Inheritance

MAD 102

Inheritance

- Inheritance is a process where the characteristics of a class – the attributes and methods – are inherited by another class
  - This reduces the amount of new code required when a new program is developed
  - It allows for updates to be passed seamlessly to other objects, ensuring updates are automatically handled

Class Hierarchy

- Inheritance creates a hierarchical relationship a parent-child relationship
  - The class that provides the inheritance is the parent
  - The class adopting the inherited traits is the child

Class Inheritance terms

- A class that inherits the attributes of another class is called a derived class
- The class that provides the attributes is the base class
- To indicate that a class is a subclass (**derived class**) – place the parent class (**base class**) name in parentheses after the class declaration

Class inheritance

```python
class Vehicle:
    def __init__(self, current_speed=0):
        self.current_speed = current_speed

    2 usages
    def description(self):
        print(f'Travelling at {self.current_speed} km/h')


1 usage
class Bicycle(Vehicle):
    def __init__(self, has_basket=False):
        Vehicle.__init__(self)
        self.has_basket = has_basket


car = Vehicle()
car.description()

bike = Bicycle()
bike.description()
```

```
Travelling at 0 km/h
Travelling at 0 km/h
```

Note how the Bicycle (derived) does not contain a description function
It inherits this functionality from the Vehicle class (base)

Overriding class methods

- A derived class may decide to have the same behaviour as its parent, but with some modifications
- This is called an overriding class method
  - It overrides (replaces) the implementation of its parent class

Overriding class methods

```python
class Vehicle:
    def __init__(self, current_speed=0):
        self.current_speed = current_speed

    2 usages
    def description(self):
        print(f'Travelling at {self.current_speed} km/h')

    1 usage
    def make_noise(self):
        print("Vroom! Vroom!")


1 usage
class Bicycle(Vehicle):
    def __init__(self, has_basket=False):
        Vehicle.__init__(self)
        self.has_basket = has_basket

    1 usage
    def make_noise(self):
        print("Honk! Honk!")


car = Vehicle()
car.description()
car.make_noise()

bike = Bicycle()
bike.description()
bike.make_noise()
```

```
Travelling at 0 km/h
Vroom! Vroom!
Travelling at 0 km/h
Honk! Honk!
```

Overriding class methods

- Your methods do not have to completely replace – they may want to make only slight changes
  - They may want to extend the functionality of the parent
  - Call the base method at the beginning of your method
  - Follow with the extension

Overriding class methods

```python
class Vehicle:
    def __init__(self, current_speed=0):
        self.current_speed = current_speed

    2 usages
    def description(self):
        print(f'Travelling at {self.current_speed} km/h')

    1 usage
    def make_noise(self):
        print("Vroom! Vroom!")


1 usage
class Bicycle(Vehicle):
    def __init__(self, has_basket=False):
        Vehicle.__init__(self)
        self.has_basket = has_basket

    1 usage
    def description(self):
        Vehicle.description(self)
        print('But in the bike lane!')

    1 usage
    def make_noise(self):
        print("Honk! Honk!")
```

```python
car = Vehicle()
car.description()
car.make_noise()

bike = Bicycle()
bike.description()
bike.make_noise()
```

```
Travelling at 0 km/h
Vroom! Vroom!
Travelling at 0 km/h
But in the bike lane!
Honk! Honk!
```

Multiple Inheritance

- A class can inherit from more than one base class
  - This is known as multiple inheritance
  - The inherited classes are added separated by a comma

Mixins

- Mixins extend the functionality of a class by mixing in additional methods
  - These are classes
  - These are not intended to be instantiated – just add addition things that your classes can do

Mixins

```python
class StuntMixin:
    def jump(self, distance):
        print(f'You have jumped {distance} metres!')

    def skid(self, distance):
        print(f'You slammed on the brakes and left a skid mark {distance} metres long!')

class CarefulMixin:
    def signal(self, direction):
        print(f'You are turning {direction}, and put on the turn signal')


class DirtBikes(Bicycle, StuntMixin):
    def __init__(self, has_basket=False):
        super().__init__(has_basket)

    def make_noise(self):
        print('Vroom! Vroom! Vrooooom!')
```

```python
duke_caboom = DirtBikes()
duke_caboom.jump(10)
duke_caboom.make_noise()
duke_caboom.current_speed = 100
duke_caboom.skid(2)
duke_caboom.description()
```

```
You have jumped 10 metres!
Vroom! Vroom! Vrooooom!
You slammed on the brakes and left a skid mark 2 metres long!
Travelling at 100 km/h
```

Is-a vs has-a

- Object's relationships can be described as an is-a or a has-a
- This tells us how one object is related to another.
- If one object is-a more defined version of another, this means it is derived
  - This indicates an **inheritance** relationship
- If one object has-a relationship with another, this is what is described as **composition**

Composition

- Composition is the concept of one object being made up of other objects
  - For example – books and authors
  - Both can be defined as objects – but they can also be composed of each other

```
class Book:
    def __init__(self, author, genre, pages):
        self.author = author
        self.genre = genre
        self.pages = pages


class Author:
    def __init__(self, num_books_written) :
        self.num_books_written = num_books_written
```

A book **has-a**n author

Inheritance

- An author is a person – a teacher is a person. They both share some attributes (name, age)
- But they have distinct details that differentiate them from each other – details that the other one does not have

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

class Author(Person):
    def __init__(self, num_books_written) :
        Person.__init__(self)
        self.num_books_written = num_books_written
```

Author **is-a** person