# Pygame Workshop Two

Kieran Vickers

# Starter

# Rules

You have (around) 5 minutes to complete three challenges (on the next slide). You must use Python. You must type **as little** as you can, but you can copy-paste as much as you wish. You may add comments.
Get a browser and IDLE open...

# Challenges

- Create a function that prints the first n numbers in the Fibonacci sequence
- Create a function that plays FizzBuzz given a number (3/5 version)
- Create a file that plays space invadors using PyGame

# What we aim to cover

# What we aim to cover

- Introduce OO in Python
- Cover inheritance in Python
- Introduce OO concepts to PyGame

# You will need

- . . . IDLE open (try `import pygame` now)
- . . . A broswer open on a search engine
- . . . A browser open on the **Pygame Docs**
- . . . to be willing to ask questions

# Main

# Objects objects objects

Have a look at task0.py. Can you spot the redundancy? There are a lot of class attributes that are repeated, which we could instead inherit.

Can you suggest a class hierarchy (with attribute list) for these classes?

# Discuss

Do we agree the hierarchy?

# Classes

Classes in python are defined using the `class` keyword. In Object-Oriented programming we aim to collect the data representing objects of a single type into a class, along with the functions that act on this data.

# Classes

```
class Dog:
    def __init__(self, name, date_of_birth):
        self.name = name
        self.date_of_birth = date_of_birth

    def greet(self):
        return f"Hello, {self.name}, who's a good dog?"
```

# Inheritance

Given a hierarchical structure of classes, we can inherit attributes and functionality from a parent class, then add more attributes and functions.

In Python, we can call `super()` to invoke the parent or superclasses method. For example, on the next slide see how we use `super().__init__()`. It is considered a must to call this in any class that inherits and overrides `__init__`.

# Inheritance

```
class Animal:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return f"Hello, {self.name}."

class Human(Animal):
    def __init__(self, name, languages_spoken):
        super().__init__(name)
        self.languages_spoken = languages_spoken
    def speaks(self):
        return ", ".join(self.languages_spoken)
```

# Inheritance Task

Take *task0.py* and rewrite it so that you only define each attribute once.

The code at the bottom should still run and give the same output after your changes.

# Overriding Methods

When we inherit from a parent/super class, objects of the child class will have all the attributes and methods of its parents, exactly the same.

# Overriding Methods

```
class Animal:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return f"Hello, {self.name}."

class Human(Animal):
    pass
```

# Overriding Methods

When we inherit from a parent/super class, objects of the child class will have all the attributes and methods of its parents, exactly the same.

You can then override the functions of the parent class in the child class. The most common function to override is the `__init__` function, but you can override any function.

# Overriding Task

Look at *task1.py*. It is an example answer to the previous task, with functions added to access the attributes. Replace the print statements with different print statements that use these functions. Can you write another function in a child class that overrides a parent class's function?

# Pygame

# Pygame

So, object oriented pygame, why is this useful?

# Fetch - Decode - Execute

Recall the FDE cycle, a three (ish) phase cycle of the CPU.

Similarly, graphical interfaces use a similar three phase cycle.

# Event handle - update - display

The handle, update, display loop is a common framework used in graphical interfaces. Many GUI toolkits abstract this. In Python's *tkinter*, you only really have to handle events.

# Event handling

In pygame, all events are stored in a queue waiting for the programmer to handle them. Each event is an object, of type Event. It is called `pygame.event` and we loop over it in a for loop:

```
python for event in pygame.event.get():      # Do
something with the event.      pass
```

# Events in Pygame

With the pygame docs you have open, can you find different categories of events in pygame?

What attributes do ALL types of events have, and what attributes do only some have?

# Hanlding events in Pygame

```
import pygame, pygame.event
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        should_quit_game = True
    elif event.type == pygame.MOUSEMOTION:
        mouse_position = event.pos
    elif event.type == pygame.KEYUP:
        print(f"Key pressed is {event.key}.")
```

# Updating in Pygame

The update section of the loop, also called stepping, is sometimes skipped or incorporated into draw. This is when you update the state of the internal representation based on things that aren't events. One example: animation (changing the x coordinate of a sprite every frame).

# Drawing in Pygame

Have a look at the pygame docs again, this time at draw. What shapes can we draw?

A *Surface* object in pygame is a pygame object for representing images, and what you draw to a surface we can then draw to the screen.

# A Worked Example

Have a look at task2.py. It is the same game we did last week, only this time I've used the inheritance to ensure all drawable *Thing*s can be looped over.

This isn't a great example, let's write a better example.

# MyFirstGame

# Introduction

Over the next few tasks (over this workshop and the next) we are going to slowly build a “game” (twice).

Please do save this work somewhere you can find it.

# Draw

Have a look at task3.py. It simply displays a gray screen, but has been written in an object oriented way. There are two classes for you to complete: `House` and `Dog`. For each, create a `.__init__(...)` function and override the `.draw(surface)` method. At the moment, you decide what needs passed into the init function. Use the documentation.

# Update/animation

For the `Dog` class, override the `.update()` function to make the dog bob up and down over time . Hint: you don't want to bob entirely every frame.

# Event Handling

For the `Dog` again, override the `.on_event(event)` function to make the dog move left and right when the arrow keys OR the 'A'/'D' keys are pressed. What is the difference between between KEYDOWN and KEYUP?

See you next session!

# See you in two weeks time!

- Hierarchical object structures, parent/child relationships
- Inter-object interaction
- Other PyGame features