Pong

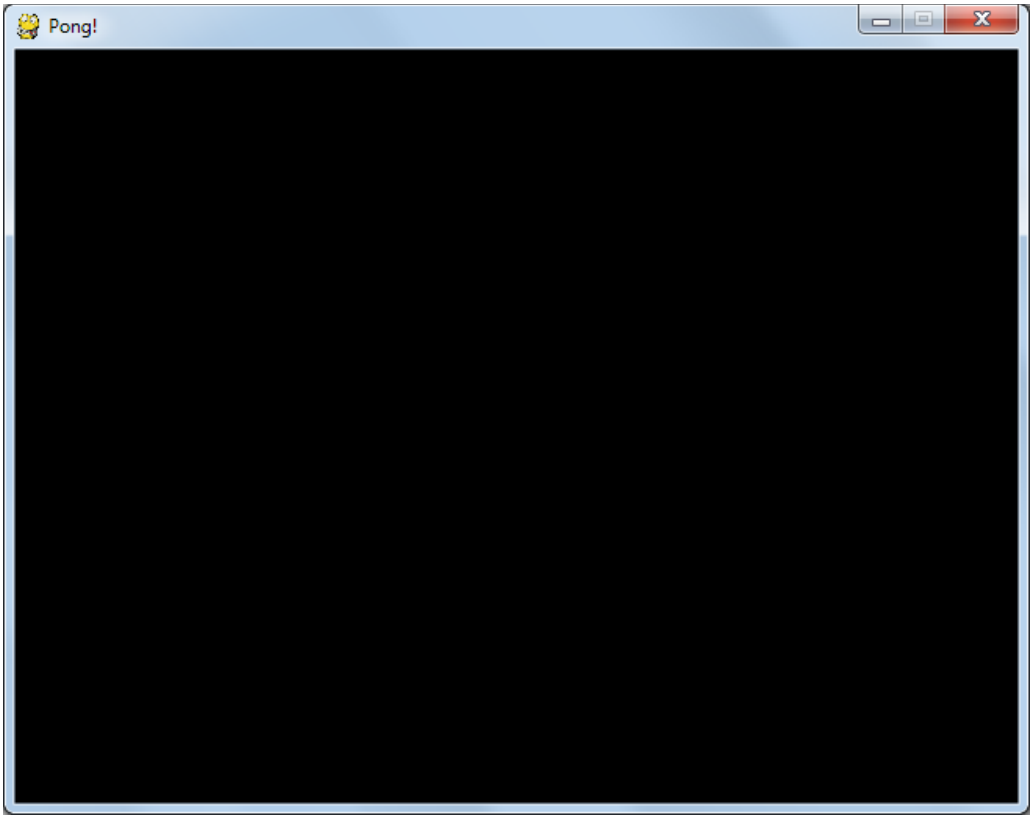
Use pygame to write a game of 'pong'. Pygame documentation and help can be found at http://www.pygame.org/docs/index.html though it isn't the easiest page to navigate and use. Looking back at the snake game may be more helpful if you are stuck.

1. Initialisation

Import and initialise the pygame module.

a. Import pygame
b. Import all the local variables from the module using 'from pygame.locals import *'
c. initialise using pygame.init()
d. Create the game screen using pygame.display.set_mode(), specifying a size of 640 x 480. Store it as a variable named 'screen'.
e. Use pygame.display.set_caption() to set the title to "Pong".

At this point you should be able to run your program, causing a game window to open. It will have "Pong" in the title bar but otherwise be blank.



2. Creating Paddles

Create shapes for the two paddles and the ball and draw them onto the display. The pygame.Surface() method creates a rectangular shape which can then be placed onto the screen. More complex shapes such as circles can be drawn onto this rectangle. For example pygame.Surface((10,50)) create a rectangle that is 10 pixels wide by 50 high. Note the double brackets since the function takes a tuple e.g. (10,50) as its argument.

a. Create a background 640 x 480 and two paddles 10 x 50, storing each of them as a variable

b. For each of them set the colour as an RGB value using the 'fill' method. For example if you called your background variable 'background' then use background.fill((255,255,255)) to colour it white. (Choose a different colour for the paddles so they show up and note again the double pair of brackets.)

