- `f4(l)` : returns all the initial sublists of `l`, e.g. `f4(list("Thor"))` should return

```
[[], ['T'], ['T', 'h'], ['T', 'h', 'o'],
['T', 'h', 'o', 'r']]
```

- `f5(l)` : returns all the lists which can be obtained from l by leaving out elements (the order doesn't matter), e.g. `f4(list("Thor"))` should return `[[], ['r'], ['o'], ['o', 'r'], ['h'], ['h', 'r'], ['h', 'o'], ['h', 'o', 'r'], ['T'], ['T', 'r'], ['T', 'o'], ['T', 'o', 'r'], ['T', 'h'], ['T', 'h', 'r'], ['T', 'h', 'o'], ['T', 'h', 'o', 'r']]`

2. Define a Python function `fib(x,y)` that calculates the Fibonacci sequence as an iterator. E.g. running `show(fib(1,1))` should produce the following output:

```
1    More? y
1    More? y
2    More? y
3    More? y
5    More? y
8    More? y
13   More? y
21   More? y
34   More? y
55   More? y
89   More? n
```


7 Implementing games with pygame

In this chapter, we are going to learn how to implement games in Python! So far, we have learned the main concepts about programming, including imperative programming, recursion, object oriented programming and functional programming. With those concepts, we are now in a very good position to implement our own game, and to do so, we are going to use a library called **pygame**. Many other libraries are available, but we found this one particularly nice and easy to use!

In our course, we ask the students to implement a game in a group. The group chooses the game, very popular are old arcade games. So far we have only written small programs and only by one person (we hope). The project is a practical introduction to *software engineering*, which involves constructing bigger programs and working together with other people. There is a lot of research going on, but we think it is better just to learn software engineering by doing. Once you realise what the basic issues are, you may be interested to learn more and research different software engineering methodologies.

We choose games because it is easy to see when a game is good - it is fun to play! Also, actually it is quite fun to write a game. We also run a contest to see which group has written the best game.

Hence ideally try to do this assignment as a group but if that is not possible, it is still a good idea to develop a game yourself. However, also the code quality matters for the assessment! Is the code easy to read?, easy to change? and well commented (not too much, not too little)?

Here is some advice how to go about running a software

project:

- Use an online repository with a software maintenance system such as `git`. There are number of free ones like github[1] or bitbucket[2]. They enable you to share the code base, it keeps track of changes and has a number of other useful facilities.
- You need to communicate regularly, for this purpose there are also online sites like slack[3] available which can also be linked to the repository.
- Develop your code in pairs, pairing people with different abilities.
- Don't try to plan every detail but allow some experimentation, play around with prototypes and be prepared to throw code away and rewrite it (refactoring).
- Try to plan a rough schedule which also includes testing and bug fixes.
- Try to make sure that everybody is involved allowing for different level of programming skills.
- Take commitments seriously and don't diverge from agreed specs without consulting the rest of the group.

7.1 What is Pygame?

Pygame[4] is a Free and Open Source Python programming language library for making multimedia applications like games. You might now be wondering, but *What is a library?*

A library is a collection of code (functions and methods that someone else wrote) that we can use in our programs by importing them. We have been doing this before in this book, when we imported the *pickle* library or the *random* library. Most libraries follow the Object Oriented programming paradigm,

[1] `github.com`
[2] `butbucket.org`
[3] `slack.com`
[4] `https://www.pygame.org/docs/`

so that, we will be able to create objects of different classes, and apply a number of methods on those objects.

You can find the pygame documentation at: `https://www.pygame.org/docs/`, including installation guidelines and the API (Application Programming Interface) with all the functionality of this library.

Important Note: the behaviour of some of the methods we will be using below may differ slightly depending on the Operating System you are running Python.

7.1.1 Installation and basics with pygame

Below we provide a very simple instruction to install **pygame** on your computer. Note that this might not work in all cases, if this doesn't work we refer you to the original pygame guidelines. If you have installed anaconda on your computer, (in most cases) you can simply use `pip` to install the library:

```
pip install pygame
```

Whenever installed, you can use the library on your preferred IDE (i.e. jupyter or Spyder). Having said that, we would recommend you to use spyder for this.

The first step is to import the library and initialise it.

```
In : import pygame

pygame 1.9.4
Hello from the pygame community.
https://www.pygame.org/contribute.html
```

To make sure that pygame is correctly installed on your computer, you should try to initialise it, running the following instruction:

```
In : pygame.init()

Out: (6, 0)
```

You really want to know what the `(6,0)` means? Check out the documentation!

When we import a library, we typically have to use the name of the library, before using any of the methods implemented in that library (e.g. `pygame.init()`). Alternatively, you could import the library as follows:

```
from pygame import init
```

In this way, we could call `init` directly. Another option is to import everything from pygame:

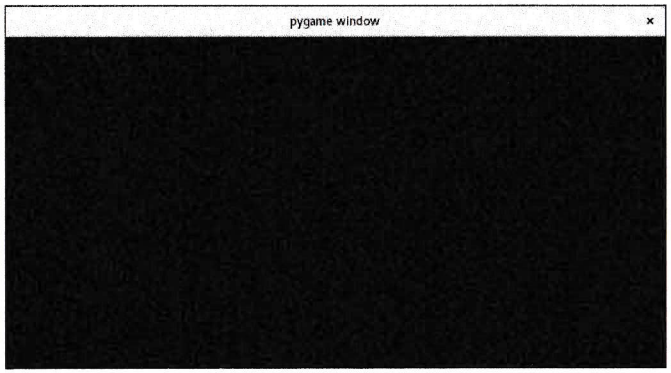
```
from pygame import *
```

However, this is not really a good practice, because otherwise you may have some clashes (i.e. name of functions on pygame may be the same that those you use in your program). I will be using the first option in this notebook.

