CLASSES AND OBJECTS

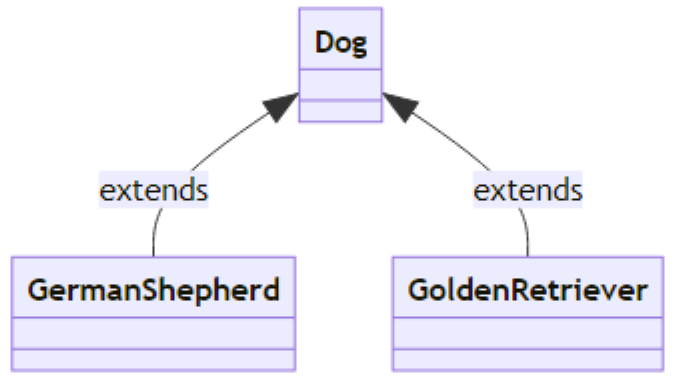
DR IAN CORNELIUS

HELLO

- Learning Objectives
    1. Understand what a class and object is
    2. Demonstrate the ability to use classes/objects

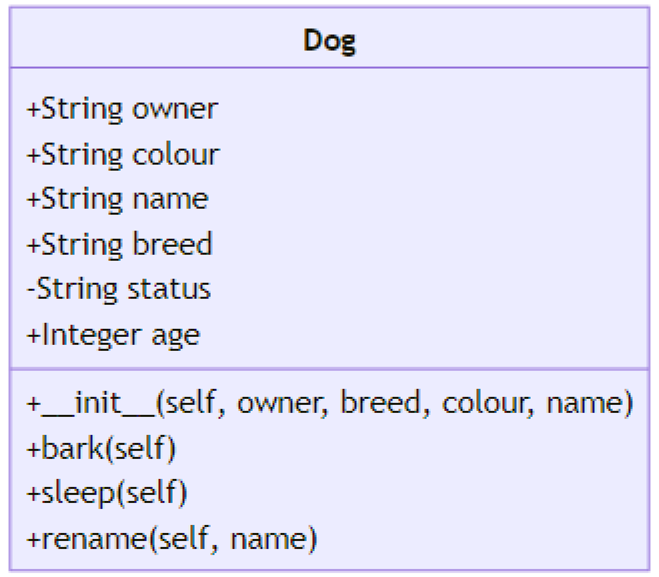
INTRODUCTION TO CLASSES

- Classes provide a structure for the objects
- They are used for defining:
  - a set of properties, represented by variables
  - the behaviour, which are represented by functions
- Objects created from classes will be referred as instance(s)
  - the process of creating an object from a class is instantiation
- Objects will have a property
  - a set of values that are associated to a real-world entity
- For example:
  - Class: Dog
  - Objects: German Shepherd, Golden Retriever



CLASS EXAMPLE

- This is a class diagram, it shows the variables and functions of a class
- + and - symbols declare whether a variable or function is **public** or **private**
- Top half denotes all the variables of a class
- Bottom half denotes all the methods/functions of a class



CLASSES IN PYTHON (1)

CREATING A CLASS

- Classes will be defined using the `class` keyword followed by the name you want to give it

```
class Student:
    name = "Ian"
```

CREATING AN OBJECT

- An object can be created from the class by calling upon the class name
  - we first create a variable for this class object
  - then we call the class name followed by brackets

```
student1 = Student()
```

- The variable name can then be accessed
  - by calling the variable name at the end of the object

```
student1.name = Ian
```

USING CLASS CONSTRUCTORS

- All classes consist of an in-built function which is used to execute code when it is being initiated
    - this is the function known as `__init__()`
- This initialiser can be used to assign values to an objects properties,
    - or other operations that are necessary to perform when an object is in the process of being created
- `__init__()` is called automatically each time the class has been used to create a new object

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

```
student1 = Student("Ian", 33)
student2 = Student("Terry", 1)
```

```
student1.name = Ian
student1.age = 33
student2.name = Terry
student2.age = 1
```

CLASS FUNCTIONS

- Classes can also consist of functions, and these will belong to the object that is created

