Software Engineering Bootcamp

Hyperiondev

Object Oriented Programming (OOP)

Lecture – Housekeeping

- ❑ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- ❑ No question is daft or silly - **ask them!**
- ❑ There are Q/A sessions at the end of the session, should you wish to ask any follow-up questions.
- ❑ For all non-academic questions, please submit a query: www.hyperiondev.com/support
- ❑ Report a safeguarding incident: http://hyperiondev.com/safeguardreporting

Objectives

1. Explain what object-oriented programming is.
2. Explain what a class is in Python.
3. Explain what class methods and attributes are.
4. Create and use classes within your projects.
5. Explain Inheritance
6. Explain the purpose of inheritance
7. Explain what method overriding is.
8. Implement inheritance within your projects.

OOP

What is object oriented programming?

OOP is a way of organizing code around objects, which are self-contained modules that contain both data and instructions that operate on that data.

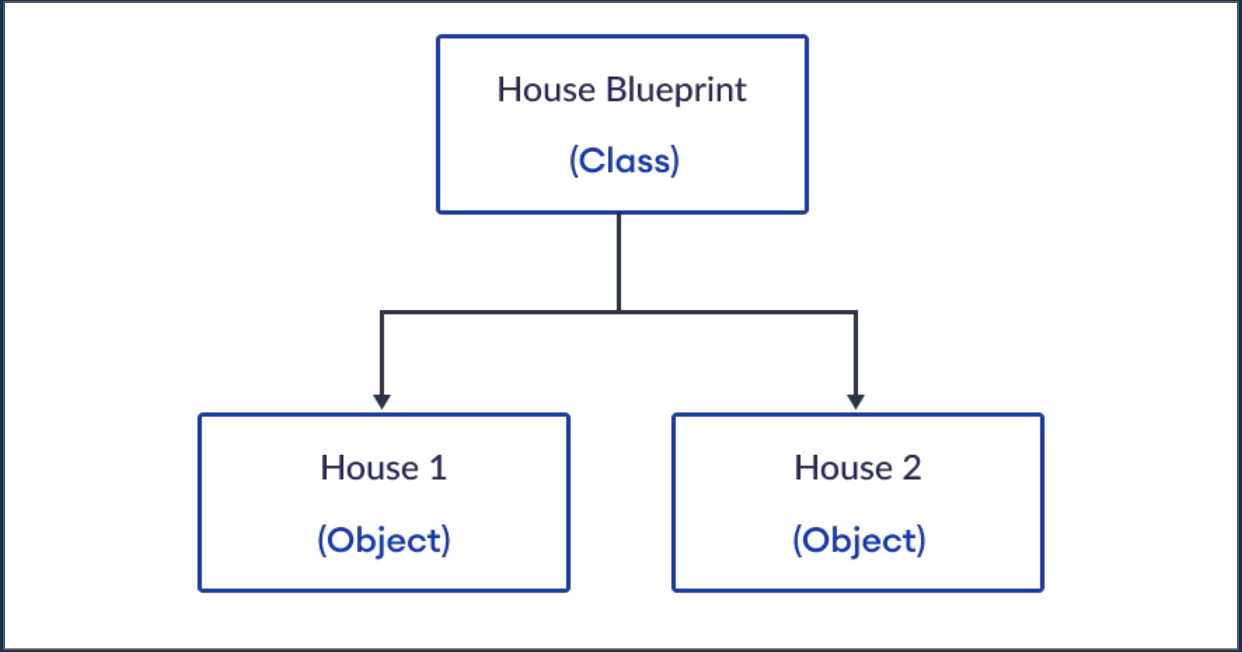
Why OOP?

- Promotes encapsulation by bundling data and behaviour together within objects.
- Promotes abstraction by focusing on essential characteristics and behaviours of objects, hiding the underlying implementation details.
- Promotes code organisation into independent modules called classes. This separation of concerns allows developers to focus on specific tasks without worrying about the intricacies of other parts of the program.
- Reduces code duplication and simplifies development effort.

Classes

- A class in Python is like a blueprint for creating objects.
- It defines a set of attributes and methods that the created objects of the class can use.
- Attributes are the characteristics of an object, while methods are the operations that an object can perform.