To use the library, one of the first things we will need to do is to initialise the `display`, where we will place/paint things for our game. `display` is a class defined in the pygame library, which contains various methods. Among them, we can set the size of the display we want to create.

```
In : screen = pygame.display.set_mode((800,400))
```

When we run the above line, you should get a new window that looks like this:



We have also stored the output of running that method on an object `screen`. What is the type of that object? The function display returns an object of class Surface:

```
In : type(screen)

Out: pygame.Surface
```

The `Surface`[5] class provides a good number of methods to do a lot of things.

Our first task will be to fill that screen with a solid colour. Looking at the API, we should use `screen.fill(argument)`, what are the arguments?

For this, we recommend you to use the API, or the in-line help! (Running `help(pygame.Surface.fill)`)

Before using this function, we need to find the class `Color`[6] and see how it works. Run `help(pygame.Color)` to see the documentation of this class.

Ok, we can have an object of class `Color` by simply calling `Color("name")`, where "name" is the name of a colour.

```
In : screen.fill(pygame.Color("red"))

Out: <rect(0, 0, 800, 400)>
```

Anytime that we make some changes on the surface (`screen` object), we need to make it visible; we do this by 'flipping' it.

```
In : pygame.display.flip()

  ---------------------------------------------------
  NameError Traceback (most recent call last)
  <ipython-input-1-d6f4c9bcf213> in <module>() --->
1pygame.display.flip()
  NameError:  name 'pygame' is not defined
```

If you run the above cell, you should now see the red background.

[5] https://www.pygame.org/docs/ref/surface.html
[6] https://www.pygame.org/docs/ref/color.html

However, what's wrong about our program? Have you tried to close the window? Try! you will see we cannot do it! (normally).

To do so, we need to use an event called `quit()`. This instruction will be using a system call to the operating system. We know that this may not work well in some versions of MacOS!

```
In : pygame.quit()
```

The previous line removed the object screen, and we will need to run the previous cells again to create it.

If you are using MacOS, we found the following code useful to close the window:

```
import os

# After the pygame.quit()
os._exit(0)
```

As we don't want that our program finishes immediately after creating the screen, what we need to do is to create a loop that waits for 'events'.

What are pygame events[7]? They are the way to interact with the user.

The event `QUIT` will be triggered when we try to exit the window (using the Operating system cross X).

We are going to create a for loop that reads events from a queue and then analyses them to decide what to do:

The method for that is: `pygame.event.poll()`

We can check the type of the event object returned from that function using the attribute `type`

```
In : screen = pygame.display.set_mode((800,400))

    while True:
        e = pygame.event.poll()
        if e.type == pygame.QUIT :
            break
    quit()
```

[7] https://www.pygame.org/docs/ref/event.html

With the above cell, you should now be able to close the window when clicking on the cross in the top corner.

Alright, something that already bothers me about the above code, is that we have hard-coded the width and height of the screen. I would rather use of global variables for this:

```
In : WIDTH = 800
     HEIGHT = 400

     screen = pygame.display.set_mode((WIDTH,HEIGHT))
     while True:
         e = pygame.event.poll()
         if e.type == pygame.QUIT :
             break
     quit()
```

7.2 The Pong Game

Alright, so now we are going to implement a very simple game. In particular, we are going to create a reduced version of the game pong[8].

Just for a single player, as if we were playing squash.

Starting from a black background screen, what we want to do first is to draw three borders (the walls).

For that, we are going to *draw* a rectangle[9].

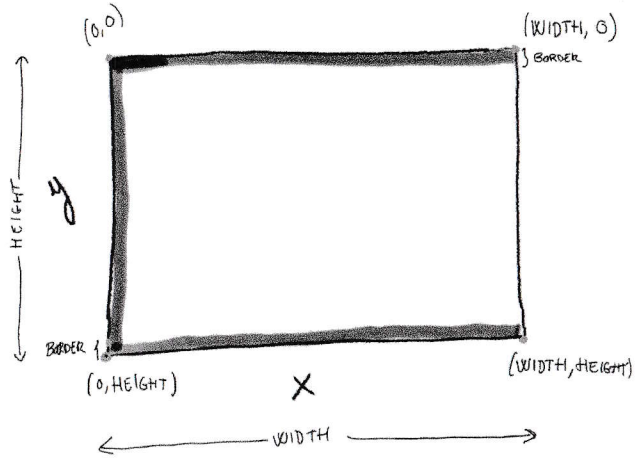
The function `pygame.draw.rect` is expecting a parameter `Rect` (note that class names are case sensitive!) with the specs of the rectangle we want to paint.

In particular, an object of class `Rect` can be created indicating the coordinates for the left top corner of the rectangle, and the desired width and height! You can read more running `help(Rect)`.

The coordinates **(x,y)** indicate the pixels in which the top left corner of every rectangle is. In the figure below, I have drawn the screen, indicating the coordinates for the three rectangles we need to draw the walls.

[8] https://en.wikipedia.org/wiki/Pong

[9] https://www.pygame.org/docs/ref/draw.html#pygame.draw.rect



The first rectangle (the one on the top) will start at (0,0), and the width of that rectangle should be equal to the WIDTH of the entire screen. The height of this rectangle, however, should be relatively small, just a few pixels to represent the thickness of the wall.