Placing one object on top of another is done using the 'blit' method (short for BLock Image Transfer according to Wikipedia). So for example to place an object on the 'screen' variable created in step 1 at position (x,y) use screen.blit(<object>, (x,y))

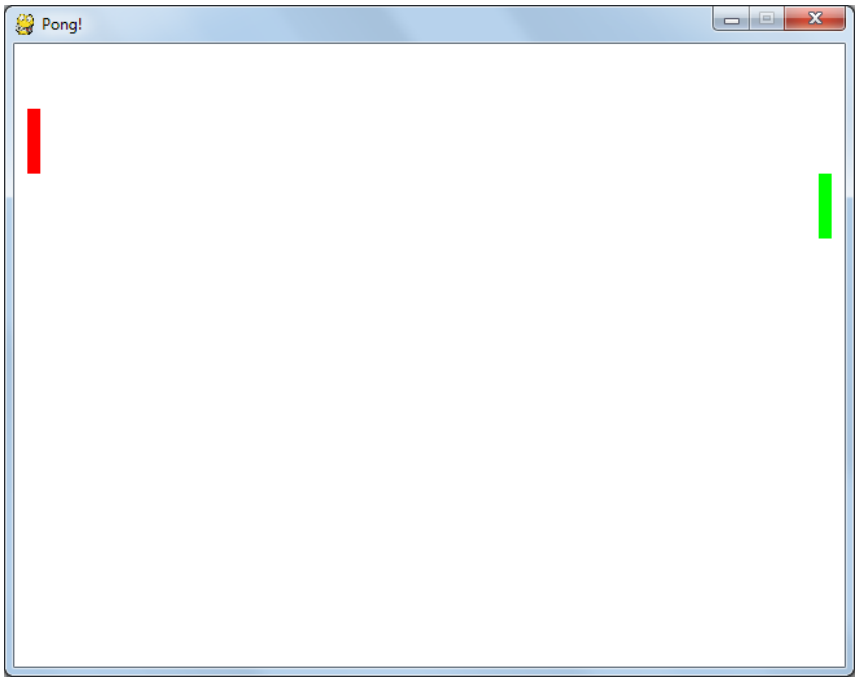
c. Use the blit method to place the background on the screen at position (0,0)

d. Place the first paddle at (10,50) and the second one at (620,100)

Pygame's display works by using two buffers – a technique known as 'double-buffering'. One is displayed to the screen while the other is modified. Once the modifications are complete the buffers are switched over and the new image is displayed. This avoids a jerky effect where the shapes appear one-by-one. This switch is performed using pygame.display.flip()

e. Call pygame.display.flip() to cause the modified buffer with the shapes drawn on it to be displayed.

At this point you should see something like this if you run your program:



3. Adding the ball

a. Create a 16 x 16 surface for the ball similar to those for the paddles.

b. Use the 'fill' method to set the colour to the same as the background.

c. Use pygame.draw.circle(<surface>,(R,G,B), (x,y), r) where RGB give the colour of the ball, 'x' and 'y' give the position of the centre - in this case (8,8) and 'r' gives the radius – in this case 8.

d. Use the 'blit' method to add the ball to the screen before flipping the display.

Try running your program to check that the ball can be seen.

