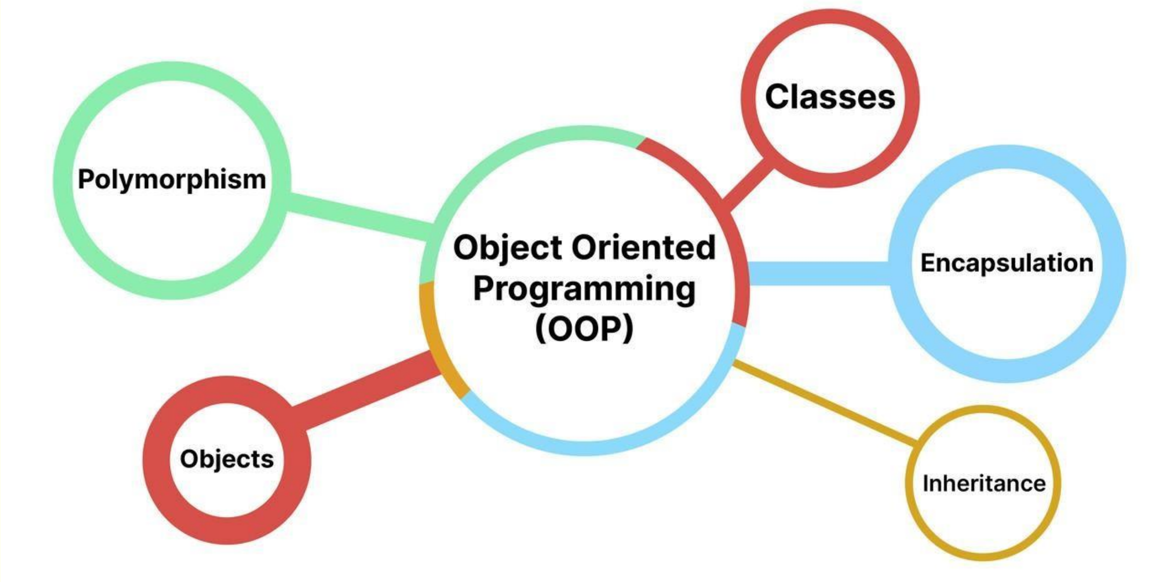
Learning Aims

- Understand OOP – Compare OOP with procedural programming.
- Use **Classes & Objects** – Define, create, and instantiate objects.
- Apply OOP Principles – **Encapsulation, Inheritance, Polymorphism, and Abstraction**.
- Work with **Methods & Attributes** (Constructor method/ getter and setter methods to access and modify attributes.)
- Control Access – Apply **public, private and protected** access
- Develop OOP programs using OCR pseudocode and python



Key words

Keyword	Definition
Class	A blueprint for creating objects that defines attributes (data) and methods (behaviour).
Object	An instance of a class with specific values for its attributes.
Encapsulation	Hiding an object's data and only allowing controlled access through methods (getters and setters).
Inheritance	A child class inherits attributes and methods from a parent class, promoting code reuse.
Polymorphism	The ability for methods to take multiple forms, either by method overloading or method overriding. Method Overriding - When a child class redefines a method inherited from the parent class to change its behaviour. Method Overloading- When multiple methods in the same class have the same name but different parameters
Abstraction	Hiding complex implementation details and exposing only essential features to the user.
Constructor	A special method that is automatically called when an object is created, used to initialise attributes.
Getter Method	A method that retrieves (gets) the value of an attribute.
Setter Method	A method that updates (sets) the value of an attribute.
Public	Allows attributes and methods to be accessed from anywhere.
Private	Restricts access to attributes and methods within the same class.
Protected	Allows access within the class and its subclasses.

Key Terminology

Class

Model/Blueprint

Attributes

variables

Methods

Behaviours

(Functions or Procedures)

Instantiation the process of creating an object from a class

Instance

each object is called an instance

Constructor method

Defines the object when it is first created

public and _private attributes/methods

Doc_string “””   “”” – used to describe the class for maintainability

Recap Procedural Programming

- Program code is divided up into **subroutines**, which are discrete blocks of code that carry out a single task.
- A **top-down approach** is used to develop a program in a structured way by breaking down a program into sub-routines and into smaller parts (stepwise refinement)
- Each subroutine contain a series of computational steps to be carried out in the order specified by the programmer.
- To run this program, our main program is simply a series of calls to the different functions.
- Global and local variables be used along with parameters as standard programming techniques
- **Parameter passing** is used - passing by reference or passing by value

```
# Function to feed the pet (reduces hunger)
function feed_pet(name, hunger)
   hunger -= 10  # Reduce hunger
   print(f"{name} has been fed! Hunger: {hunger}")
   return hunger  # Return updated hunger value

# Main program
procedure main()
   name = "Fluffy"
   hunger = 50  # Initial hunger level

   while True
      print(”1: Feed Pet”
      print(”2: Exit”)
      choice = input("Choose an option: ")

      if choice == "1":
         hunger = feed_pet(name, hunger)  # Pass and return hunger value
      elif choice == "2":
         print("Goodbye!")
         break
      else:
         print("Invalid choice, try again.")

# Run the program
main()
```

Recap Procedural programming benefits and drawbacks

Benefits

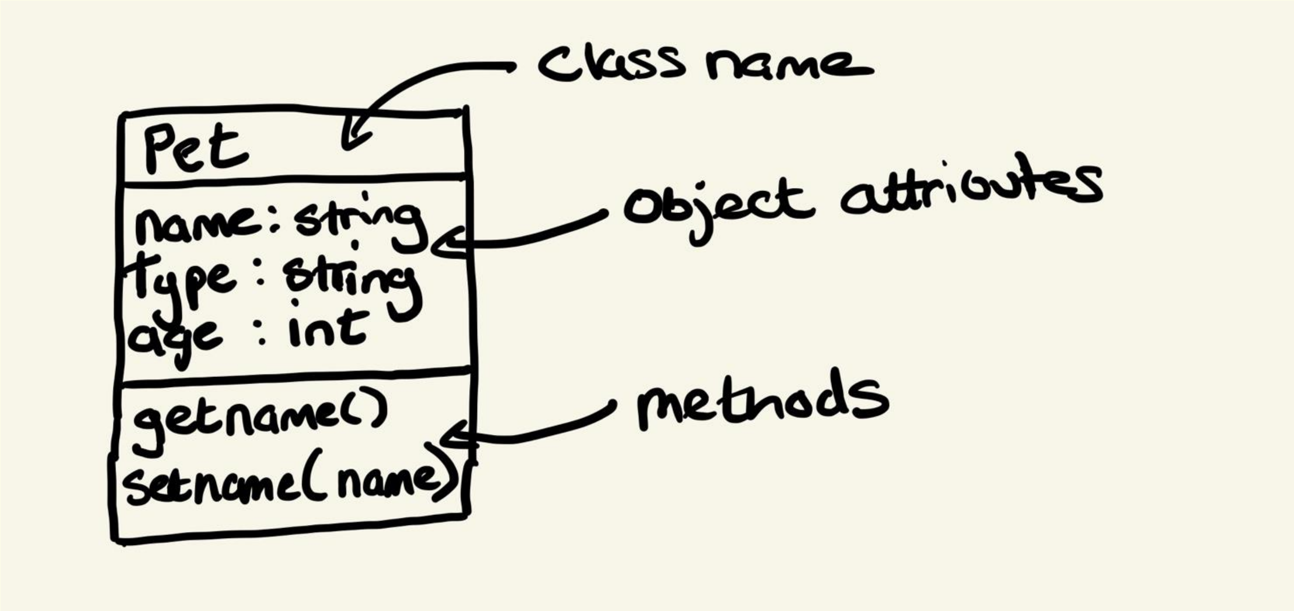
- Splitting code into smaller chunks has many benefits:
- It is much easier to test and debug, e.g. tracing 20 lines of code instead of 200 lines of code.
- Subroutines can be called many times, reducing the amount of repeated code.
- Subroutines can manipulate shared data.
- Subroutines can be saved in libraries and imported into programs when they are needed. This means that code is reusable; it's good if you can write code once and make use of it many times.
- Very large programs can be worked on by a whole team of programmers who can develop subroutines concurrently (at the same time) so that a project can be completed faster.

