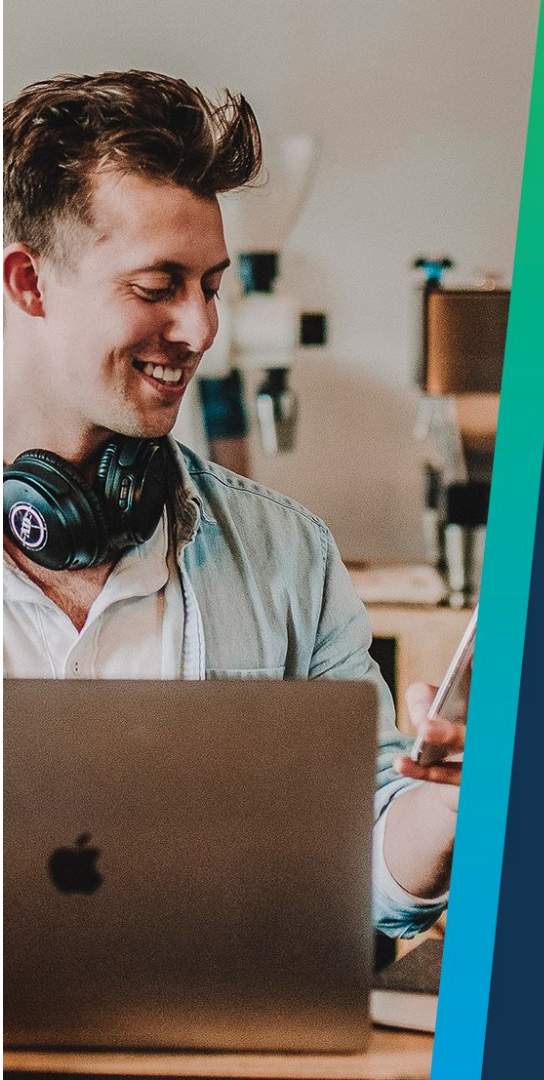


Hyperiondev

# Workshop on Python Object Model

# 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 midway and at the end of the session, should you wish to ask any follow-up questions.
- You can also submit questions here: http://hyperiondev.com/sbc4-se-questions
- For all non-academic questions, please submit a query: www.hyperiondev.com/support
- Report a safeguarding incident: http://hyperiondev.com/safeguardreporting
- We would love your feedback on lectures: https://hyperionde.wufoo.com/forms/zsgv4m40ui4i0g/

# Objectives

1. What is the Python Object Model?
2. Recap on OOP:
    a. Classes & Objects
    b. Attributes & Methods
    c. Inheritance

# Github Repository - Lecture Examples

https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

# What is the Python Object Model?

- Essentially, everything in python is an object.
- Every variable you make is an object, every operation you execute is between objects.
- Seriously. int, float, functions, and anything else you can think of: all objects that inherit from python's base object class, unsurprisingly named **object**.

# What is Object-Oriented Programming?

- A form of programming that models real-world interactions of physical objects.
- Relies on **classes** and **objects** over functions and logic.
- Powerful tool for abstraction.

# OOP Components

- **Class**
  - Different to an object.
  - Think of an object as a house - the class is the blueprint.
- **Properties**
  - Data contained in classes.
  - For example, a student has a name, grade, ID, etc. These are properties of a student.
  - Comes in the form of variables that you can access (e.g. student.name).

# Class Properties

- Most often in Python, this comes in the form of a built-in method.
- These can be accessed using the "." e.g. string.upper() - this calls the upper() method present in the string object.
- FUN/USEFUL FACT: You can actually see all of the properties an object using dir().

Let's explore this a little bit by just looking at an integer.

We'll create an int, then see what attributes it has by calling the dir function.

When dir is called on an object, it recursively searches all the **attributes** of the argument and it's parents or base classes.

```
>>> n = 1
>>> dir(n)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__',
'__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__',
'__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__',
'__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
'__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate',
'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

# Attributes of objects

Some of the attributes listed through dir(n) are class specific to int, and some are inherited from object.

Let's see what attributes the object class has by calling dir on the object class.

```
>>> dir(object)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__']
```

# Creating a Class

- __init__ function is called when class is instantiated.

```
class Student():

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

# Creating an object - Class Instantiation

- Objects are basically initialised versions of your blueprint
- They each have the properties you have defined in your constructor.

```
my_student = Student("Luke Skywalker", 23, "Male")
```

- Class takes in three values: a name, age and gender.

# Creating Methods within a Class

- Within the class, you define a function.
- First parameter is always called self - this references the object itself.
- Let's say you want to average all grades that a student achieved with a single call:
-

```
def average_grades(self):
    return sum(self.grades) / len(self.grades)
```

```
class Student():

    def __init__(self, name, age, gender, grades):
        self.age = age
        self.name = name
        self.gender = gender
        Self.grades = grades

    def average_grades(self):
        average = sum(self.grades)/len(self.grades)
        print(f"The average for student {self.name} is {average}")

my_student = Student("Luke Skywalker", 23, "Male", [80, 75, 91])

# Call the method on the objects
my_student.average_grades()
```

# Class Variables vs. Instance Variables

- Class variable: static, value will never change.
- Instance variable: assigned at instantiation, can change.