Classes



Class Properties

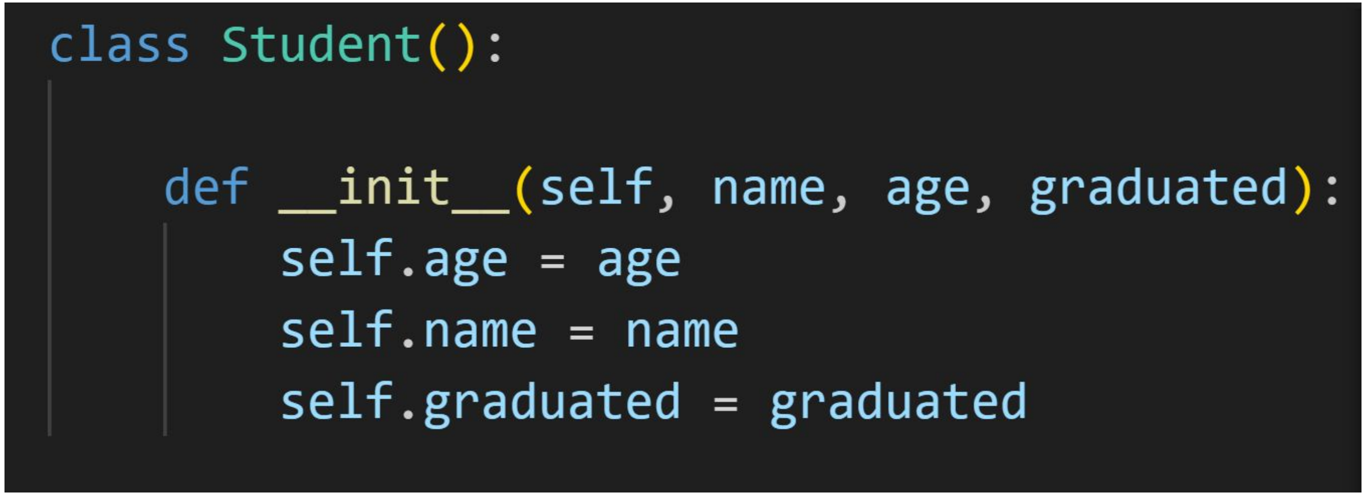
- Each class can have two main things: **attributes** and **methods**.
  - **Attributes** are variables that belong to a class. They represent the properties or characteristics of the class that objects can have.
  - **Methods** are functions that belong to a class. They define the behaviors or actions that an object of the class can perform.

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

Attributes

- To instantiate a class the __init__() function will be called this will allow us to initialise all of our attributes.



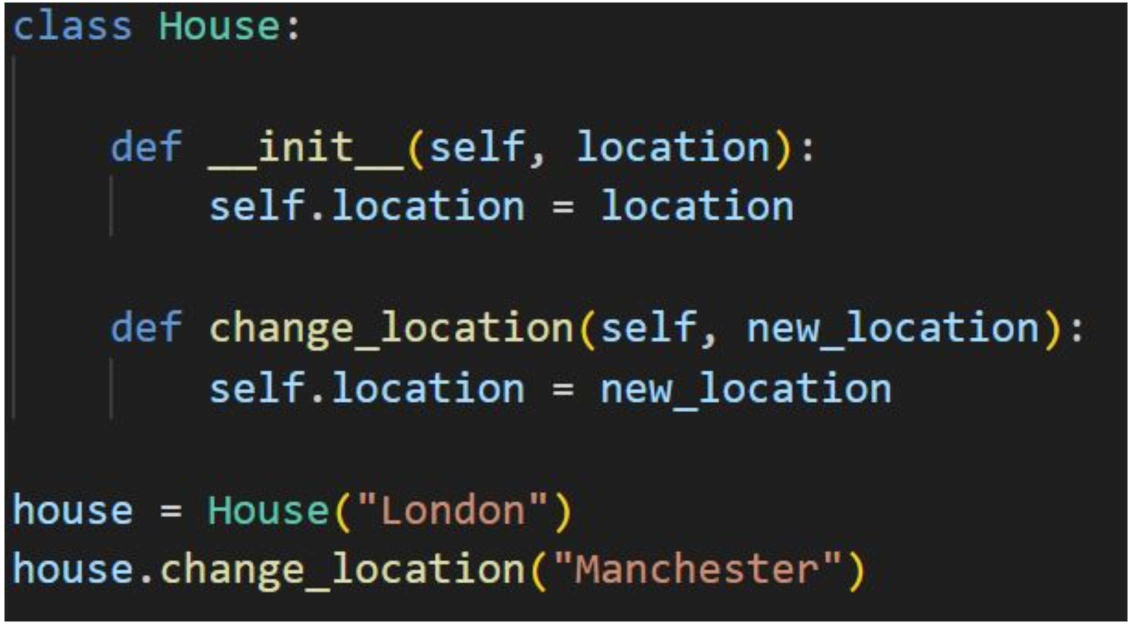
```
class Student():

    def __init__(self, name, age, graduated):
        self.age = age
        self.name = name
        self.graduated = graduated
```