Drawbacks

- Overuse of global variables.
- The problem with global variables is that, since they can be accessed and modified by every subroutine in a program, it is really hard to work out where a value is changed.
- If you are trying to debug a program, you may have to consider every subroutine that modifies the global state. That is possible, but with a really large application, it is very difficult to do.
- Global variables should always be used very carefully.
- Indeed, there is nothing in the procedural paradigm that truly **protects data.**
- Data is passed into subroutines, used, and sometimes modified, and the programmer must be fully aware of what they should and shouldn't do, and abide by the rules of good practice to ensure **data integrity.**

Object-oriented programming

- A class is a **template/ blueprint** for creating objects
- A class defines an object’s **attributes** (data) and **behaviours** (methods)



The pet class is a blueprint for representing all pets, in other words an abstract model of a pet.

Constructor Method

```
class Pet()
   #define the attributes as private
   private petName
   private petType
   private petAge
   #Constructor method
   public procedure new(pName, pType, pAge)
       petName = pName
       petType = pType
       petAge = pAge
   end procedure
end class

########main program ###########
# pet object - an instance of a class
myPet = new Pet(“Buster”,”dog”,5)
```

This constructor defines the object’s attributes when it is first **instantiated** in the main program

myPet object

Buster
Dog
5

Accessing and modifying attributes

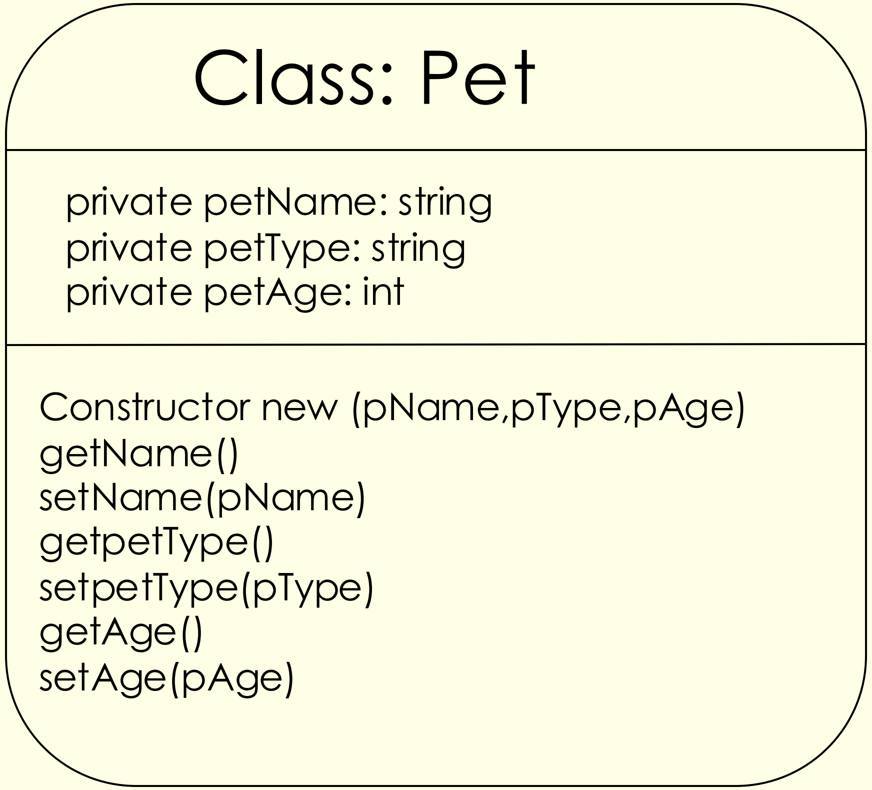
- Attributes should **only** be accessed or modified using **getter and setter methods**

- **This is a form of Encapsulation (Information hiding)**

- Data should never be accessed directly.

Example:

```
#Getter method used to return the value of the attribute
public function getName()
    return petName
end function

# Setter method used to modify the value of an attribute
public procedure setName( pName)
    petName = pName
end procedure
```



Class: Pet

private petName: string
private petType: string
private petAge: int

Constructor new (pName,pType,pAge)
getName()
setName(pName)
getpetType()
setpetType(pType)
getAge()
setAge(pAge)

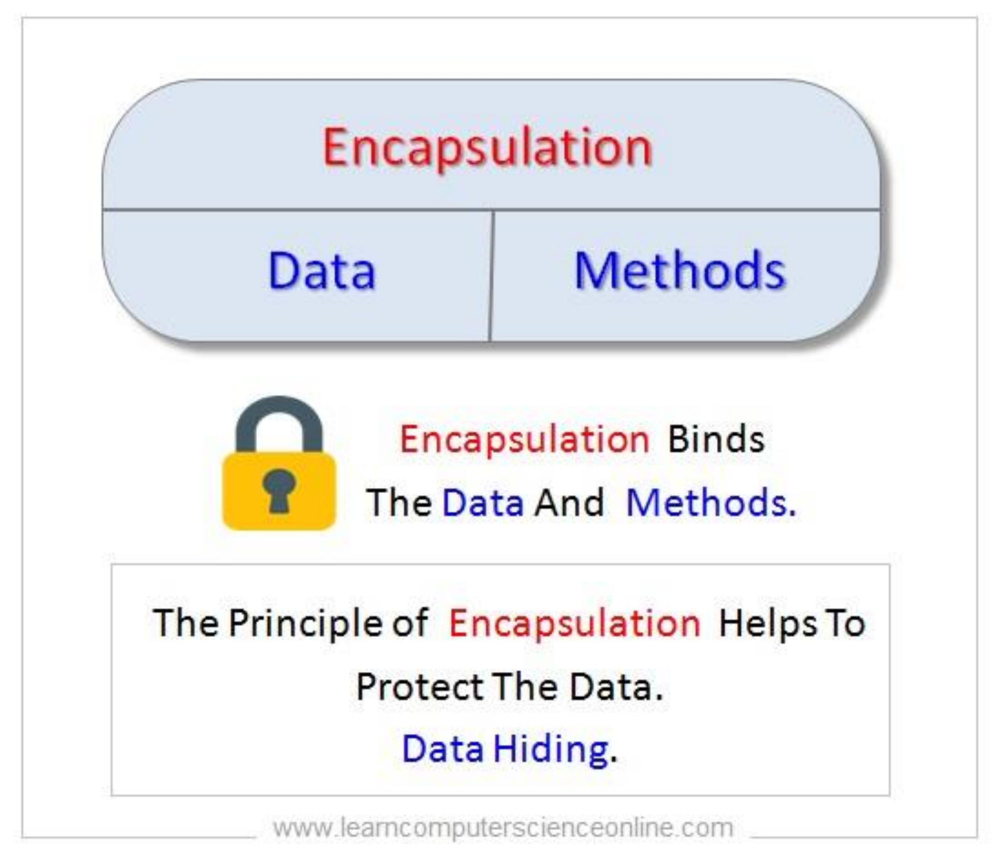
Encapsulation (Information Hiding)

- In OOP Encapsulation refers to the concept of **bundling** an object's **data** (attributes) and the **methods** (functions) that operate on that data into a **single unit**, called a **class**.