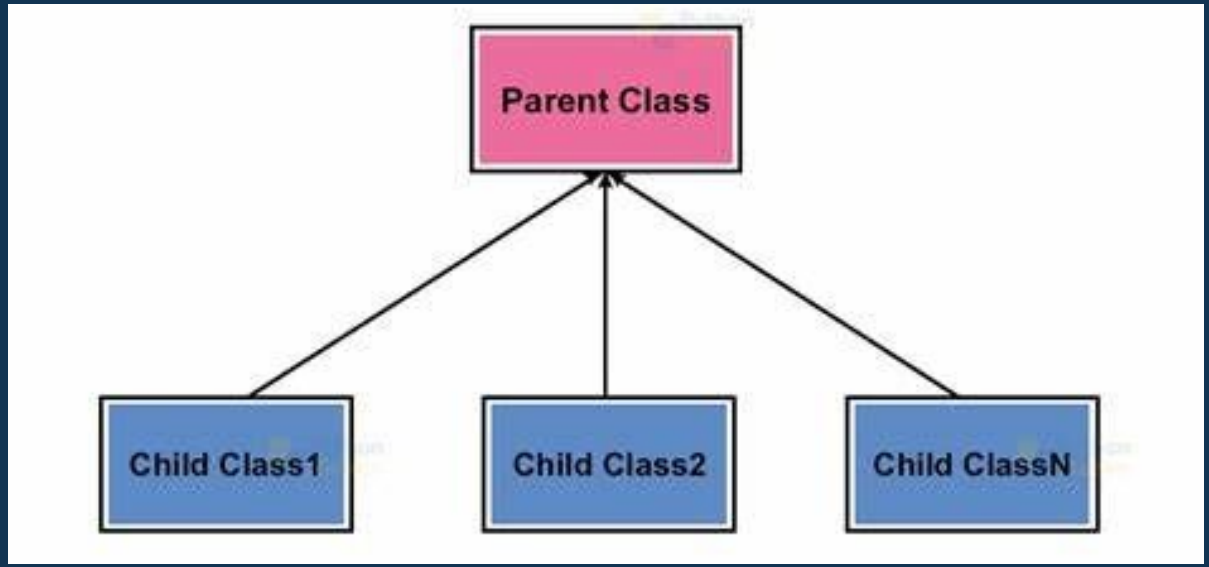
```
Class Student:
    bootcamp = "Software Engineering"
    def __init__(self, name):
        self.name = name

my_se_student = Student("Me")
print(my_se_student.bootcamp) # class variable
print(my_se_student.name) # instance variable
```

# What is Inheritance?

- Inheritance is the ability to define a new class that is a modified version of the existing class.
- The primary advantage of this feature is that you can add new methods to a class without modifying the existing class.
- It is called inheritance because the new class inherits all of the methods of the existing class.
- The existing class is the parent class or base class.
- The new class may be called the child class or subclass

# Parents and Children



# The super() Function

- To access an attribute in the current class, you can use self.
- However, if you need to access an attribute in the parent class, you can use super().

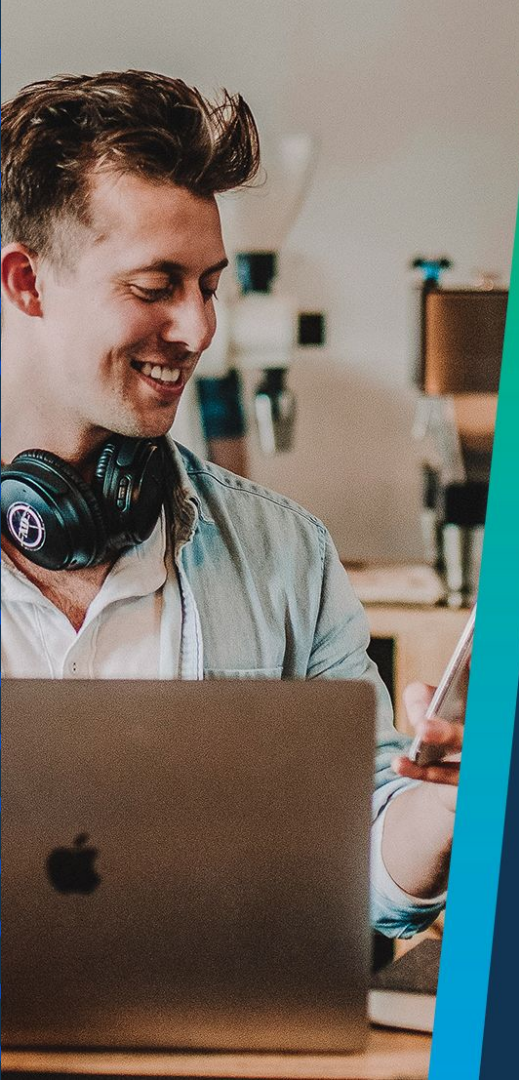
```
# Define parent class
class Parent():
    def __init__(self, name, surname, language):
        self.name = name
        self.surname = surname
        self.language = language

# Define subclass
class Child(Parent):
    def __init__(self, name, surname, language, add_language):
        super().__init__(name, surname, language)
        self.add_language = add_language
```

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