Hyperiondev

Methods

- Methods, 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



```python
class House:

    def __init__(self, location):
        self.location = location

    def change_location(self, new_location):
        self.location = new_location

house = House("London")
house.change_location("Manchester")
```

Hyperiondev

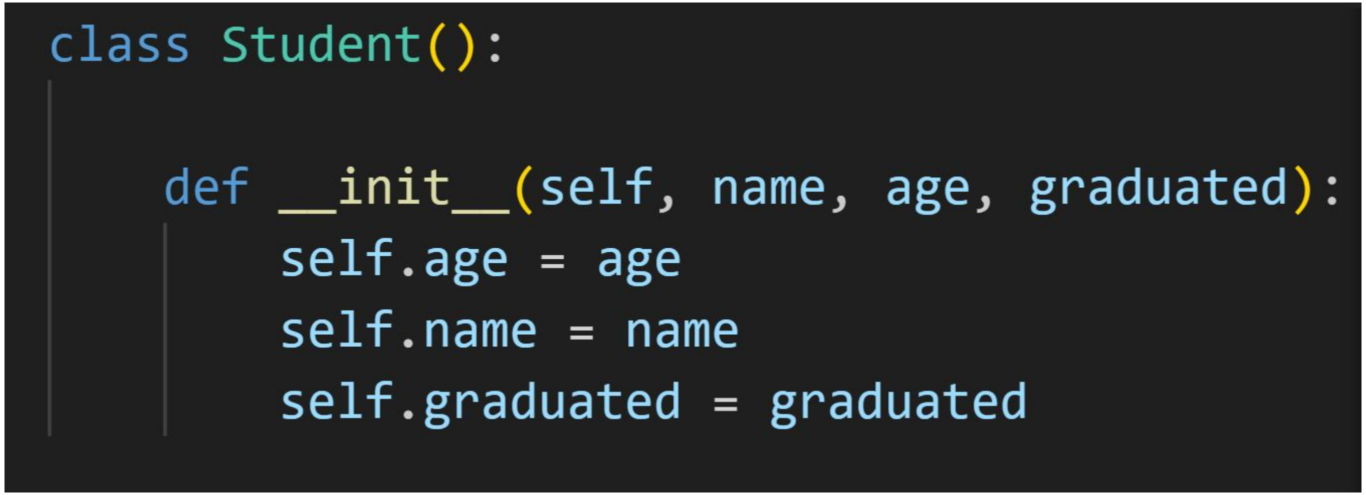
Objects

- An object is a fundamental building block that represents a real-world entity or concept. It encapsulates both data and behaviour.
- Objects represent key characteristics or attributes of real world entities.
- Objects also encapsulate the actions or behaviours associated with real-world entities.

Objects

- In Python, everything is an **object**. Every entity, including data values and functions, are considered objects.
- They allow you to hide the internal implementation details of data and only expose methods for interacting with data.
- Without knowing it, you have actually been using objects in Python.
- For example: string.split() - this uses the split() method present in the string object.
- Imagine needing to call split(string, delimiter) - not as powerful of a notation!