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greeting(self):
        return "Hello " + self.name + " and welcome to 4061CEM!"
```

```
student1 = Student("Ian", 33)
```

```
student1.greeting() = Hello Ian and welcome to 4061CEM!
```

MODIFYING AN OBJECTS PROPERTIES (1)

- Objects can be modified by accessing the variables directly
    - this is not a very good method of doing this

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

```
student1 = Student("Ian", 33)
student1.age = -1
```

```
[Before] student1.age = 33
[After] student1.age = -1
```

MODIFYING AN OBJECTS PROPERTIES (2)

- A better way of modifying a classes variable is by creating a function

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def change_age(self, new_age):
        self.age = new_age
```

```
student1 = Student("Ian", 33)
student1.change_age(-1)
```

```
[Before] student1.age = 33
[After] student1.age = -1
```

MODIFYING AN OBJECTS PROPERTIES (3)

- Variables and functions can be private inside a class
- This is achieved by adding two underscores ('__') to the beginning of the variable name

```
class Student:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age
    def change_age(self, new_age):
        self.__age = new_age
    def get_age(self):
        return self.__age
```

```
student1 = Student("Ian", 33)
student1.change_age(-1)
```

```
[Before] student1.get_age() = 33
[After] student1.get_age() = -1
```

MODIFYING AN OBJECTS PROPERTIES (4)

- Objects can be deleted by using the `del` keyword

```
del student1
```

PASS KEYWORD

- Classes cannot be empty, but if you require a class to be empty you can use the `pass` keyword

```
class Student:
    pass
```

INHERITANCE WITH CLASSES

- Inheritance allows you to define a class that will inherit all the functions and properties from another class
- There are two important terminologies to know:
    - **parent class** which is the class from which another class is being inherited from
    - **child class** which is the class that is inherited from the parent class

CREATING A PARENT CLASS

- Any class you create in Python can be a parent class
    - the syntax is the same as creating any other class

```
class Student:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age
    def change_age(self, new_age):
        self.__age = new_age
    def get_age(self):
        return self.__age
    def get_name(self):
        return self.__name
    def greeting(self):
        return f"Hello {self.__name} and welcome to 4061CEM!"
```

```
student1 = Student("Ian", 33)
```

```
student1.get_name() = Ian
student1.get_age() = 33
student1.greeting() = Hello Ian and welcome to 4061CEM!
```

CREATING A CHILD CLASS

- A child class is created by passing through the parent class as parameter when creating the child class
  - the child class will inherit all properties and functions of the parent class

```
class Person(Student):
    pass
```

```
person1 = Person("Ian", 33)
```

```
person1.get_name() = Ian
person1.greeting() = Hello Ian and welcome to 4061CEM!
```

MODIFYING A CHILD CLASS (1)

- When adding an `__init__` function to the child class, it will no longer inherit the `__init__` function from the parent class
- To keep the variables from the parent class, we need to call the `__init__` function from the parent class

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

```
person1 = Person("Ian", 33, "Coventry")
```

```
person1.get_name() = Ian
person1.location = Coventry
```

MODIFYING A CHILD CLASS (2)

- The same process can be achieved using the `super()` function, instead of using the parent classes name

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

```
person1 = Person("Ian", 33, "Coventry")
```

```
person1.get_name() = Ian
person1.location = Coventry
```

ADDING A FUNCTION TO A CHILD CLASS

- Functions can be added to the child class in the same manner
  - let’s add a function to greet the person and acknowledge where they have come from

```
class Person(Student):
    def __init__(self, name, age, location):
        super().__init__(name, age)
        self.location = location
    def greeting(self):
        return f"Hello {self.get_name()} and welcome to 4061CEM from {self.location}!"
```

```
person1 = Person("Ian", 33, "Coventry")
```

```
person1.greeting() = Hello Ian and welcome to 4061CEM from Coventry!
```

GOODBYE

- Questions?
    - Post them in the **Community Page** on Aula
- Contact Details:
    - Dr Ian Cornelius, ab6459@coventry.ac.uk