To do this, we define a new global variable `BORDER` of size 10 for example. And we are now ready to draw this first rectangle using the `pygame.draw.rect` function, and creating a `Rect` object, starting at (0,0), width =WIDTH, and height = BORDER. Something like:

```
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,WIDTH, BORDER))
```

The second rectangle (the vertical rectangle on the left of the screen) will also start at (0,0), however, its width should be just the thickness of the wall (i.e. BORDER), and the height of it should be equal to the entire HEIGHT of the screen. So:

```
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,BORDER, HEIGHT))
```

How about the last rectangle? What are the coordinates? I put in red the coordinates for this rectangle, just above the left

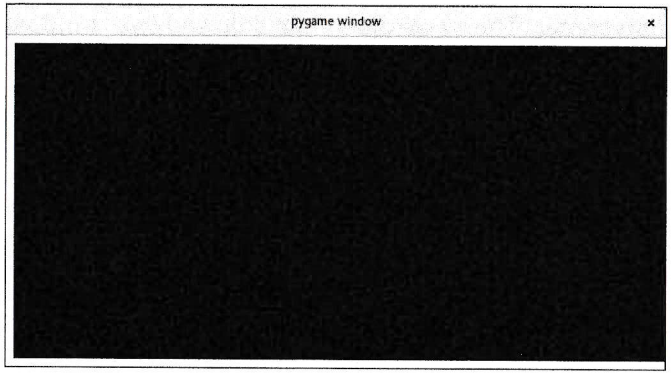
bottom corner. The x coordinate can still be 0 (we could move it a bit to the right, but there is not need for that). However, the y coordinate, is all the way down HEIGHT minus the border size! The width of this rectangle will also be equal to the entire WIDTH of the screen, and the height is just the size of the BORDER.

```
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,HEIGHT-BORDER,WIDTH,\
                             BORDER))
```

So, if we put these three lines in our previous code, we end up with something like this: (don't forget to 'flip' after drawing the rectangles! otherwise the changes won't be visible!)

```
In : BORDER = 10

    WIDTH = 800
    HEIGHT = 400


    screen = pygame.display.set_mode((WIDTH,HEIGHT))
    screen.fill(pygame.Color("black"))

    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,0,WIDTH, BORDER))
    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,0,BORDER, HEIGHT))
    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,HEIGHT-BORDER,WIDTH,\
                                 BORDER))

    pygame.display.flip()
    while True:
        e = pygame.event.poll()
        if e.type == pygame.QUIT :
            break
    quit()
```

This should look like this:



So far, we have simply designed the layout of the game, but now we need to create the **Ball** and the **Paddle** to play the game. Both are 'objects' that will move on the screen, so they are going to have some functionality associated to it. Do you know what we need to do? Yes, we better define a class for `Ball` and one for `Paddle` in which we will code the functionality of both type of objects!

```
In : class Ball:
        pass

     class Paddle:
        pass
```

7.2.1 The Ball class

Within the Ball class, we need to draw a circle this time.

`pygame.draw.circle` draws a circular shape on the Surface. The `pos` argument is the centre of the circle, and radius is the size. The width argument is the thickness to draw the outer edge. If width is zero then the circle will be filled.

An object of class `Ball` will therefore have some coordinates (x,y) which could initialise to some values within the **init** function. The ball will have a given radius, this could be for example a class variable `RADIUS`. So, this could look like this:

```
In : class Ball:

        RADIUS = 10

        def __init__(self, x,y):
            self.x = x
            self.y = y
```


We now want to draw the ball on the coordinates established in `self.x` and `self.y`; what we can do is to add a method `show` that will print the ball in a particular colour (we will see later why this is particularly useful!). Note that this method will be using the pygame.draw.circle method and will act on the `screen` we initialised before. Thus, `screen` is going to be a global variable, and we indicate that specifically in the code as below:

```
In : class Ball:

        RADIUS = 10

        def __init__(self, x,y):
            self.x = x
            self.y = y

        def show(self, colour):
            global screen
            pygame.draw.circle(screen, colour,\
                               (self.x, self.y), self.RADIUS)
```

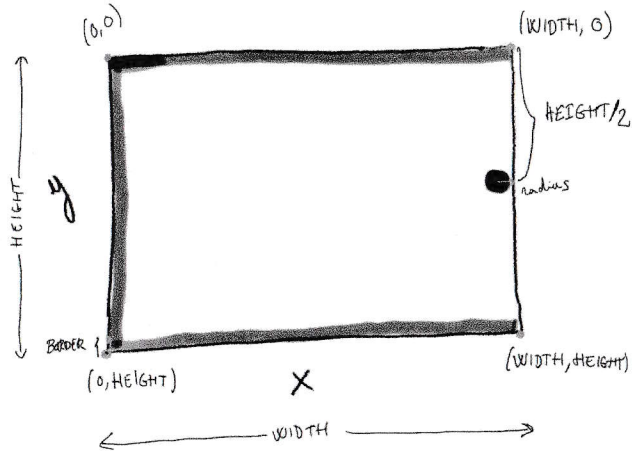
Alright, we now have the class `Ball` which will allow to create ball objects!

```
ball = Ball(x,y)
```

And right after that, we could use the function `show` to plot it in a given colour (remember this needs to be a `pygame.Color`!

```
ball.show(pygame.Color("white"))
```

Initially, I would like to place the ball on right side, but by the middle, something like:



What are the coordinates of the centre of the ball?

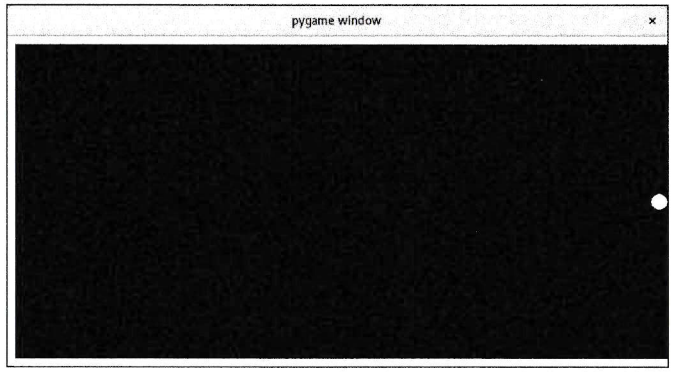
- The x coordinate is simply the WITDH - the radius of the ball.
- The y coordinate is half of the HEIGHT! we will use the operator // to obtain an integer value.

