CoGrammar

Tech Talk: OOP

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]()

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.

Why use OOP?

- Imagine that you want to find the average of a student’s grades.

- While the code to find grades, sum them up and average them is easy, it can sometimes look a bit vague.

- It would be nice to simply have a single line of code such as **student.get_average_grades()**.

Classes

- **Class**
  - Blueprint for the class instance
- **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).

Objects in Python

- 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!

Class Methods

- 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()**.

Creating a Class

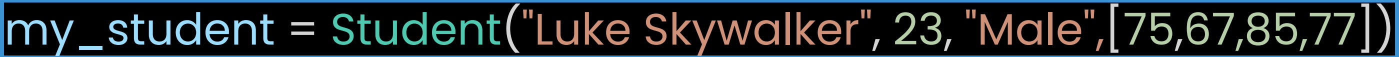
- __init__ function is called when class is instantiated.

```
class Student():

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

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",[75,67,85,77])
```

- Student class takes in four values: a name, age, gender and grades.

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", [75,67,85,77])


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

Class Variables vs. Instance Variables

- Class variable: useful for storing data that is shared among all instances of a class(constants, default values)
- Instance variable: used to store data that is unique to each instance of the class

```
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

Why Inheritance?

- Inheritance is a powerful feature

- Some programs that would be complicated without inheritance can be written concisely and simply with it.

- Also, inheritance can facilitate code reuse, since you can customise the behavior of the parent classes without having to modify them.

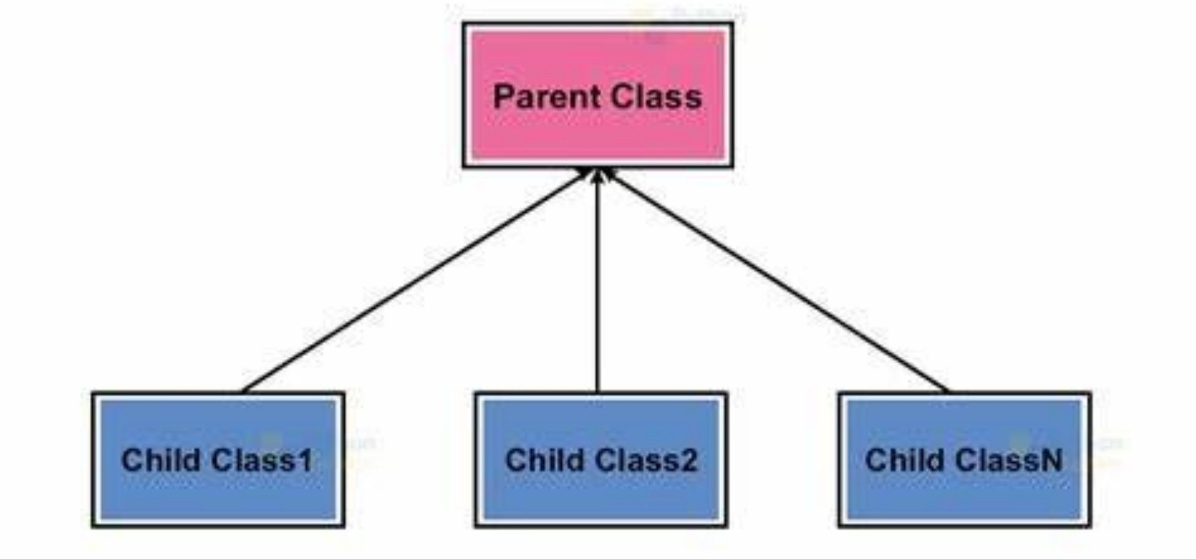
Apples and Fruit

- Is an apple a fruit? Yes.
- Is a fruit an apple? No.
- Let’s think of them as two classes - Fruit and Apple. Let’s also consider other classes like Banana, Mango and Kiwi.
- The Apple, Banana, Mango and Kiwi classes all share similar attributes.
- These attributes can be defined in the Fruit class. Apple, Banana, etc. then all inherit from that class.

Parents and Children

In our Apples and Fruit example, there are some points to note. The Fruit class is considered the **parent** class, and the Apples class is considered the **child** class.

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()**.

Example of super() Function

```
# Define parent class
class Computer():
    def __init__(self, computer, ram, ssd):
        self.computer = computer
        self.ram = ram
        self.ssd = ssd

# Define subclass
class Laptop(Computer):
    def __init__(self, computer, ram, ssd, model):
        super().__init__(computer, ram, ssd)
        self.model = model
```

CoGrammar

Questions

CoGrammar

Thank you for joining

CoGrammar