- It also means **restricting access** to the internal workings (data) of the object, allowing interaction only through well-defined **public methods**.

- **Data Hiding**: The internal data (like variables) of an object is **hidden** from the outside world and can only be accessed or modified via **getter** and **setter** methods.

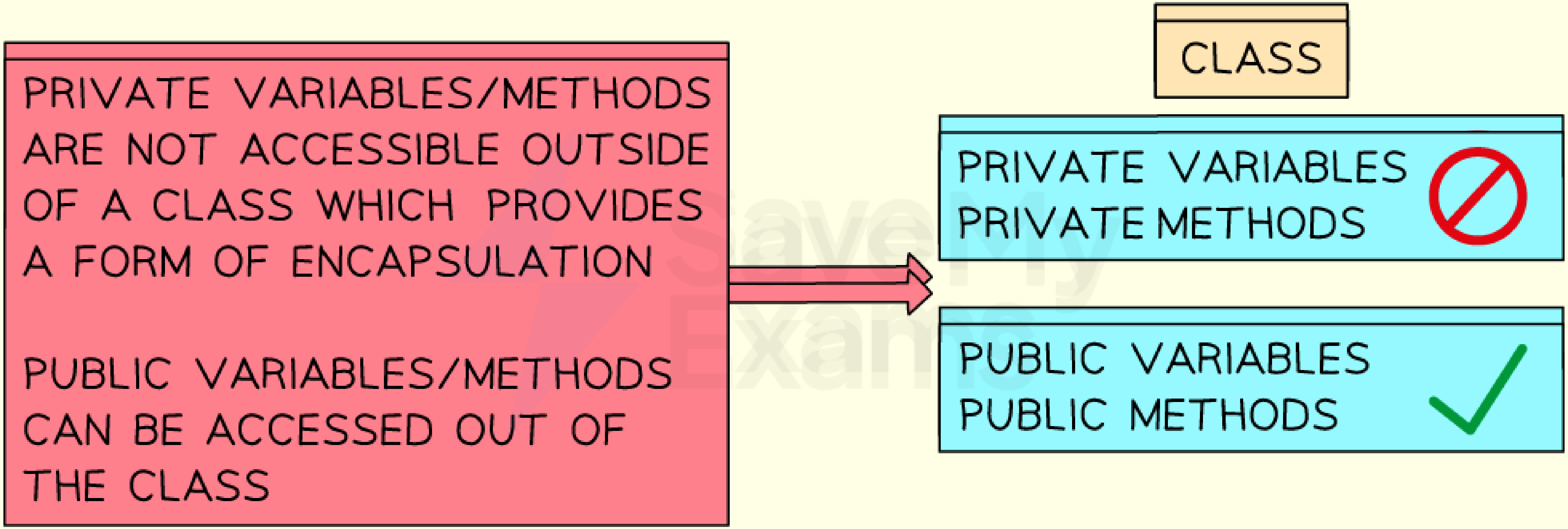
- **Controlled Access**: By providing controlled access to the data, encapsulation prevents unwanted interference or accidental changes, making the program safer and easier to maintain.


Encapsulation
Data
Methods
Encapsulation Binds The Data And Methods.
The Principle of Encapsulation Helps To Protect The Data.
Data Hiding.
www.learncomputerscienceonline.com

Private and Public

Private attributes and methods can only be accessed within the same class.

public methods can be accessed by other classes and in the main program.



PRIVATE VARIABLES/METHODS ARE NOT ACCESSIBLE OUTSIDE OF A CLASS WHICH PROVIDES A FORM OF ENCAPSULATION

PUBLIC VARIABLES/METHODS CAN BE ACCESSED OUT OF THE CLASS

Encapsulation is making attributes private in a class but allowing them to be changed and accessed through the public methods

Private attributes & Public methods

- **Private attributes** Properties are encapsulated and can only be accessed through their methods and within the class only (not directly accessed and changed in the main program)
- Enforce validation through the method // inappropriate data can be caught before entered
- Cannot be changed/accessed accidentally

Main program

```
plane = new Airplane()
plane.setWeight = (500)
```

```
Class Airplane
        private weight
        private fuel
        private passengers
```

You can not access/change the attributes directly and would cause an error:
Main program

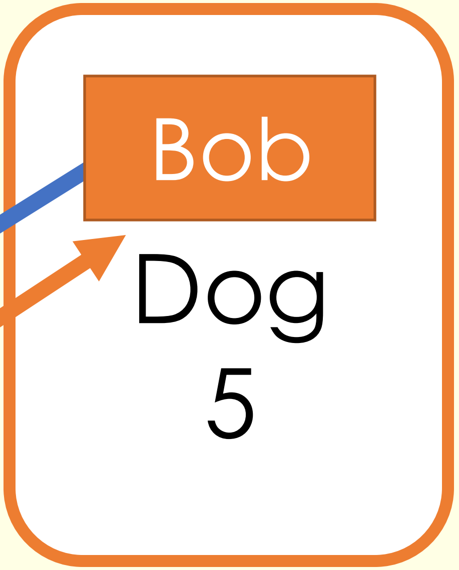
```
plane = new Airplane()
plane.weight = 9999
```

```
public procedure setWeight(enteredWeight)

        if enteredWeight > maxWeight then
                print("Too Heavy")
        else
                weight=enteredWeight
                updateFuel()
        endif

public function getWeight()
        return weight
End function
```

Public Methods so they can be accessed by other classes and outside the class in the main program

```
class Pet()
   #Constructor
   public procedure new(pName, pType, pAge)
       petname = pName
       petType = pType
       petAge = pAge
   end procedure

   #Getter method returns the value of an attribute
   public function getName(self)
      return self.name
   end function
   # Setter methods modifies the value of an attribute
   public procedure setName(self, name)
      self.name = name
   end procedure
```

```
#main program
# pet object
myPet = new Pet(“Buster”,”dog”,5)
#Access an attribute
print(myPet.getName())
#Change an attribute
myPet.setName(“Bob”)
```

