OOP With Python

Unit 04 - Inheritance

{<oding:lab}

{coding:lab}

Quick Review of Prior Sessions

Check Point 40 (Before we start)

- All students must be able to
  - Create a custom class with
    - Attributes
    - Methods
  - Create an instance of a class
  - Access and update an object's attributes

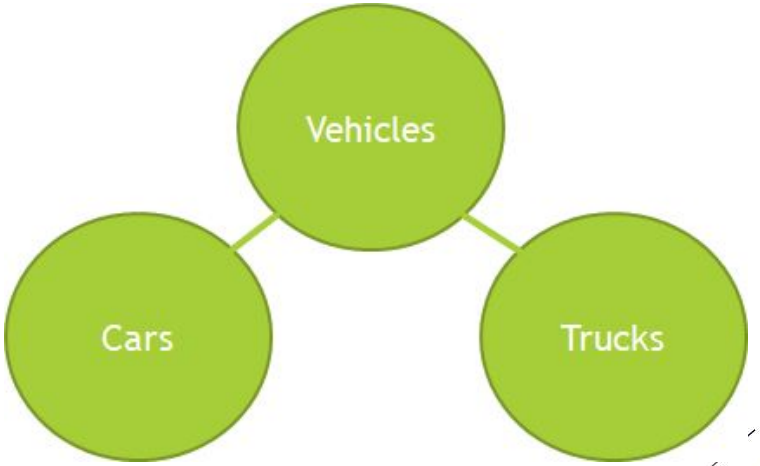
Inheritance

Understanding Inheritance

What is Inheritance in OOP? (1 /2)

- Enables a new object or class to take on (inherit) the properties of another existing class
- For example, trucks and cars are vehicles and thus share similar properties

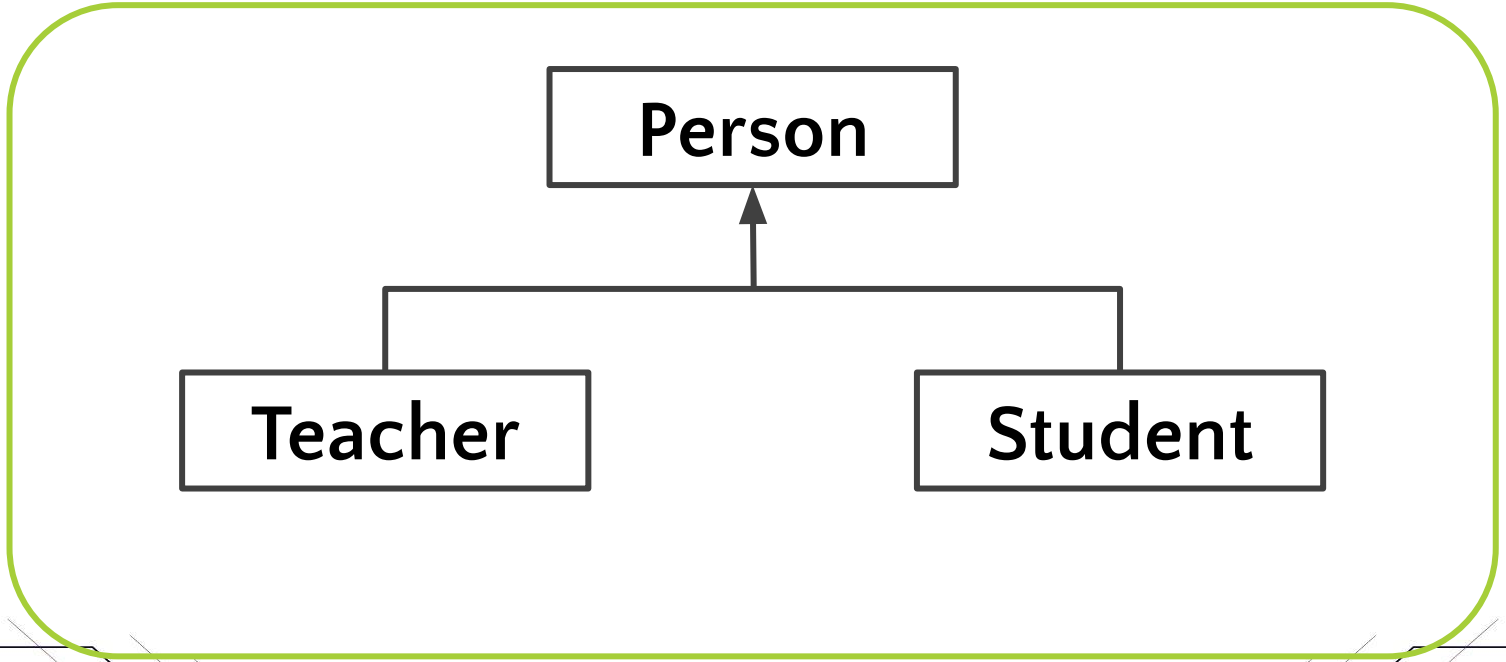
  (such as having engines and wheels)
- We can say cars inherit their properties from vehicles



