Intro to Coding

Class 7

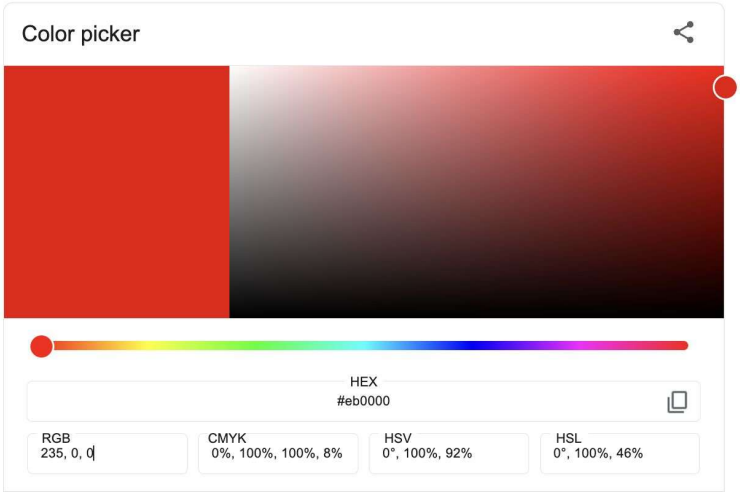
Outline

These are all the required concepts for Tic Tac Toe, sorted in ascending difficulty. They can also be covered in class 8 if there's not enough time

- Filling the screen
- Click detection (just knowing that a click occurred and the cursor's coordinates)
- Drawing shapes (lines and rectangles)
- Displaying text
- Detecting collisions

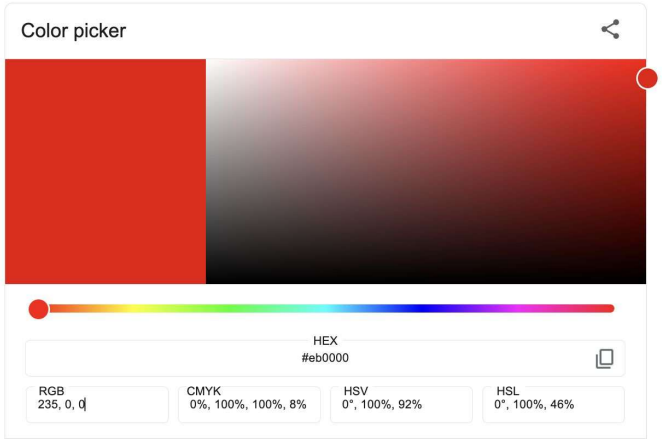
Filling the Screen

- Colours are displayed in the form (R, G, B)
  - R = red value, G = green value, B = blue value
  - The parameter for each color is an integer from 0-255 inclusive
  - The higher the number the more intense the color is
- An example color is (235, 0, 0)
  - (235, 0, 0) is red because the R value is large



Filling the Screen

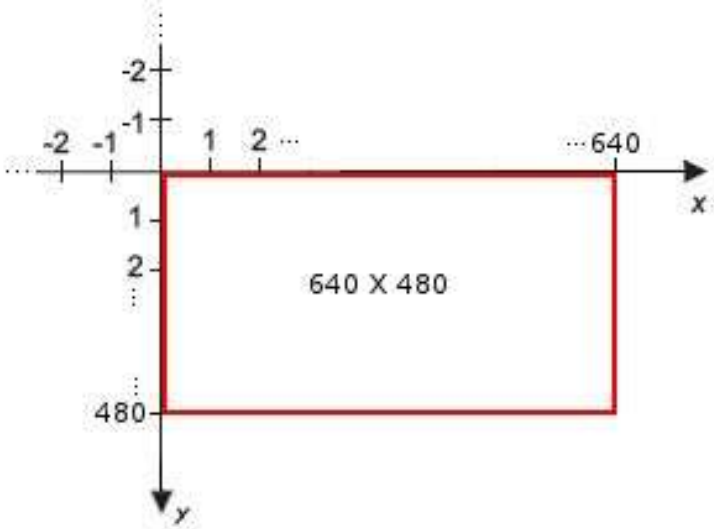
- First we pick a screen resolution, which is just how many pixels wide and tall the window is
- For each loop in the game, we fill the screen with the red color
  - This does not look like the “game” you may be used to, but it is the beginning of drawing shapes in our game window.




```python
import sys
import pygame

resolution = (800, 450)
color = (235, 0, 0)
screen = pygame.display.set_mode(resolution)

while True:
    screen.fill(color)
    pygame.display.flip()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
```