The program could be like this:

```
In : BORDER = 10

    WIDTH = 800
    HEIGHT = 400

    screen = pygame.display.set_mode((WIDTH,HEIGHT))
    screen.fill(pygame.Color("black"))

    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,0,WIDTH, BORDER))
    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,0,BORDER, HEIGHT))
    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,HEIGHT-BORDER,WIDTH,\
                                 BORDER))

    ball = Ball(WIDTH-Ball.RADIUS, HEIGHT//2)
    ball.show(pygame.Color("white"))
```



```
pygame.display.flip()
while True:
    e = pygame.event.poll()
    if e.type == pygame.QUIT :
        break
quit()
```



We now have to make the ball move, but how do we do that?

Basically, what we have to do, is to 'remove' the ball from the screen, move it a few pixels and draw it again. For that, we will have to define a `VELOCITY` that will be applied to both X and Y coordinates of the ball to make it move.

Our class `Ball` needs to have two more attributes velocity for x and y coordinates. So we modify the `__init__` method to include the velocity for the x axis (`vx`) and the y axis (`vy`).

```
def __init__(self, x,y, vx, vy):
    self.x = x
    self.y = y
    self.vx = vx
    self.vy = vy
```

We also need a method to update the position (i.e. modify `self.x` and `self.y` with the new coordinates). Instead of 'eliminating the ball', how about just setting the colour to black, and then back to white in its new position? This could be something as simple as:

```
    def update(self):
        # change the colour
        self.show(Color("black"))
        # change position
        self.x = self.x + self.vx
        self.y = self.y + self.vy
        self.show(Color("white"))
```

Thus, the class will be like this:

```
In : class Ball:

    RADIUS = 10

    def __init__(self, x,y, vx, vy):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy

    def show(self, colour):
        global screen
        pygame.draw.circle(screen, colour,\
                           (self.x, self.y),\
                           self.RADIUS)

    def update(self):
        # change the colour
        self.show(pygame.Color("black"))
        # change position
        self.x = self.x + self.vx
        self.y = self.y + self.vy
        self.show(pygame.Color("white"))
```

So now, we have to create the object `ball` with the new class definition, and add a `VELOCITY`.

Once again, this will be a global variable; what is a good value for `VELOCITY`? Well, that depends on what we want the ball to do. Let's say I just simply want to move the ball from right to left horizontally.

As we want to move on the x-axis, from right to left, what we need is a value for `VELOCITY` that will decrement the x position of the ball. So this value must be negative. I am going to try something like -4 to see what it does. The velocity in the y-axis

will be set to 0! as I don't want this to draw a diagonal, but to go horizontally.

```
ball = Ball(WIDTH-Ball.RADIUS, HEIGHT//2,\
            -VELOCITY, 0)
```

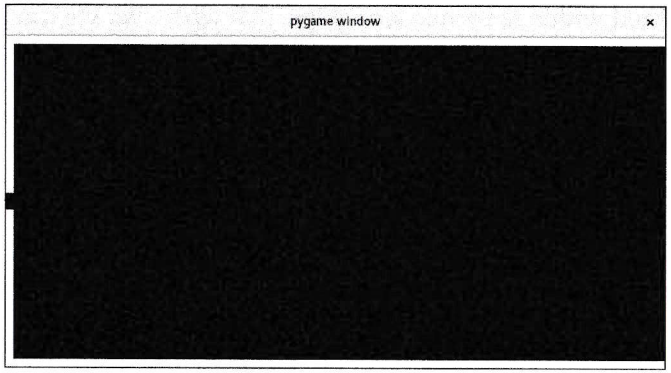
How do we make it move? well, we could call the update method within the while loop. Note that whenever we run this update method, it should be followed by a `display.flip()`, otherwise, we will not see any change!

```
while True:
    e = pygame.event.poll()
    if e.type == pygame.QUIT :
        break
    ball.update()
    pygame.display.flip()
```

The program could be like this: (it should do something funny!)

```
In : BORDER = 10

    WIDTH = 800
    HEIGHT = 400
    VELOCITY = 4

    screen = pygame.display.set_mode((WIDTH,HEIGHT))
    screen.fill(pygame.Color("black"))

    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,0,WIDTH, BORDER))
    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,0,BORDER, HEIGHT))
    pygame.draw.rect(screen, pygame.Color("white"),\
                     pygame.Rect(0,HEIGHT-BORDER,WIDTH,\
                                 BORDER))

    ball = Ball(WIDTH-Ball.RADIUS, HEIGHT//2,\
                -VELOCITY, 0)
    ball.show(pygame.Color("white"))

    pygame.display.flip()
    while True:
```

```
        e = pygame.event.poll()
        if e.type == pygame.QUIT :
            break
        ball.update()
        pygame.display.flip()
    quit()
```



Wow, that went fast!! and it clashed immediately with the left wall, leaving a back hole!

Alright, how do we prevent this from happening? In other words, I want that the ball to bounce back when it touches one of the borders! What I am saying is basically we want to change the sign of the velocity whenever it clashes with a border, which is, whenever the current coordinates + the increment in velocity (either `vx` or `vy`) will make it clash.

First thing I would do in the `update` function is to compute the new coordinates x and y before painting it. Something like this:

```
newx = self.x + self.vx
newy = self.y + self.vy
```

Now, I would check if the `newx` and the `newy` are colliding with the borders.