4. Getting the ball rolling

The next step is to make the ball move.

a. Create a 'while' loop at the end of your program. For now use 'while True:' so it continues forever

b. Move all of the blit calls and the display flip inside the while loop.

c. Before the 'while' loop create variables 'ball_pos_x' and 'ball_pos_y' for the x and y co-ordinates of the ball. Use these when calling blit.

d. After flipping the display increase the x and y values by 1

When you run your program you should see the ball shoot off towards the bottom right at high speed. Now we'll make the ball bounce off the walls.

e. Before the 'while' loop create variables 'speed_x' and 'speed_y' for the x and y components of the ball's speed (how fast it is moving horizontally and vertically). Initialise them both to 1.

f. Modify the lines that increase the value of ball_pos_x and ball_pos_y by 1 to instead add speed_x and speed_y. i.e . ball_pos_x += speed_x

g. After this, check if ball_pos_x is outside the bounds of the screen and if so set speed_x = -speed_x so it starts travelling in the opposite direction. Do the same for y. Note that ball_pos_x and ball_pos_y give the position of the top left corner of the ball. The diameter is 16 pixels and you'll want to take that into account so it bounces off the bottom and right walls correctly.

When you run your program now the ball should perpetually bounce around the screen.

5. Slowing things down

Currently the only thing regulating the speed of the ball is the speed at which the computer can execute the instructions in the 'while' loop. The faster the computer the faster the ball. We need some way to regulate the speed so it is the same on all hardware and for that we need a clock.

a. Create a 'clock' variable using pygame.time.Clock()

b. At the end of the 'while' loop call clock.tick(30). This limits the game to a maximum of 30 frames per second (fps) which is sufficient to make it look smooth.

c. If you run your came now you'll find the ball moves much slower. Far too slowly in fact, so change the initial speed of ball from 1 to 5.

6. Adding some interactivity

Now we need to allow the user to control the game. Firstly we'll make it exit properly if you click on the X in the top right of the window.

a. Create a variable called 'game_over' and initialise it to False. Change the 'while' loop so that it stops if 'game_over' is set to True

b. Inside the 'while' loop create another loop to handle any events that the user has entered using 'for event in pygame.event.get():'

c. If event.type is QUIT then set 'game_over' to True

d. At the end of the program, outside the while loop, call pygame.quit()

Now when you run the program the window should close if you click the X in the top right.

7. Moving the paddles

Now to allow the user to control the paddle.

a. Create a variable called 'l_paddle_pos' to represent the (vertical) position of the left paddle. Use this when calling 'blit' to draw the paddle. (The horizontal position is fixed at 10.)

b. Create a variable called 'max_paddle_speed' to represent the speed at which the paddles move and initialise it to 5.

c. Create a variable 'l_paddle_speed' to represent the current speed of the left paddle and initialise it to 0. Within the 'while' loop after updating the position of the ball also update the position of the paddle using 'l_paddle_pos += l_paddle_speed'

d. Inside the 'for' loop where events are handled add 'if event.type == KEYDOWN:' to handle key presses. Within this check if event.key is ord('s') and if so set l_paddle_speed to max_paddle_speed. (i.e if the S button is pressed then start moving the paddle). If W is pressed then set the speed to -max_paddle_speed to move in the opposite direction

e. If event.type == KEYUP check if they key is S or W and set the speed back to zero

f. Now if you run the program you should be able to move the paddle with the W and S keys. Add checks to stop it from going off the top or bottom of the screen then repeat for the right paddle using the up and down keys (event.key == K_UP or K_DOWN).

8. Hitting the paddles

Instead of bouncing off the left and right walls we need to check whether the ball hits the paddle and only bounce off if it does

a. Instead of checking when the ball hits the edge of the screen check when it reaches the edge of the paddle. The paddle is 10 pixels wide and is inset by 10 pixels so adjust by 20 pixels.

b. Rather than always bounce we need to check whether the paddle is in the right place. The paddle is 50 pixels high. The ball is 8 pixels. Check if the Y coordinate of the top of the ball is between -8 and 50 pixels below the top of the paddle and only reverse the speed if this is the case.