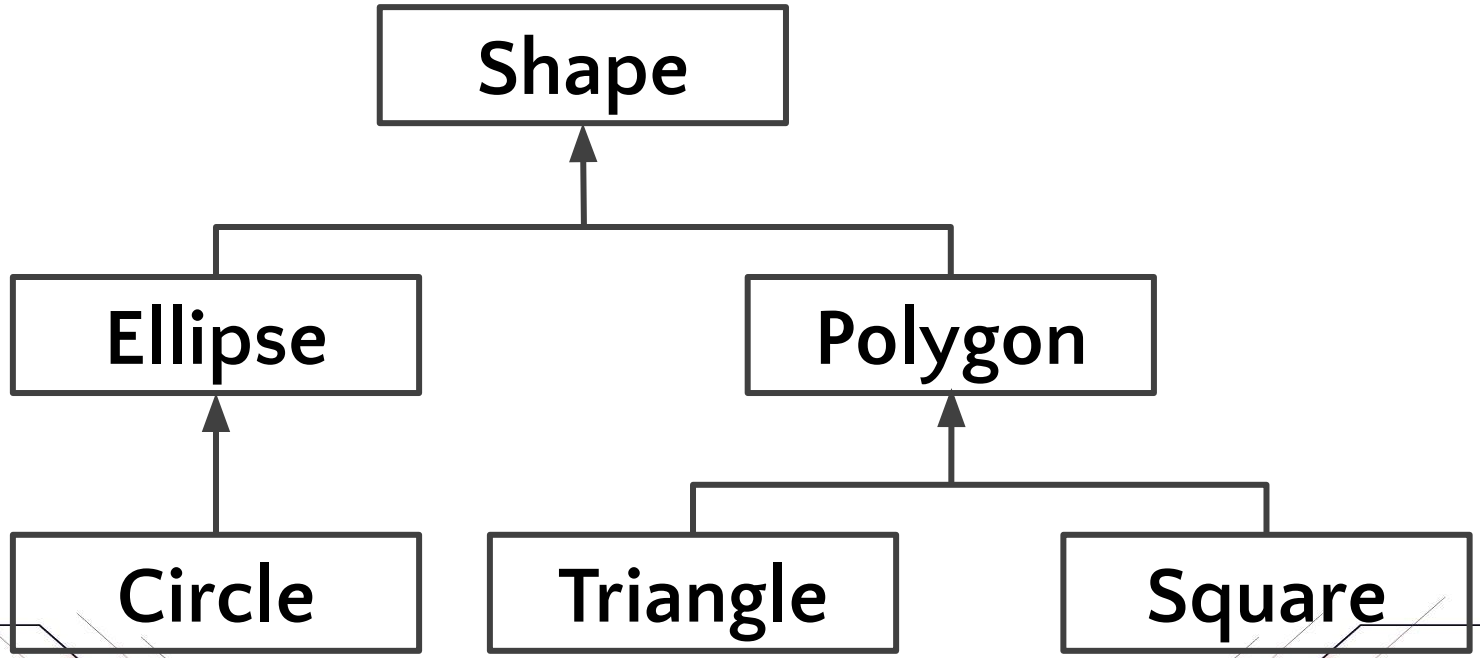
What is Inheritance in OOP? (2 /2)

- The class that other classes inherit from is called a superclass
- The class that inherits from the superclass is call a subclass
- Inheritance models the IS-A relationship. For example
  - Circle IS-A Shape
  - Car IS-A Vehicle

Example of Inheritance (Person)



Example of Inheritance (Shapes)



Why use Inheritance?