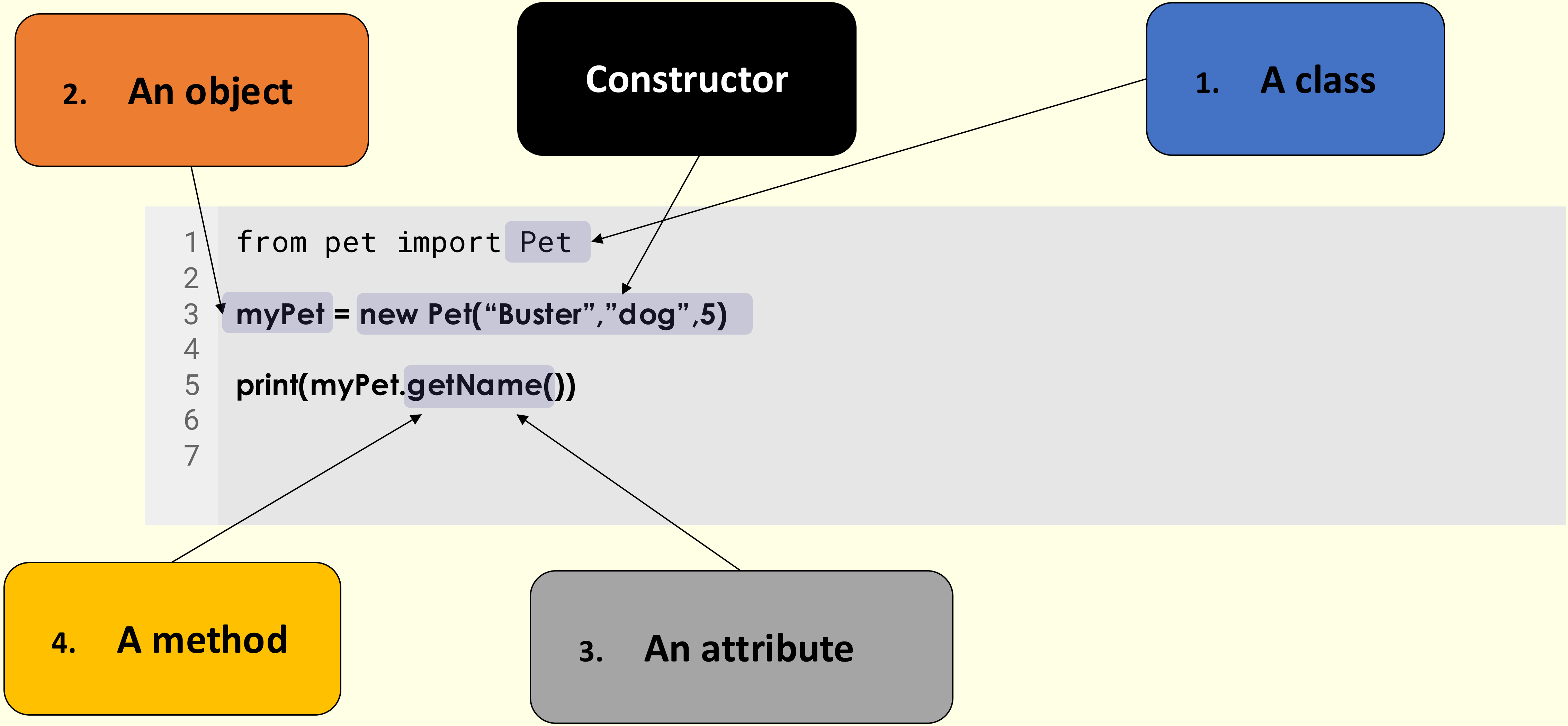


Bob

Dog

5

Python



2. An object

Constructor

1. A class

```
from pet import Pet

myPet = new Pet(“Buster”,”dog”,5)

print(myPet.getName())
```

4. A method

3. An attribute

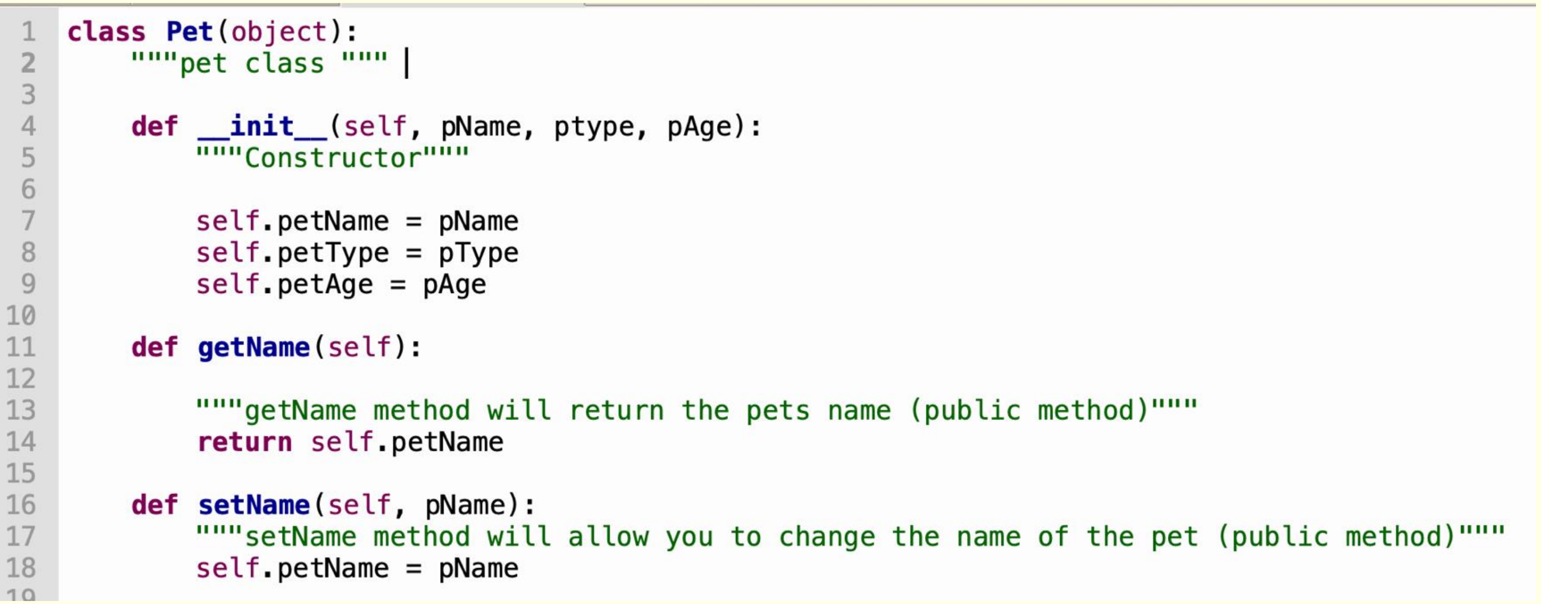
Why Use Object Oriented Programming (OOP)

- The **object-oriented programming paradigm (OOP)** introduces a fundamentally different approach to program design.
- The term 'object' in object-oriented programming represents a specific way of organising code
- OOP defines an object as an independent entity
- OOP defines the attributes (data) of the object and the methods that can be applied to it

Why Use Object Oriented Programming (OOP)

- Attributes could be private to restrict accidental changes
- Can modify and maintain existing code much more easily
- Reuse of code in other programs through libraries
- Has better structure and design so is suited to big projects
- Clear modular structure with a defined interface and which can abstract and hide away details