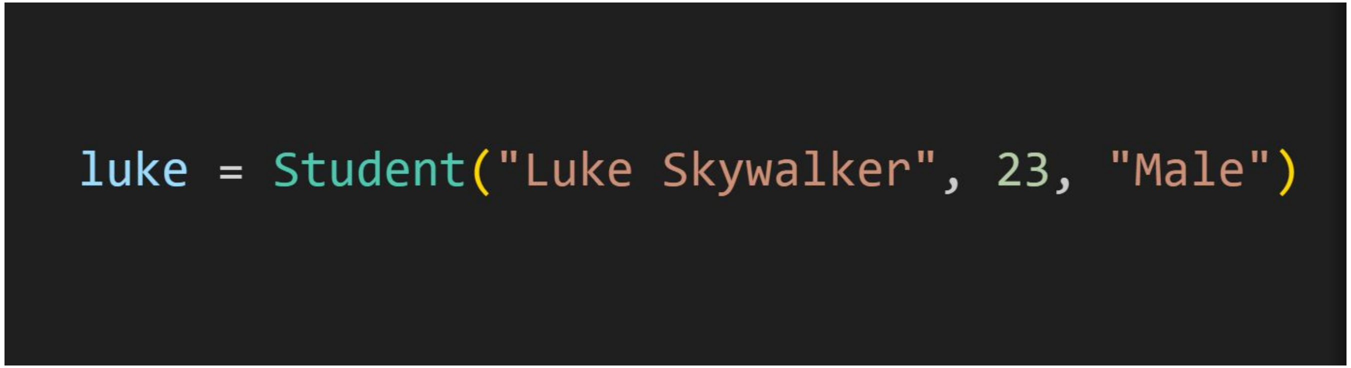
Objects



```
class Student():

    def __init__(self, name, age, graduated):
        self.age = age
        self.name = name
        self.graduated = graduated
```

Class Instantiation

- To instantiate a class we use it’s ame followed by parentheses. Inside the parentheses we add all the required values.



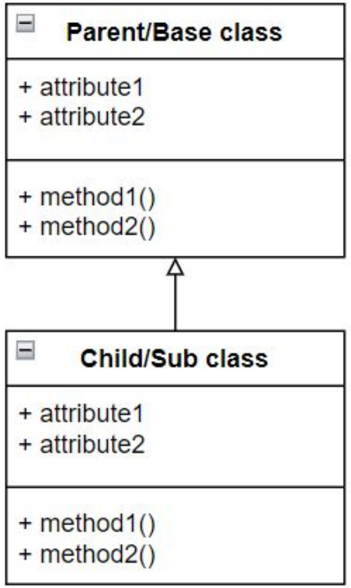
```python
luke = Student("Luke Skywalker", 23, "Male")
```

Hyperiondev

Inheritance

- 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.
- Using inheritance we can create a new class with all the properties and attributes of a base class instead of having to redefine them.

Inheritance



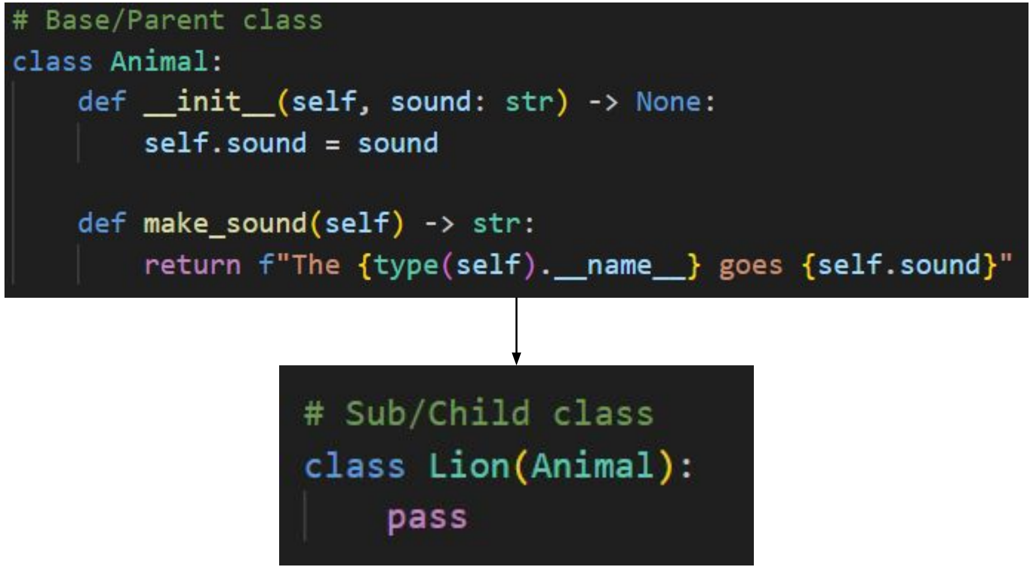
Inheritance

- **Parent/Base class**
  - The parent or base class contains all the attributes and properties we want to inherit.
- **Child/Subclass**
  - The sub class will inherit all of its attributes and properties from the parent class.