- For the x axis, this would be easy. Whenever `newx` is less than BORDER size, we have touched it!


```
if newx < BORDER:
        self.vx = -self.vx
```

- For the y axis, we need to check it doesn't collide with the top and bottom walls!

```
if  newy < BORDER or newy > HEIGHT-BORDER:
        self.vy = -self.vy
```

Putting everything together, the function could look like this:

```
def update(self):

    newx = self.x + self.vx
    newy = self.y + self.vy

    if newy < BORDER or newy > HEIGHT-BORDER:
        self.vy = -self.vy
    elif newx < BORDER:
        self.vx = -self.vx
    else:
        # change the colour
        self.show(pygame.Color("black"))
        self.x = self.x + self.vx
        self.y = self.y + self.vy
        self.show(pygame.Color("white"))
```

Let me put everything together again here:

```
In : class Ball:

    RADIUS = 10

    def __init__(self, x,y, vx, vy):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy

    def show(self, colour):
        global screen
        pygame.draw.circle(screen, colour,\
                           (self.x, self.y),\
```

```
                                self.RADIUS)

    def update(self):

        newx = self.x + self.vx
        newy = self.y + self.vy

        if newy < BORDER or newy > HEIGHT-BORDER:
            self.vy = -self.vy
        elif newx < BORDER:
            self.vx = -self.vx
        else:
            # change the colour
            self.show(pygame.Color("black"))
            self.x = self.x + self.vx
            self.y = self.y + self.vy
            self.show(pygame.Color("white"))

BORDER = 10

WIDTH = 800
HEIGHT = 400
VELOCITY = 10

screen = pygame.display.set_mode((WIDTH,HEIGHT))
screen.fill(pygame.Color("black"))

pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,WIDTH, BORDER))
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,BORDER, HEIGHT))
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,HEIGHT-BORDER,WIDTH,\
                             BORDER))

ball = Ball(WIDTH-Ball.RADIUS, HEIGHT//2,\
            -VELOCITY, 0)
ball.show(pygame.Color("white"))

pygame.display.flip()
while True:
    e = pygame.event.poll()
    if e.type == pygame.QUIT :
        break
    ball.update()
    pygame.display.flip()
quit()
```


If you run the above cell, well, it is kind of fun! it does reverse, however, we forgot to account for the radius of the ball! We need to slightly modify the conditions to add the radius:

```
    def update(self):

        if newy+Ball.RADIUS > HEIGHT-BORDER \
              or newy-Ball.RADIUS < BORDER :
            self.vy = -self.vy
        elif newx-Ball.RADIUS < BORDER :
            self.vx = -self.vx
        else:
            # change the colour
            self.show(Color("black"))
            self.x = newx
            self.y = newy
            self.show(Color("white"))
```

7.2.2 The Paddle class

So now we are tasked to implement the Paddle, a few things to keep in mind for this:

- The paddle will be always in the same 'x' position, we only move it up and down
- The movement needs to be managed with the mouse (new type of event)
- We need to define its width and height
- Note that when we draw a rectangle we need to provide Top-left coordinates, however, to centre the Paddle on the screen, we need to compute its centre.

For now, I am going to simply create the class `Paddle`, and initialise any attributes. As I just said, we only need to move the paddle along the y axis, this means we only need the *y* attribute. The paddle will have its own width and height, so as we did with the `Ball`, these attributes will be class variables:

```
class Paddle:
    WIDTH = 10
```

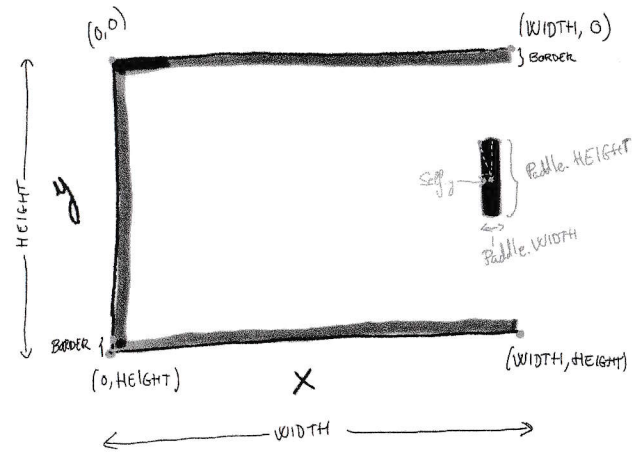
```
    HEIGHT = 80

    def __init__(self, y):
        self.y = y
```

Now we have to draw the rectangle representing the paddle. Similar to what we did before, we are going to create a `show` method that will be using `pygame.draw.rect` to draw it. Annoyingly, to draw the rectangle we need to provide the coordinates of the top left corner, but in `self.y` what we want to represent is the centre of the paddle.

To get the paddle initially centred: (this might get confusing! I recommend you to have a look at the figure I prepared below)

- The x coordinate will be the WIDTH of the screen minus the WIDTH of the Paddle (Paddle.WIDTH)
- The y coordinate will be the current position (self.y) minus half of the HEIGHT of the paddle

```
pygame.draw.rect(screen, colour,\
                 pygame.Rect(WIDTH-self.WIDTH,\
                      self.y-self.HEIGHT//2,\
                       self.WIDTH,self.HEIGHT))
```



Putting everything together:

```
In : class Paddle:
        WIDTH = 10
        HEIGHT = 80

        def __init__(self, y):
            self.y = y

        def show(self, colour):
            global screen
            pygame.draw.rect(screen, colour,\
                             pygame.Rect(WIDTH-self.WIDTH,\
                                  self.y-self.HEIGHT//2,\
                                  self.WIDTH,\
                                         self.HEIGHT))
```

We could now simply plot the paddle creating an object of the class, as we did before with the ball.

```
In : paddle = Paddle(HEIGHT//2) # we put it in the middle
     paddle.show(pygame.Color("white"))
```

We now have to move the paddle! We will need to revisit the `Ball` class to check collisions with the paddle.

We want to play the game with the mouse, and pygame provides that functionality![10].