Coding classes in python (See Trinket)



```python
class Pet(object):
    """pet class """

    def __init__(self, pName, ptype, pAge):
        """Constructor"""

        self.petName = pName
        self.petType = pType
        self.petAge = pAge

    def getName(self):

        """getName method will return the pets name (public method)"""
        return self.petName

    def setName(self, pName):
        """setName method will allow you to change the name of the pet (public method)"""
        self.petName = pName
```

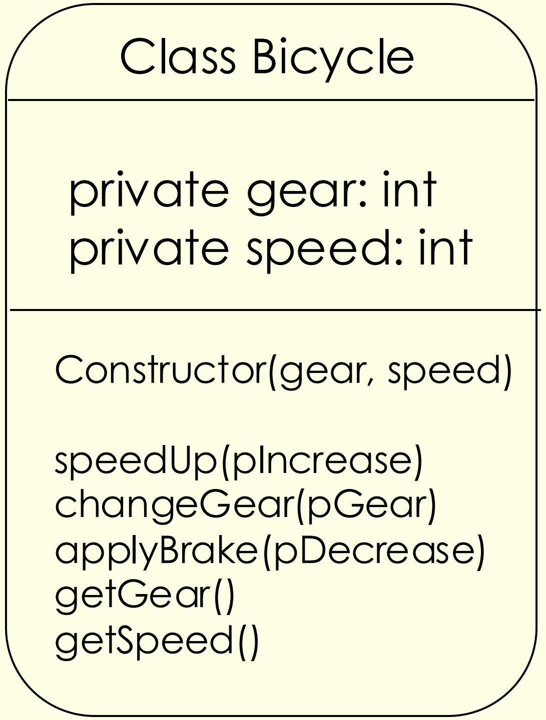
```python
#Main program
if __name__ == "__main__":

    name = input("Name of pet")
    petType = input("Type of pet")
    age = int(input("Age of pet"))

    # object instance of pet called myPet
    myPet = Pet(name, petType, age)

    #print just the name of myPet
    print(myPet.getName())
    #change the name
    myPet.setName("Fred")
    print(myPet.getName())
```

Challenge:

In the code above write a method for displaying all the pet attributes. Then use this method in the main program.

Bicycle class activity

```
class Bicycle( )
     #Class variables
    private gear
    private speed

 """ Constructor """
    public procedure new()
         gear = 1 #using the class variables
         speed = 0
      end procedure

 """ Getter methods return the value of an attribute """
    public function getGear()
        return gear
    end function

""" Getter methods return the value of an attribute """
    public function getSpeed()
        return speed
    end function
```



Class Bicycle

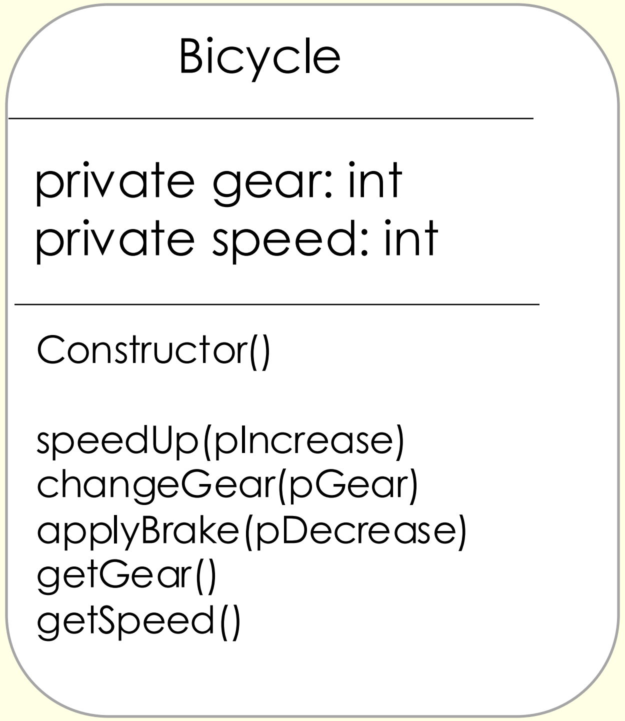
private gear: int
private speed: int

Constructor(gear, speed)

speedUp(pIncrease)
changeGear(pGear)
applyBrake(pDecrease)
getGear()
getSpeed()

Class Methods

```
""" method to change the speed attribute """
procedure speedUp(pIncrease)
    speed += pIncrease
end procedure

""’method to change the gear attribute """
procedure changeGear(pGear)
    gear = pGear
end procedure

""’method to change the speed attribute """
procedure applyBrake(pDecrease)
    speed -=pDecrease
end procedure
```



Bicycle

private gear: int
private speed: int

Constructor()

speedUp(pIncrease)
changeGear(pGear)
applyBrake(pDecrease)
getGear()
getSpeed()

Advantages

Modularity
One of the advantages of the object-oriented languages are they are modular, which means it is easy to change one module without affecting the other.

Encapsulation
Protecting data/ Can't be changed accidentally as data should never be accessed directly. Data inside our object is not modified unexpectedly by external code in a completely different part of our program.

Code re-use
Object oriented programming promotes the code reuse.

Extensibility
Extensibility in an object oriented programming is easy. New functionality is easy to add without affecting existing functionality.

Disadvantages

Object oriented programs are hard to develop. Real world problems don’t always fit into the objects.

For smaller programs, it may be easier to use the list of commands rather than a full blown object oriented program.

How is OOP different from procedural programming?

Procedural	Object-oriented
Problem is divided into smaller parts called **procedures** (or functions), and systems are built by combining procedures. These procedures share data by passing data between them or through global variables. In procedural it would need to make sure the correct values are passed and returned, or global variables may be required which uses excess memory.	Problem is divided into smaller parts called **objects**, and systems are built around objects. Objects are representations of things that exist in the real world that we wish to model in a computer system. The object's data and the methods are bundled together in a class, making the code more modular and reusable. OOP reduces amount of code needed therefore fewer errors are likely as code is written once and then used multiple times

