Modifiers
    - **Public:** Attributes and methods used for accessibility from outside the class.
    - **Protected:** Attributes and methods used for accessibility within the class and its subclasses.
    - **Private:** Attributes and methods used for accessibility only within the class itself.

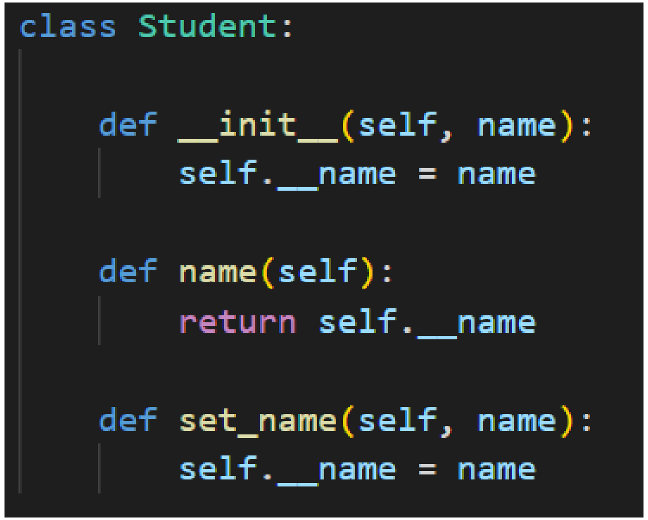
CoGrammar

Encapsulation (Continue)

- In python we cannot enforce encapsulation but we can use naming convention to show which properties and attributes should be accessible outside of the class.
- We can add a single underscore at the start of a variable name to signify that it should be protected. E.g. _name or _grades
- We can add 2 underscores at the start of a variable name to signify that it should be private. E.g. __name or __grades

Encapsulation

- We can now add getter and setter methods to our class to control access to data.

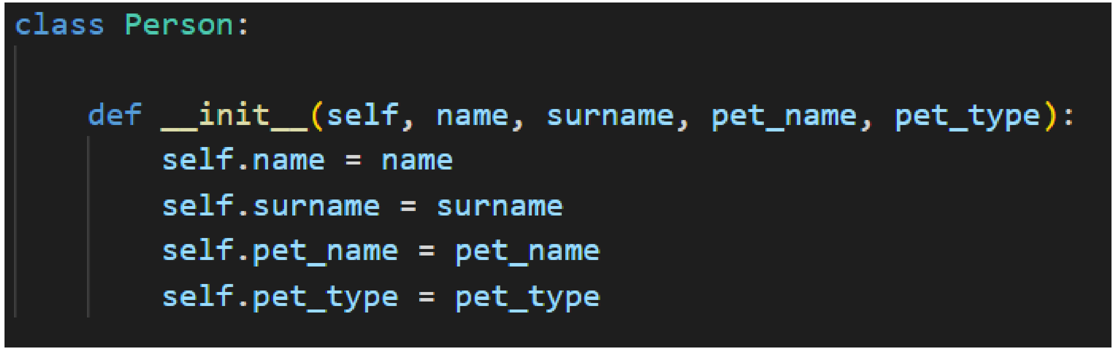


```python
class Student:

    def __init__(self, name):
        self.__name = name

    def name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name
```

Single Responsibility

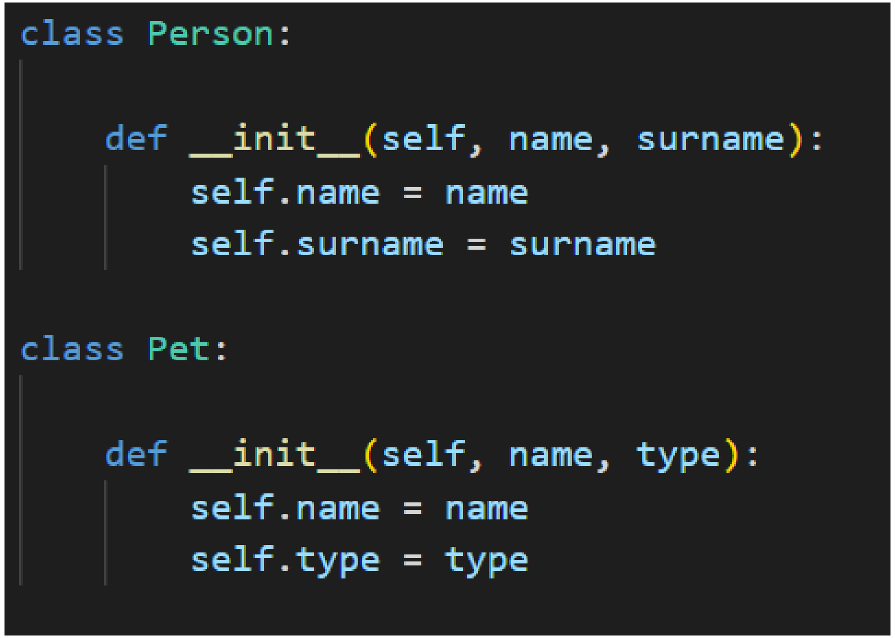
- Make sure your classes represent a single idea.

- 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.



```python
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



```python
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
```

Challenge:

Explain the scope and purpose of instance variables in OOP?

Wrapping Up

Classes

A class is a blueprint for creating objects. Objects are instances of a class, and they encapsulate attributes and the methods that operate on those attributes.

Attributes and Methods of a Class

Attributes are variables that store data within a class or an object. Methods are functions within a class that operate on its data.

Progression Criteria

✓ **Criterion 1: Initial Requirements**

- Complete 15 hours of Guided Learning Hours and the first four tasks within two weeks.

✓ **Criterion 2: Mid-Course Progress**

- Software Engineering: Finish 14 tasks by week 8.
- Data Science: Finish 13 tasks by week 8.

✓ **Criterion 3: Post-Course Progress**

- Complete all mandatory tasks by 24th March 2024.
- Record an Invitation to Interview within 4 weeks of course completion, or by 30th March 2024.
- Achieve 112 GLH by 24th March 2024.

✓ **Criterion 4: Employability**

- Record a Final Job Outcome within 12 weeks of graduation, or by 23rd September 2024.

CoGrammar

Questions around classes

CoGrammar

Thank you for joining