Hyperiondev

Inheritance

- Here we have an example of having a child class inherit from a parent class



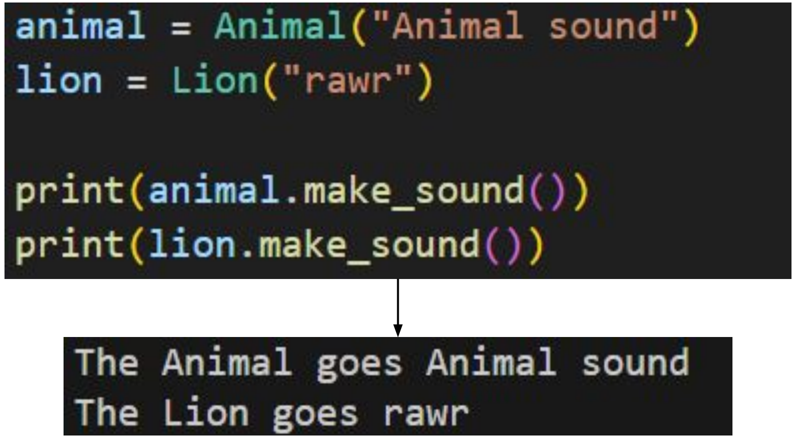
```python
# Base/Parent class
class Animal:
    def __init__(self, sound: str) -> None:
        self.sound = sound

    def make_sound(self) -> str:
        return f"The {type(self).__name__} goes {self.sound}"
```

```python
# Sub/Child class
class Lion(Animal):
    pass
```

Hyperiondev

Inheritance

- Let's create one of each object and see how they behave.



```python
animal = Animal("Animal sound")
lion = Lion("rawr")

print(animal.make_sound())
print(lion.make_sound())
```

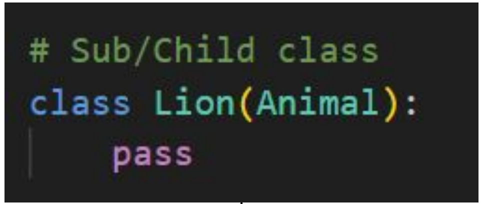
```
The Animal goes Animal sound
The Lion goes rawr
```

Method Overriding

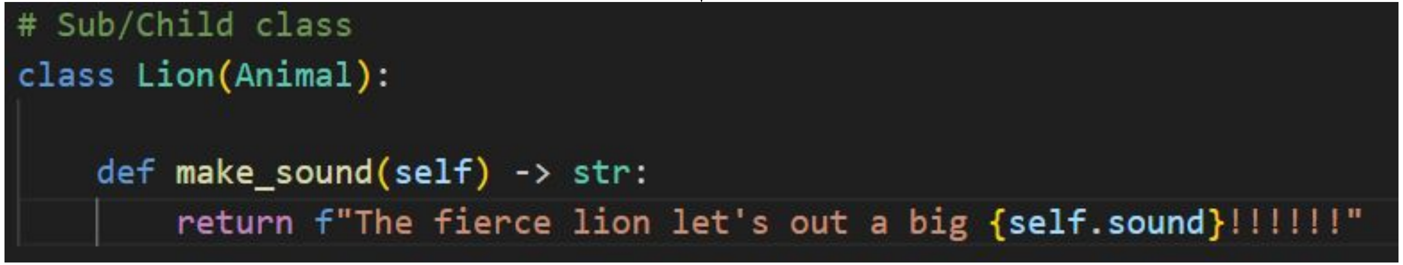
- We can override methods in our subclass to either extend or change the behaviour of a method.
- To apply method overriding you simply need to define a method with the same name as the method you would like to override.
- To extend functionality of a method instead of completely overriding we can use the super() function.

Method Overriding

- Let’s override the make_sound() method.