Procedural vs OOP

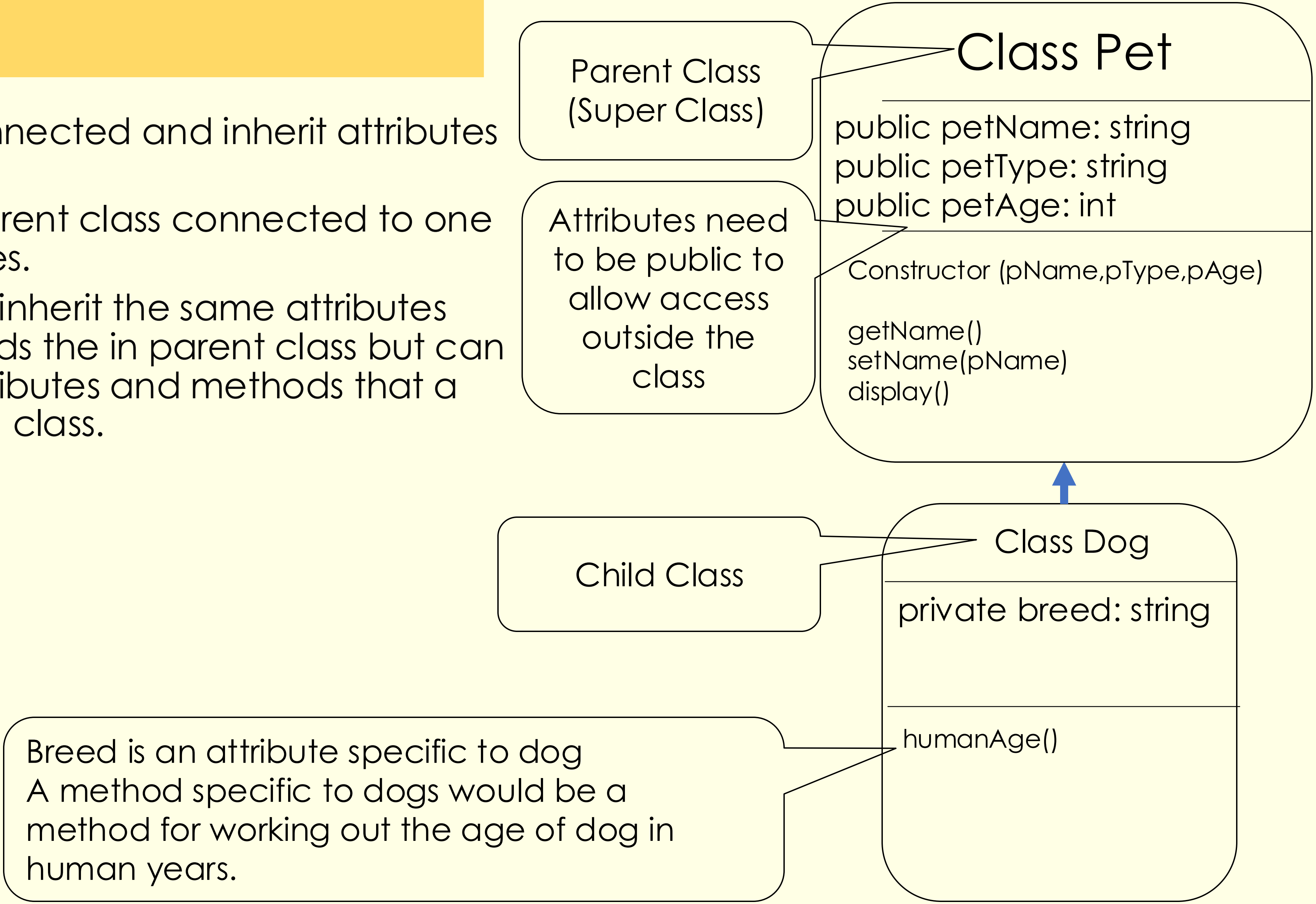
Feature	**Procedural Programming**	**Object-Oriented Programming (OOP)**
Approach	Step-by-step (functions)	Uses objects with data & behaviour
Data & Functions	Separate—functions modify data but don't belong to it	Bundled together in objects (like a real-world entity)
Structure	Organised into procedures (functions)	Organised into classes and objects
Best for	Small, simple programs	Large, complex applications
Reusability	Limited—code needs to be rewritten for different uses	High—objects and classes can be reused
Example Use Cases	Simple calculators, scripts, basic automation	Games, web apps, large software projects

Summary

- Classes, this a template. It will define what attributes and methods an object should have.
- Objects, when you create an instance of a class. Each object that is instantiated from the same class will share the same attributes and methods.
- Encapsulation, this protects attributes of an object by making them private so that they can’t be accessed or altered accidentally by other objects.

Inheritance

- Classes can be connected and inherit attributes and behaviours.
- You can have a parent class connected to one or more child classes.
- The child class can inherit the same attributes and use the methods the in parent class but can also have extra attributes and methods that a specific to the child class.



Parent Class (Super Class)

Attributes need to be public to allow access outside the class

Class Pet

public petName: string
public petType: string
public petAge: int

Constructor (pName,pType,pAge)

getName()
setName(pName)
display()

Child Class

Class Dog

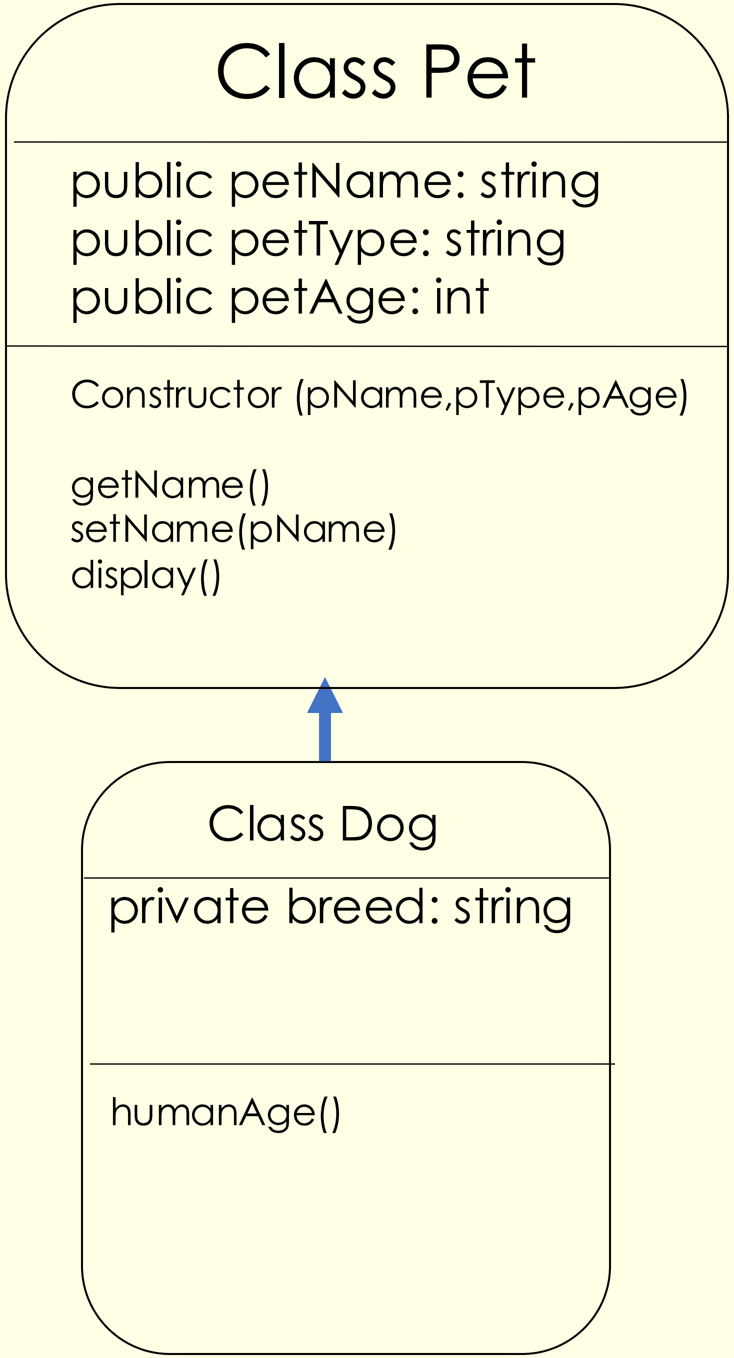
private breed: string

humanAge()

Breed is an attribute specific to dog
A method specific to dogs would be a method for working out the age of dog in human years.

Inheritance

Main advantages of using inheritance	**Main disadvantage of using inheritance**
Minimise duplicate code	• is that the two classes (base and inherited class) get **tightly coupled**. • This means one cannot be used independent of each other.