- More efficient way of coding
  - Don't have to re-write all the code which is similar
- Easier for maintenance and versioning
- Easier to make future changes

Let's Experiment with Creating a Super Class and a Sub class (Demo/Practice - 41)

- Write a "Person" class which have the following attributes
  - First Name
  - Last Name
  - ID Number
- Write a "Teacher" Class which is a subclass of "Person" with the following additional attributes
  - Teacher ID
  - Department

Defining the Super Class

- Create a class as you normally would

```
1   class Person:
2       def __init__(self, first, last, id):
3           self.firstName = first
4           self.lastName = last
5           self.id = id
6       def getFullName(self):
7           return self.firstName + " " + self.lastName
8       def getFullInfo(self):
9           return self.firstName + " " + self.lastName + ", ID: " + self.id
```

Define the Sub Class which Inherits from the Super Class

- Create a class by doing the following

```
1    # Derived / Sub Class name    : Teacher
2    # Base    /  Super Class name : Person
3    class Teacher(Person):
4        def __init__(self, first, last, id, teacherId, department):
5            # Call the super class initialiser to initialise the attributes
6            Person.__init__(self, first, last, id)
7            # Set the additional attributes of the
8            self.teacherId = teacherId
9            self.department = department
```

Create One Instance of each class

- Create an instance as follow

```
1    #Create an instance, julius of the class, Person
2    julius = Person("Julius", "Ang", "S12345678")
3
4    #Create an instance, teacherSharon of the subclass, Teacher
5    teacherSharon = Teacher("Sharon", "Tze", "S12345679", "CL001", "Math")
```

Call The Method getFullName

- Call the instance method getFullName() and notice how the "Teacher" Class had inherited that method from the "Person" class

```
1    #"Julius Ang" will be printed
2    print(julius.getFullName())
3
4    #"Sharon Tze" will be printed
5    print(teacherSharon.getFullName())
```

Check Point - 41

- Every student must be able to
  - Define a class which inherits from a superclass
  - Create an instance of the subclass
  - Call the method inherited from the superclass
- For students who are waiting, try the following:
  - Experiment with accessing and writing to the object's attributes
  - Create another subclass Student

Overriding an Inherited Method

Subclass can provide a different implementation of a method already defined by its superclass

What is Overriding a Method

- A subclass may override a method it has inherited
- Overriding allows the subclass to implement the method in a way which is more suited for its purpose
- For example, the getFullInfo() method for the “Teacher” class would want to want to print out all the five attributes of a teacher, including the teacher’s ID and department instead of just the person’s name and ID.

Let's Experiment with Overriding (Demo/Practice - 42)

- Override the getFullInfo() method for the "Teacher" subclass to have it return all the teachers attributes:
  - First Name
  - Last Name
  - ID
  - Teacher ID
  - Department

Overriding the Method

- We override the method by defining a method with the same name getFullInfo() in the subclass, Teacher

```
class Teacher(Person):
    def __init__(self, first, last, id, teacherId, department):
        Person.__init__(self, first, last, id)
        self.teacherId = teacherId
        self.department = department

    # Same method name
    def getFullInfo(self):
        # "Sharon Tze, ID: S12345679 CL001 from Math department" will be printed
        # super().getFullInfo() : Calling the super class method. This will ensure that any changes
        # in the super class will be reflected in the sub class
        return super().getFullInfo() + " " + self.teacherId + " from " + self.department + " department
```

Let's Experiment with Overriding (Check Point - 42)

- Every student must be able to
  - Override a method in the super class
- For students who are waiting, try the following:
  - Go on and create a simple school database on the next slide

A Simple School Database

- Create a simple school database system which allows you to maintain the records of the teachers and students in a school
  - Make use of persistent storage
  - Allow administrator to add, update, display and remove teachers and students

- Additional Challenges
  - Define a Class which holds the list of the records for a school

Reflections

What is Inheritance and why use it?

- Enables a new object or class to take on (inherit) the properties of another existing class
- Enables more efficient coding

Questions to Ask Yourself

- Why do we use object oriented programming?
  - Code reusability
  - Clear modular structure
  - Easy to maintain and modify existing codes
- Why do we utilise inheritance in OO?
  - More efficient coding
  - Easy to maintain and modify existing codes