If we look at the documentation, we can get the position of the mouse with `pygame.mouse.get_pos()`. This will provide a tuple (x,y) with the coordinates of the mouse. We simply need to use that to update the position of the paddle. Once again, we are going to create an `update` method that will hide the paddle, update its position, now based on the mouse position, and then show the paddle in its new position.

```
def update(self) :
     self.show(Color("black"))
    # check where the mouse is.
    self.y = pygame.mouse.get_pos()[1]
    self.show(Color("white"))
```

[10] https://www.pygame.org/docs/ref/mouse.html

As before, we need to include call the update method within the while loop to make it happen!

```
while True :
    e = pygame.event.poll()
    if e.type == pygame.QUIT :
        break
    ball.update()
    paddle.update()
    pygame.display.flip()
```

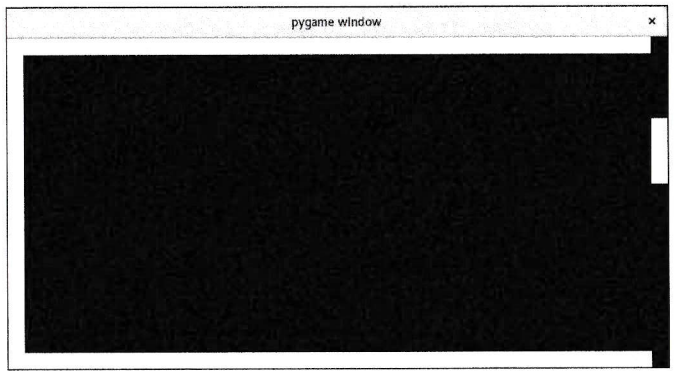
So our game so far is like this:

```
In : import pygame

    class Ball:

        RADIUS = 10

        def __init__(self,x,y,vx,vy):
            self.x=x
            self.y=y
            self.vx=vx
            self.vy=vy


        def show(self,colour):
            global screen
            pygame.draw.circle(screen, colour,\
                               (self.x,self.y), Ball.RADIUS)

        def update(self):

            newx= self.x+self.vx
            newy= self.y+self.vy

            if newx < BORDER+Ball.RADIUS:
                self.vx = -self.vx
            elif newy < BORDER+Ball.RADIUS or \
            newy > HEIGHT-BORDER-Ball.RADIUS:
                self.vy = -self.vy
            else:
                self.show(pygame.Color("black"))
                self.x= newx
                self.y= newy
                self.show(pygame.Color("white"))
```



```
class Paddle:

    WIDTH= 20
    HEIGHT= 80

    def __init__(self,y):
        self.y = y

    def show(self, colour):
        global screen
        pygame.draw.rect(screen, colour,\
                         pygame.Rect(WIDTH-self.WIDTH,\
                                     self.y-self.HEIGHT//2,\
                                     self.WIDTH,\
                                     self.HEIGHT))


    def update(self):
        self.show(pygame.Color("black"))
        self.y = pygame.mouse.get_pos()[1]
        self.show(pygame.Color("white"))

WIDTH = 800
HEIGHT = 400
BORDER = 20
VELOCITY = 4

pygame.init()

screen = pygame.display.set_mode((WIDTH,HEIGHT))

screen.fill(pygame.Color("black"))

pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,WIDTH,BORDER))
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,BORDER,HEIGHT))
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,HEIGHT-BORDER, WIDTH,\
                             BORDER))

ball = Ball(WIDTH-Ball.RADIUS, HEIGHT//2,\
            -VELOCITY,-VELOCITY)
ball.show(pygame.Color("white"))

paddle = Paddle(HEIGHT//2)
```

```
paddle.show(pygame.Color("white"))

while True:
    e = pygame.event.poll()
    if e.type == pygame.QUIT:
        break
    ball.update()
    paddle.update()
    pygame.display.flip()
pygame.quit()
```



So far, this works, but there are a few things that we need to fix. - The paddle clashes with the borders - The ball doesn't stop when hit by the paddle!

As we did before with the ball, we need to check that whenever we are going to update the position of the object, it doesn't collide with the borders. In this case, we only need to check with the top and bottom walls, and whenever the user moves further than those points, there won't be any update. However, we need to remember that the position we are storing (self.y) is the centre of the paddle.

```
def update(self) :
    newy = mouse.get_pos()[1]
    if newy-self.HEIGHT//2 >= BORDER \
       and newy+self.HEIGHT//2 <= HEIGHT-BORDER :
        self.show(pygame.Color("black"))
        self.y = newy
        self.show(pygame.Color("white"))
```


Something I didn't realise before is that the ball should be next to the paddle. So, I am going to move the original position of the ball slightly to the left to avoid initial collision with the paddle. Also, as we have now controlled the collision with the borders, I am going to set both initial velocities (`vx` and `vy`) with negative values to make the ball goes in a diagonal!

```
ball = Ball(WIDTH-Ball.RADIUS-Paddle.WIDTH,\
            HEIGHT//2, -VELOCITY,-VELOCITY)
ball.show(pygame.Color("white"))
```

We also have to check where the paddle is on the `Ball` class to prevent collisions. For simplicity, the (object) `paddle` will be used here as a global variable.

We need to add an extra condition that if the `newx` of the ball (+ its radius) is greater than the x position of the Paddle (remember it remained fixed at WIDTH-Paddle.WIDTH), and (at the same time) the `newy` makes it collide with the paddle, the velocity should be reversed, so that, we simulate that bouncing.

As we are using `paddle` as a global variable, `paddle.y` will give us the current centre of the paddle, whenever `newy` is above or below half the height of the paddle (Paddle.HEIGHT//2), this means they are colliding!

