PyGame Hello World Programme

Starting

To begin coding with PyGame we will write a short programme that will display a blank window with a window caption that reads

```
Hello World!
```

This is a very simple programme that will allow us to get used to the basics of using PyGame.

Open your python editor, e.g. IDLE or Komodo Edit, etc and open a new Python 3 file, name it

```
PyGameHelloWorld.py
```

The Code

For the programme to be able to use the PyGame modules we must first **import** the PyGame library for this program. To do this we start the programme with the lines:

```
import pygame # Import all available pygame modules
```

For this programme we will also add

```
import sys
```

To add some of the PyGame constants and functions to the **global namespace** we will then add the following lines:

```
from pygame.locals import *
```

Some of the modules within PyGame require initialisation, we could initialise each of the modules we need, one by one, but we can also just initialise all of the modules by adding the following line:

```
pygame.init()
```

That's the basic setup done. Now we need to generate a window to display on the screen to the programme's users. To do this we create a **display surface**. For now you can think of a display surface as being the same thing as a window.

When we create a display surface we need to tell the constructor the size of surface that we want. To do this we pass it a tuple that consists of the width and height, or x and y, dimensions in pixels. In our example we will say that we want a surface with a width of 400 pixels and a height of 300 pixels, so the tuple will be:

```
(400, 300)
```

This is what we will pass to the surface constructor as an argument. The constructor returns a reference to the surface so we need to create a variable that the reference will be assigned to, or saved in. We will call this variable:

```
DSPLAYSURF
```

Putting this altogether makes out next line of code:

```
DISPLAYSURF = pygame.display. set_mode((400, 300))
```

Here **set_mode** is a method within the **display** module that constructs the display surface for us.

Now we need to set the caption, or title, of our newly created window. We want this title to be "Hello World!" so our next line uses this string as the argument to a method, **set_caption()**, which is again in the display module.

```
pygame.display.set_caption('Hello World!')
```

We want this window to be displayed for as long as the program is running so the next thing we add is a loop that runs forever, an infinite loop. This is often referred to as the **main game loop**. To do this we just need to add:

```
while true:
```

Because this loop will run forever we need to add some way of terminating the program inside of the loop. The way a window is normally closed is by clicking the "x" button at the top of the window. Clicking this button create an **event** named QUIT. Luckily for us PyGame listens out for events from the program and makes a **list** of all the events that it hears. All we need to do then is check this list to see if the QUIT event is on it, if it is we then know the programme should close the window and stop.

To check through all the events on the event list we need another loop, this inside the while loop of course, so it should be intented:

```
      for event in pygame.event.get()
```

Inside this loop we need to check if each of the events we are looking at is the QUIT event, so we add an if statement:

```
            if event.type == QUIT
```

If it happens that an event on the list is a QUIT event, that is, our user has clicked the "x" button, then the if statement condition will become true and we should close the window and the programme.

```
                  pygame.quit()     # Quit pygame
                  sys.exit()  # Exit the python programme
```

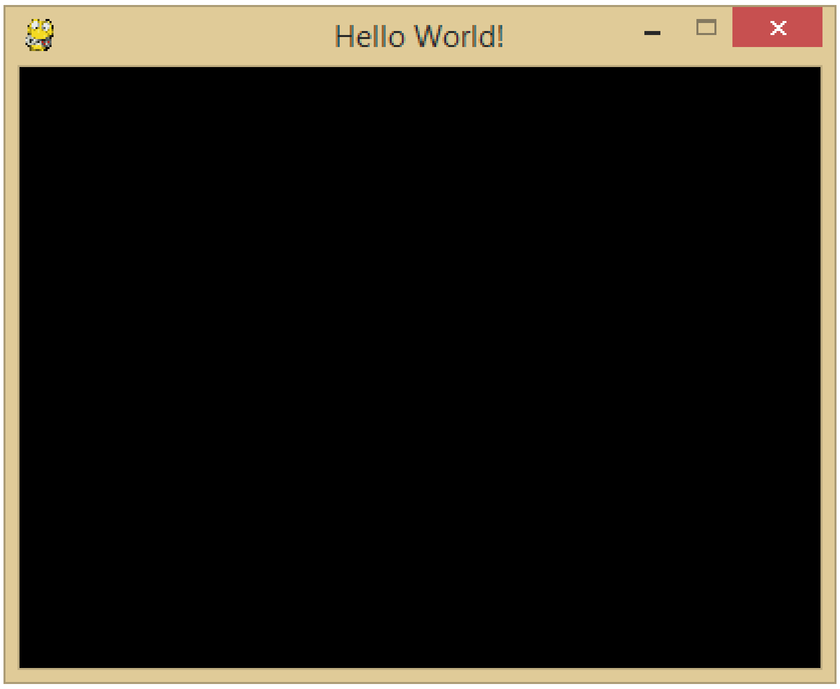
The first line above quits PyGame by cleaning up the modules that were initialised at the start with the `pygame.init()` line. The second line exits ython, ths ending the programme.

We are not quite finished here. We need to make sure that if the programme has *not* received a QUIT event it will make sure the window keeps up to date in case events happen later on or we try to draw something to the window. This means we need one more line, outside of the if statement and the for loop (but still inside the while loop) we add the final line:

```
      pygame.display.update()
```

The Result

Now if the programme is correct we should see the following window when the programme is run:



And there we have it, the most boring game ever! To build more, check out the PyGame documentation at http://www.pygame.org/docs/, and maybe start with the bouncing ball programme.