Class Pet

public petName: string
public petType: string
public petAge: int

Constructor (pName,pType,pAge)

getName()
setName(pName)
display()

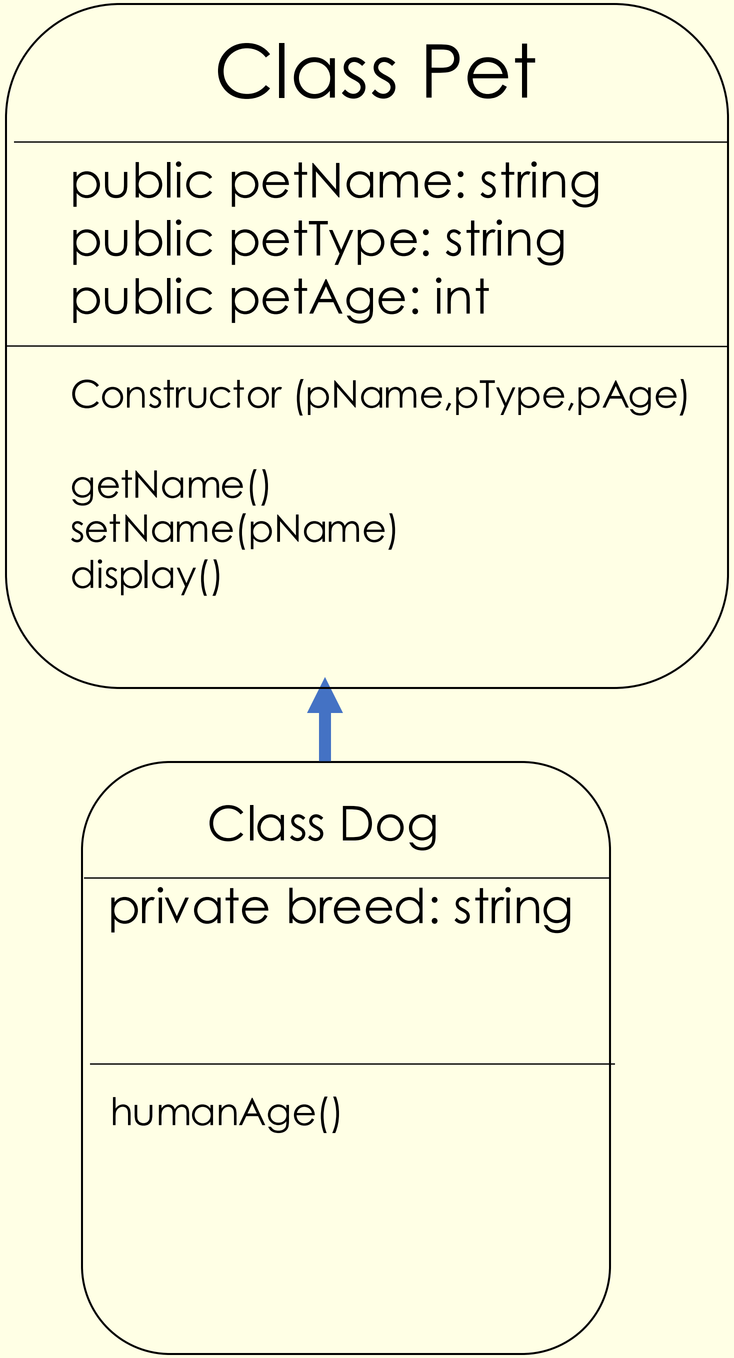
Class Dog

private breed: string

humanAge()

```
class dog inherits pet

   “”””constructor class””””
  public procedure new (pName, pType, pAge, pBreed)
    super(pName, pType, pAge)
    breed = pBreed

  end procedure

  public procedure humanAge()

    humanAge = petAge * 7
    print(humanAge)

  end procedure
end class
```

Super is used to refer to the parent (or superclass) of a class.



Class Pet

public petName: string
public petType: string
public petAge: int

Constructor (pName,pType,pAge)

getName()
setName(pName)
display()

Class Dog

private breed: string

humanAge()

```
class dog inherits pet

  “”””constructor class””””
  public procedure new (pName, pType, pAge, pBreed)
    super.petName = (pName
    super.petType = pType
    super.petAge = pAge
    breed = pBreed

  end procedure

  public procedure humanAge()

    humanAge = petAge * 7
    print(humanAge)

  end procedure
 end class
```

Super is used to refer to the parent (or superclass) of a class.

Using inheritance in the main program

```
myDog = new Dog(“Fido”, “dog”, 3,”Scottish Terrier”)
```

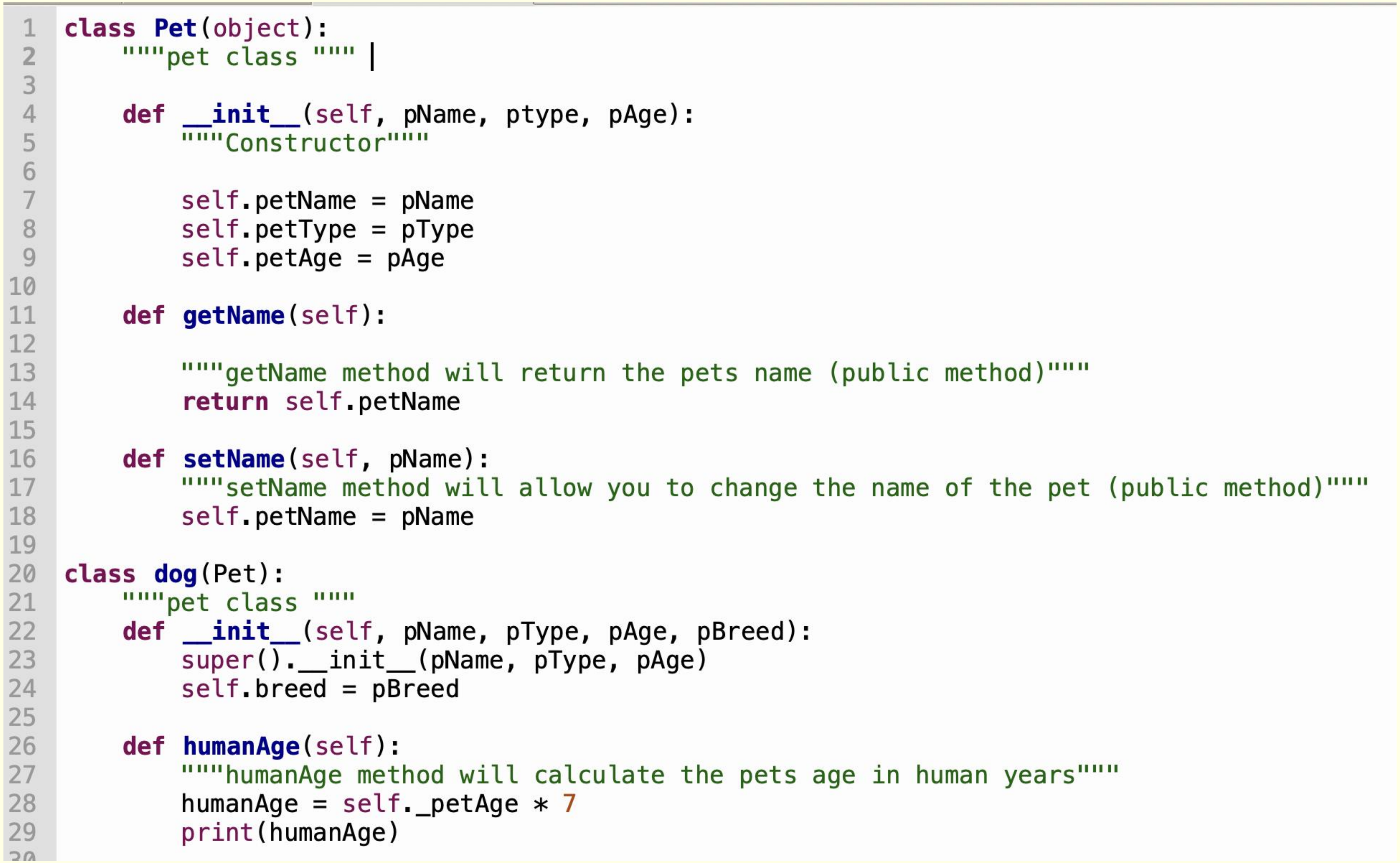
Inherits the getName() method from the Pet class:

```
print(myDog.getName())
```