The easiest way to calculate that is to check if the absolute difference between the paddle centre and the ball is less than half of the paddle height:

```
...
elif newx+Ball.RADIUS >= WIDTH-Paddle.WIDTH \
     and abs(newy-paddle.y) <= Paddle.HEIGHT//2 :
         self.vx = - self.vx
...
```

So, the update method in `Ball` will now look like this:

```
class Ball:
   ....
   def update(self):
       # change the colour
```

```
        newx = self.x + self.vx
        newy = self.y + self.vy

        if newy+Ball.RADIUS > HEIGHT-BORDER \
            or newy-Ball.RADIUS < BORDER :
             self.vy = -self.vy
        elif newx-Ball.RADIUS < BORDER :
            self.vx = -self.vx
        elif newx+Ball.RADIUS >= WIDTH-Paddle.WIDTH \
             and abs(newy-paddle.y) <= Paddle.HEIGHT//2 :
             self.vx = - self.vx
             # REPEATED CODE: How do we solve it? :-)
        else:
             # change position
             self.show(pygame.Color("black"))
             self.x= newx
             self.y= newy
             self.show(pygame.Color("white"))
```

Something that annoys me a bit is that the ball starts moving immediately after the program runs, and I would rather wait for the user to click with the mouse (or the keyboard) and the the ball starts moving. How do we do this?

We could add an extra attribute `moving` to the ball that indicates with True or False if the Ball should move. The `update` method won't do anything if this variable is not set to True.

```
class Ball:

    RADIUS = 10

    def __init__(self,x,y,vx,vy) :
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.moving = False

    def update(self) :
        if not self.moving :
```



```
            return
        # .. rest of the code below
```

Then we need to include a new event type[11] (`MOUSEBUTTONDOWN`) to capture the click of the user.

Whenever the user clicks the mouse, the variable `moving` will be set to True:

```
while True :
    e = pygame.event.poll()
    if e.type == pygame.QUIT :
        break
    elif e.type == pygame.MOUSEBUTTONDOWN :
        ball.moving = True
    ball.update()
    paddle.update()
    pygame.display.flip()
```

Task: How would you use the mouse click to start and pause the game?

7.2.3 Adding Lives and Score display

What else can we do to make our game nicer?

We could count the number of successful hits to the ball, and also have a number of lives, so that, the game is over if we miss the ball a number of times.

To do this, you could have two global variables:

```
lives = 5
score = 0
```

On class `Ball`, we could modify the `update` method, so that, whenever we hit the ball, we increase that score. But if we miss it, we will lose one life. In addition, if we have lost, I would like to re-initialise the position of the Ball (waiting again for the click of the user); we could just simply call the **init** method!

[11] `https://www.pygame.org/docs/ref/event.html`

```
elif newx+Ball.RADIUS >= WIDTH-Paddle.WIDTH:
    if abs(newy-paddle.y)<=Paddle.HEIGHT//2:
        self.vx = -self.vx
        score += 1
    else:    # or we missed it!
        lives -= 1
        # reinitialise the position of the ball!
        self.show(pygame.Color("Black"))
        self.__init__(WIDTH-Ball.RADIUS-Paddle.WIDTH,\
                      HEIGHT//2,-VELOCITY,-VELOCITY)
        self.show(pygame.Color("White"))
```

We could then print the values of lives and points in the terminal, something like this:

```
while True:
    print ("lives ={}, points = {} "\
           .format(lives, score))
    if lives < 0: # game OVER!
        break

    e = pygame.event.poll()
    if e.type == pygame.QUIT:
        break
    elif e.type == pygame.MOUSEBUTTONDOWN :
        ball.moving = True

    ball.update()
    paddle.update()
    pygame.display.flip()
pygame.quit()
```

However, this is going to be very slow, and each iteration of the while loop is printing off a line. Why don't we try to do this nicer, and display those values on the game?

We need a function `show` that takes some text and print it on the screen. This is surprisingly difficult - here is the recipe:

1. We first need to set the font style that we will use, see `font`[12].

[12] https://www.pygame.org/docs/ref/font.html


```
pygame.font.SysFont(font.get_default_font(),15)
```

2. Render the String using `Font.render`[13].

This creates a new Surface with the specified text rendered on it. pygame provides no way to directly draw text on an existing Surface: instead you must use Font.render() to create an image (Surface) of the text, then blit this image onto another Surface.

3. Show that on the Surface using blit[14], i.e. draw an image onto another.

```
screen.blit(surf,(0,0)
```

```
In : def show(text):
        pygame.font.init()
        myFont = pygame.font.\
        SysFont(pygame.font.get_default_font(),25)
        surf = myFont.render(text, False,\
                             pygame.Color("black"),\
                             pygame.Color("white"))
        screen.blit(surf, (0,0))
```

With this, we could now call this function in every iteration:

```
while True:
    show("lives ={}, points = {} "\
         .format(lives, score))
    if lives < 0: # game OVER!
        break

    e = pygame.event.poll()
    if e.type == pygame.QUIT:
        break
    elif e.type == pygame.MOUSEBUTTONDOWN :
        ball.moving = True

    ball.update()
    paddle.update()
    pygame.display.flip()
pygame.quit()
```

[13] https://www.pygame.org/docs/ref/font.html#pygame.font.Font.render

[14] https://www.pygame.org/docs/ref/surface.html?highlight=blit#pygame.Surface.blit

7.2.4 Adding Sounds

Pygame also allows us to add sounds into the game with `pygame.mixer.Sound`[15].