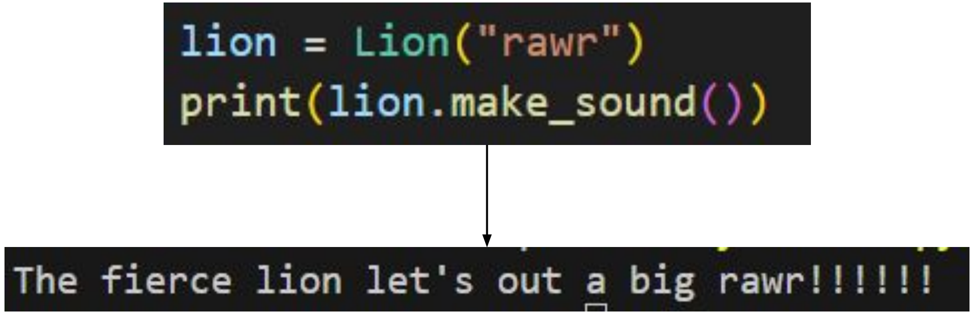
```python
# Sub/Child class
class Lion(Animal):
    pass
```



```python
# Sub/Child class
class Lion(Animal):

    def make_sound(self) -> str:
        return f"The fierce lion let's out a big {self.sound}!!!!!!"
```

Method Overriding

- This will now be the new behaviour of the make_sound() method in the Lion class.



```
lion = Lion("rawr")
print(lion.make_sound())
```

```
The fierce lion let's out a big rawr!!!!!!
```

Hyperiondev

Super()

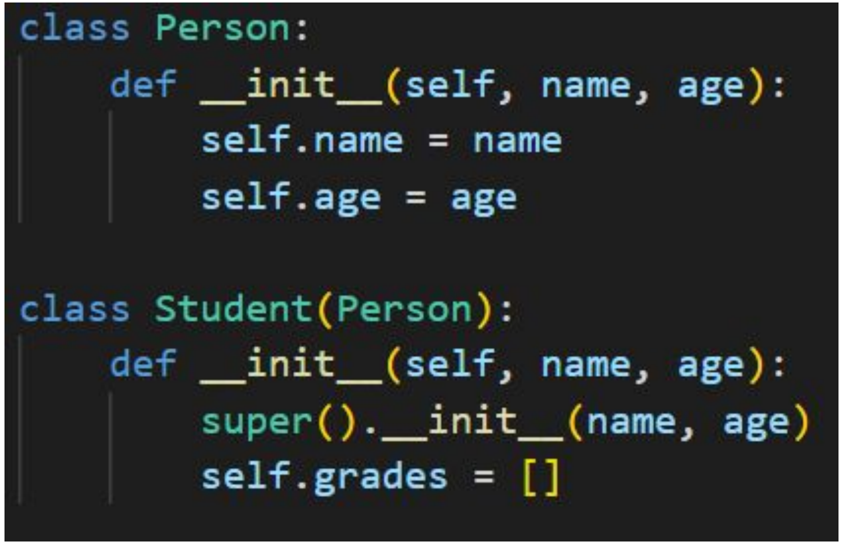
- The super() function allows us to access the attributes and properties of our Parent/Base class.
- Using super() followed by a dot "." we can call to the methods that reside inside our base class.
- When extending functionality of a method we would first want to call the base class method and then add the extended behaviour.

Methods overriding and Super()

- The method you will override the most will be __init__()
- When adding more instance attributes to the subclass we have to call to the base class __init__() to avoid having to redeclare all our base class attributes.
- We can use super().__init__() to call the constructor of the base class and set the values if the inherited attributes.