Having done this for both of the paddles the ball should bounce correctly off them, but if it misses it will sail on until infinity and never come back.

c. If the ball goes off the screen to the left or right then switch the direction and move it back to the middle

9. Scores

Now to add the scores.

a. At the start of the program choose the font to use for the writing using the following. 'Calibri' is the name of the font and '40' is the font size.

```
font = pygame.font.SysFont("calibri",40)
```

b. Before the 'while' loop create variables for 'l_score' and 'r_score' and initialise them both to zero

c. Inside the 'while' loop create the graphic for the score using the following, where (R,G,B) is the colour of the writing. (The middle parameter enables anti-aliasing. Leave it set to True.) Do the same for the right score

```
l_score_str = font.render(str(l_score),  True, (R,G,B))
```

d. After the existing calls to blit (so the ball moves under the scores instead of over them) use blit to draw the score strings at the top of the screen.

e. If the ball goes off the screen on the left then increase 'r_score' by 1 and vice versa

At this point you should have a functional game which you can play with a friend.

10. Images

As well as simple shapes like rectangles and circles it is easy to use other images in games. To try this out replace ball with an image of your choice.

a. Find an image you want to use for the ball and save it in the same directory as your code, either as a .jpeg or .png file. E.g. ball.png

b. Find the place in the code where the ball was created. There should be 3 lines that look like this:

```
ball = pygame.Surface((16,16))
ball.fill((255,255,255))
pygame.draw.circle(ball, (0,0,255), (8,8), 8)
```

c. Remove the lines above and replace them with one like this.

```
ball = pygame.image.load('ball.png')
```

d. Try running the program. Unless your image happened to be very small you'll probably find the ball is massive. Solve this by adding the following line to scale the image to 16x16 pixels.

```
ball = pygame.transform.scale(ball, (16,16))
```

11. Tidying Up

As your program evolves and becomes more complex it is important to keep the code neat and tidy so you can read it and work out what it does when you go back to look at it. There are several conventions and techniques for doing this which you should use in all of your programs.

- Declare constants at the top of the file after the 'import' statements. Constants are any values that don't change throughout the program and by convention their names are all in capitals. For example:

```
SCREEN_WIDTH = 640
SCREEN_HEIGHT = 480
BALL_WIDTH = 16
RED = (255, 0, 0)
```

a. Find any values like these you have used within your program and replace the numbers with constants. For example:

```
screen = pygame.display.set_mode((SCREEEN_WIDTH, SCREEN_HEIGHT))
```

- Use functions to separate code that does a specific job from the rest of the script. Put the functions near the top of the file after the constants have been declared. For each function write a comment describing what it does.

b. Create a function to draw the game screen.

```
def draw_game_screen():
  """ Draw the paddles, ball and scores onto the screen. """
```

Move all the 'blit' calls in the while loop into this function and replace them with a call to this function. i.e.

```
draw_game_screen()
```

c. Create a function to update the position of the ball and move all the code that deals with moving the ball and checking its position into this function. Any global variables (i.e. those not created within this function) must be declared at the start of the function so python knows which one to use. For example:

```
def move_ball():
    """ Update the position of the ball and check if it has hit the paddle or
        the edge of the screen """
    global ball_pos_x, ball_pos_y
    global ball_speed_x, ball_speed_y
    global l_paddle_pos, l_paddle_speed
    global l_score, r_score
```

d. Create a function to handle all the user input and move the code that does this here.

Hopefully by now your while loop should be much tidier and look something like this:

```
while not game_over:
    handle_user_input()
    draw_game_screen()
    move_ball()

    pygame.display.flip()
    clock.tick(30)
```

12. Start Screen

Now that we've tidied up the while loop and put all the drawing code in separate functions we can easily add more game states. We'll create a splash screen to show the name of the game at the start.

a. Create a new variable called 'game_started' and initialise it to False.

b. Create a new function called 'draw_splash_screen'. This should be similar to the draw_game_screen function and draw the background, but instead of drawing the paddles, ball and scores it should have some text such as "Pong", "by <your name>" and "Press Any Key to Start" etc.

c. Change the code in the 'while' loop so that if game_started == False it calls the draw splash screen function, otherwise it draws the game screen and moves the ball. After this you should see the splash screen when you run your program. Try varying the fonts and sizes or add an image.

d. Now we just need to make the game start when you press a key. In the 'handle_user_input' function declare 'global game_started' and then set it to True when event.type == KEYDOWN.