We create an object of type mixer, that reads a music file from the drive:

```
In : pong = pygame.mixer.Sound("pong.wav")
```

Then, we could play the sound whenever the ball hits the paddle or the walls, by using:

```
pong.play()
```

I am going to let you decide where to put that line of code :-)

7.2.5 Adjusting the speed of the ball

If you run the same game on a Windows operating system, the ball will move very fast, and you want the same speed on all devices, you need to use a clock[16].

```
FRAMERATE = 50

clock = time.Clock()

while True:
    show("lives ={}, points = {} "\
         .format(lives, score))
    if lives < 0: # game OVER!
        break

    e = pygame.event.poll()
    if e.type == pygame.QUIT:
        break
    elif e.type == pygame.MOUSEBUTTONDOWN :
        ball.moving = not ball.moving
```

[15] https://www.pygame.org/docs/ref/mixer.html#pygame.mixer.Sound

[16] https://www.pygame.org/docs/ref/time.html#pygame.time.Clock


```
    ball.update()
    paddle.update()
    clock.tick(FRAMERATE)
    pygame.display.flip()
pygame.quit()
```

Why does it happen? Each iteration of the loop is called a frame. To ensure that the game runs the same way independently of your operating system and your computer, we added the line `clock.tick(FRAMERATE)` which is locking the frame rate to 50 frames per second. Thus, we have avoided the game to run at different speeds depending on the speed of the machine.

The final game will look like this:

```
In : import pygame

    class Ball:

        RADIUS = 10

        def __init__(self,x,y,vx,vy):
            self.x=x
            self.y=y
            self.vx=vx
            self.vy=vy
            self.moving = False


        def show(self,colour):
            global screen
            pygame.draw.circle(screen, colour,\
                               (self.x,self.y),\
                               Ball.RADIUS)

        def update(self):
            global lives, score

            if not self.moving:
                return

            newx= self.x+self.vx
            newy= self.y+self.vy
```

```
            if newx < BORDER+Ball.RADIUS:
                pong.play()
                self.vx = -self.vx
            elif newy < BORDER+Ball.RADIUS or \
            newy > HEIGHT-BORDER-Ball.RADIUS:
                pong.play()
                self.vy = -self.vy
            elif newx+Ball.RADIUS > WIDTH-Paddle.WIDTH:
                if abs(newy -paddle.y) <= Paddle.HEIGHT//2 :
                    score += 1
                    pong.play()
                    self.vx = -self.vx
                else: # we missed the ball
                    lives-=1
                    self.show(pygame.Color("black"))
                    self.__init__(WIDTH-Ball.RADIUS-\
                                  Paddle.WIDTH,
                                  HEIGHT//2,\
                                  -VELOCITY,-VELOCITY)
                    self.show(pygame.Color("white"))
            else:
                self.show(pygame.Color("black"))
                self.x= newx
                self.y= newy
                self.show(pygame.Color("white"))


    class Paddle:

        WIDTH= 20
        HEIGHT= 80

        def __init__(self,y):
            self.y = y

        def show(self, colour):
            global screen
            pygame.draw.rect(screen, colour,\
                             pygame.Rect(WIDTH-self.WIDTH,\
                                         self.y-self.HEIGHT//2,\
                                         self.WIDTH,\
                                         self.HEIGHT))


        def update(self):
            newy = pygame.mouse.get_pos()[1]

            if newy >= BORDER+self.HEIGHT//2\
```



```
                and newy <= HEIGHT- BORDER-self.HEIGHT//2 :
                    self.show(pygame.Color("black"))
                    self.y = newy
                    self.show(pygame.Color("white"))



WIDTH = 800
HEIGHT = 400
BORDER = 20
VELOCITY = 4

FRAMERATE = 50

lives = 5
score = 0


def show(text):
    pygame.font.init()
    myFont = pygame.font.\
    SysFont(pygame.font.get_default_font(),25)
    surf = myFont.render(text, False,\
                         pygame.Color("black"),\
                         pygame.Color("white"))
    screen.blit(surf, (0,0))


pygame.init()

screen = pygame.display.set_mode((WIDTH,HEIGHT))

screen.fill(pygame.Color("black"))

pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,WIDTH,BORDER))
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,0,BORDER,HEIGHT))
pygame.draw.rect(screen, pygame.Color("white"),\
                 pygame.Rect(0,HEIGHT-BORDER, WIDTH,\
                             BORDER))

ball = Ball(WIDTH-Ball.RADIUS-Paddle.WIDTH,\
            HEIGHT//2, -VELOCITY,-VELOCITY)
ball.show(pygame.Color("white"))

paddle = Paddle(HEIGHT//2)
```

```
    paddle.show(pygame.Color("white"))


    pong = pygame.mixer.Sound("pong.wav")

    clock = pygame.time.Clock()

    while True:
        show ("Lives {}, Score {}".format(lives,score))

        if lives <1:
            break

        e = pygame.event.poll()
        if e.type == pygame.QUIT:
            break
        elif e.type == pygame.MOUSEBUTTONDOWN:
            ball.moving = not ball.moving

        clock.tick(FRAMERATE)
        ball.update()
        paddle.update()
        pygame.display.flip()
    pygame.quit()
```

```
pygame 1.9.4
Hello from the pygame community.
https://www.pygame.org/contribute.html
```

This implementation of the game uses only very basic features of pygame. By using some more advanced classes, e.g. `sprites` we could have automatised the collision detection.

7.3 Project

Develop a simple game in a group of 3-4 people. The game should include a graphical interactive component and it is suggested that you use the pygame API. E.g. you may implement an arcade game like tetris or space invaders.

While it is a good idea to look at code samples please do not use other people's code but write your own! Please be explicit

about ideas/structures you have learned from other people's code.

Try to address the following goals:

1. Quality of game play
    - Is the game fun to play?
    - Does it look and sound good?
    - Is it free from errors?
    - Is it platform independent (i.e does it run on mac, windows and linux) ?
2. Software quality
    - Good use of Python (objects, functions etc)
    - Reusable code
    - Is the code well documented?
3. Other factors
    - Did the team work well together?
    - Are the instructions clear?
    - Did the demo go well?
    - Clear references to code used for inspiration
    - Good use of tools (e.g. git)