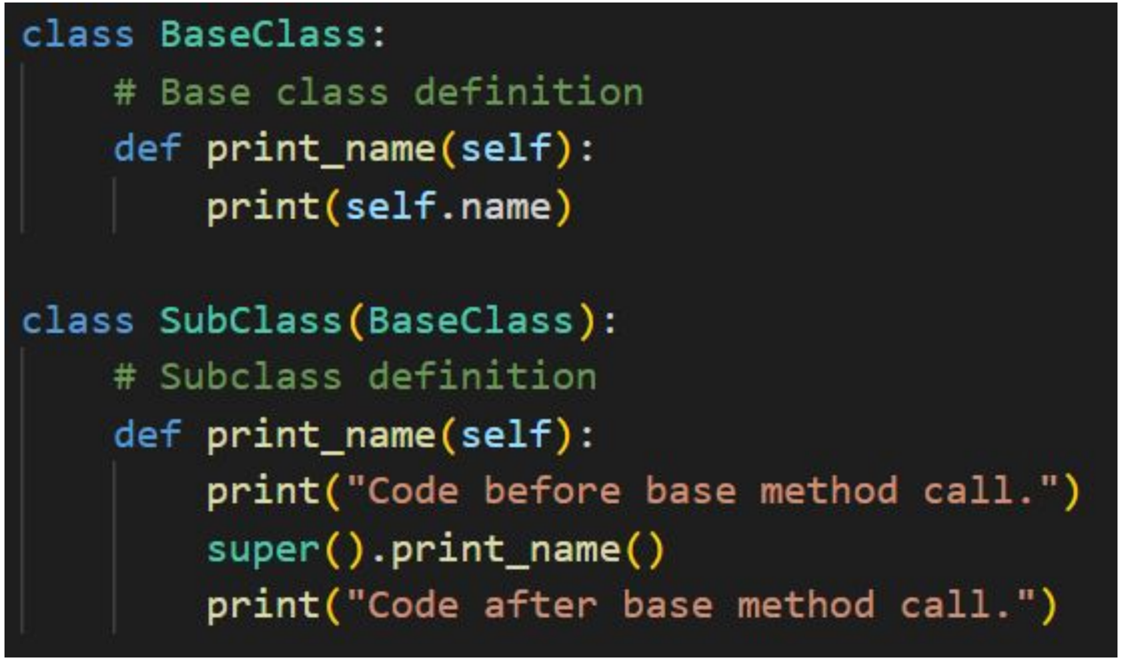
Method Overriding

Here we call __init__() from the Person class to set the values for the attributes “name” and “age”.



```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Student(Person):
    def __init__(self, name, age):
        super().__init__(name, age)
        self.grades = []
```

Method Overriding



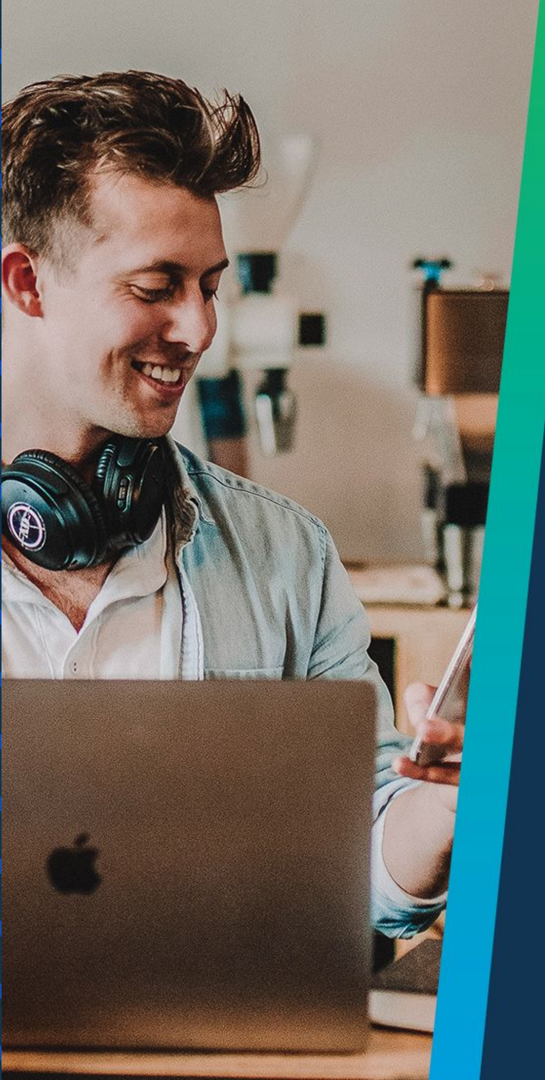
```python
class BaseClass:
    # Base class definition
    def print_name(self):
        print(self.name)

class SubClass(BaseClass):
    # Subclass definition
    def print_name(self):
        print("Code before base method call.")
        super().print_name()
        print("Code after base method call.")
```

Hyperiondev

Hyperiondev

Q & A Section

Please use this time to ask any questions relating to the topic explained, should you have any



Hyperiondev

Thank you for joining us

Take regular breaks.
Stay hydrated.
Avoid prolonged screen time.
Remember to have fun :)

Some useful links

Python Classes: https://docs.python.org/3/tutorial/classes.html

Inheritance:
https://www.digitalocean.com/community/tutorials/understanding-class-inheritance-in-python-3

Polymorphism: https://www.geeksforgeeks.org/polymorphism-in-python/