Using the procedure in the dog class:

```
print(myDog.humanAge())
```

Python implementation



```python
class Pet(object):
    """pet class """

    def __init__(self, pName, ptype, pAge):
        """Constructor"""

        self.petName = pName
        self.petType = pType
        self.petAge = pAge

    def getName(self):

        """getName method will return the pets name (public method)"""
        return self.petName

    def setName(self, pName):
        """setName method will allow you to change the name of the pet (public method)"""
        self.petName = pName

class dog(Pet):
    """pet class """
    def __init__(self, pName, pType, pAge, pBreed):
        super().__init__(pName, pType, pAge)
        self.breed = pBreed

    def humanAge(self):
        """humanAge method will calculate the pets age in human years"""
        humanAge = self._petAge * 7
        print(humanAge)
```

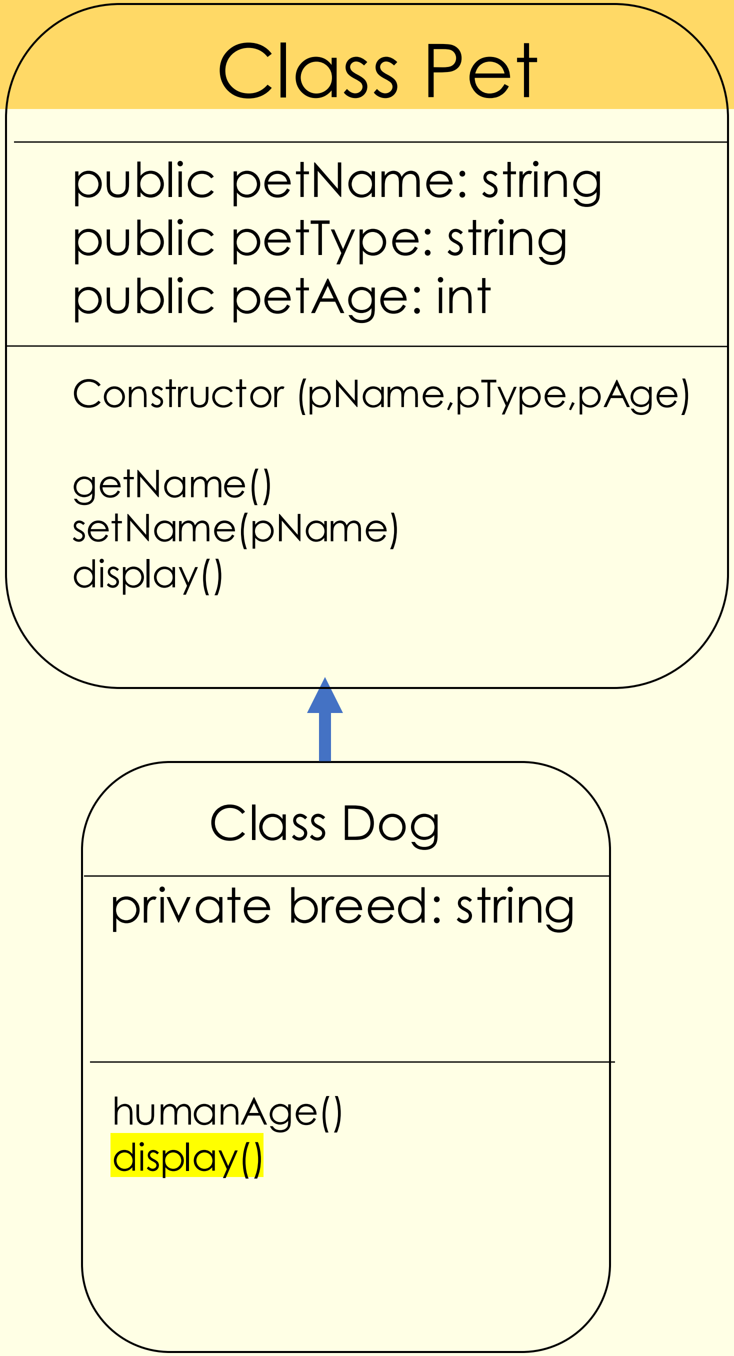
Polymorphism

Polymorphism
Polymorphism lets the same method name work in different ways, either in different classes or with different parameters in the same class.

Method Overriding
A child class replaces a method from the parent class with its own version.

```
Parent Class – Pets

procedure display()
    print( “Name”, petName)
    print(“Type”, petType
    print(“Age”, petAge)
end procedure
```

```
Child Class – Dogs

procedure display()
    print( “Name”, self.name)
    print(“Type”, petType)
    print(“Age”, petAge)
    print(“Breed”, breed)
end procedure
```



Class Pet

public petName: string
public petType: string
public petAge: int

Constructor (pName,pType,pAge)

getName()
setName(pName)
display()

Class Dog

private breed: string

humanAge()
display()

How to apply overriding polymorphism

```
mydog = New Dog("Fido","dog",3,"Scottish Terrier")

mydog.display() #use the over-riding method in Dog class

mydog.super.display() #use the parent class method
```

Over-Loading Polymorphism

Method Overloading, a way to create multiple methods with the same name but different arguments, is not possible in Python.

Overloading Polymorphism is used when a method can be implemented more than once to accept different parameters.

We could have a method in the Pet Class to define the data of birth:

The method could accept a date parameter as a string in the "DD/MM/YYYY" format.

```
PROCEDURE setDoB(string:date) # date in DD/MM/YYYY format
```

The same method can be implemented a second time, taking three parameters as follows:

```
PROCEDURE setDoB(integer:day, integer:month, integer:year)
```

Summary

- Inheritance, when a sub class takes on the attributes and methods from a superclass/parent class.
- It can also have its own extra attributes/methods.
- Overriding, when a method name is the same in a parent and sub class, then the method in the parent/super class will be overridden (polymorphism)

Class as a separate file

- If you wanted to have the main program as a separate file to the class then at the top of the source code use:

```
from somefile import className
```

- For example

```
from pet import Pet
```