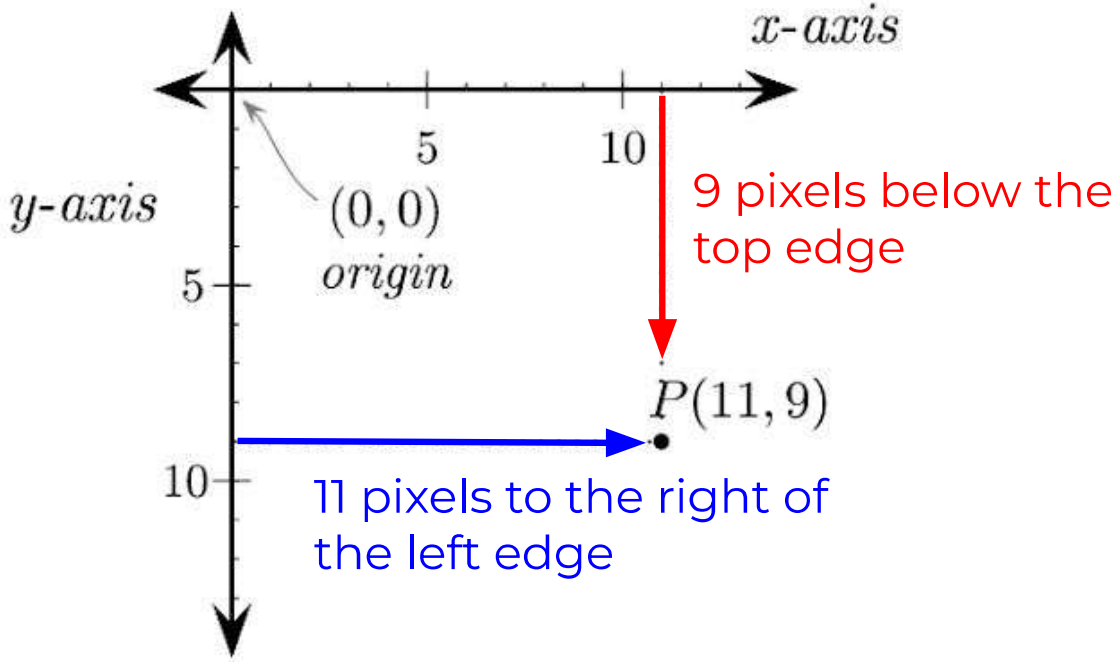
Pygame Coordinates

- The origin `(0,0)` of a Pygame window is on the top left of the window.
- Here is a visual representation of how a `640 x 480` Pygame window would look on a coordinate plane:
  - This plane is identical to a plane used in math, except the positive direction of the y-axis points **downward** instead of upward.



Pygame Coordinates

In Pygame, coordinates are specified in the format `(x, y)`, where `x` represents the x-value and `y` represents the y-value of the coordinate.



Detecting Mouse Input

When you click on the screen, Pygame registers this as an `event`, just like the `quit` event in the template.

To check mouse input, we ask if the most recent event is a mouse click, using:

`if event.type == pygame.MOUSEBUTTONDOWN():`

```
import pygame

pygame.init()
w = 640
h = 480
screen = pygame.display.set_mode((w, h))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            print("You just clicked!")
```

Detecting Mouse Input

When you click, Pygame will store the coordinates of your cursor in `event.pos`

Note that this is stored as a pair of two numbers; the x-coordinate and the y-coordinate

```
import pygame

pygame.init()
w = 640
h = 480
screen = pygame.display.set_mode((w, h))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            x, y = event.pos
```

Detecting Mouse Input

Here are some extra utilities:

`pygame.MOUSEBUTTONDOWN()` detects when you press down on the mouse button.

`pygame.MOUSEBUTTONUP()` detects when you lift the mouse button after clicking

`event.button` returns a value based on which button is pressed:

`event.button = 1` (left click)
`event.button = 2` (middle click)
`event.button = 3` (right click)

```
elif event.type == pygame.MOUSEBUTTONDOWN:
    if event.button == 1:
        # Left click
        print("Left Click")
    if event.button == 2:
        # Middle click (ie. clicking the scroll wheel)
        print("Middle Click")
    if event.button == 3:
        # Right click
        print("Right Click")
```

Activity

Write a program that initially sets the background to white. Each time the user clicks the screen, the background should change to the next colour of the rainbow. Once the user presses space on a purple screen, the program should close.

Here is the list of colours:

```
colours = [(255, 85, 85), (255, 125, 65), (255, 245, 85), (85, 205, 85), (85, 245, 255), (65, 55, 130), (100, 45, 105)]
```

Solution

```
import sys
import pygame

resolution = (800, 450)
screen = pygame.display.set_mode(resolution)
screen.fill((255, 255, 255))
pygame.display.flip()
i = -1
colours = [(255, 85, 85), (255, 125, 65), (255, 245, 85), (85, 205, 85), (85, 245, 255), (65, 55, 130), (100, 45, 105)]

while True and i < len(colours):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            i += 1
            screen.fill(colours[i])
            pygame.display.flip()
```

We start with `i = -1` so that we can increment `i` **before** accessing the next colour. If we increment `i` after, the program will close before the purple screen is displayed.

Drawing a Line

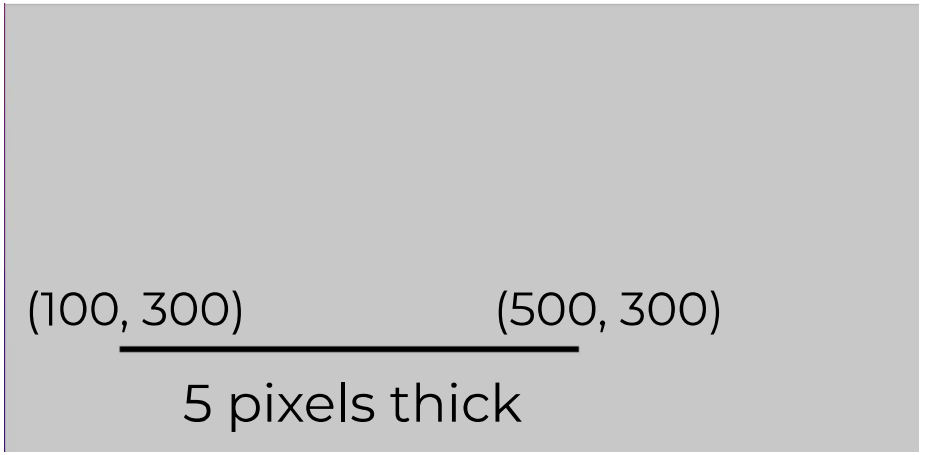
Syntax:

```
pygame.draw.line(surface, color, start_point, end_point, width)
```

- **Surface:** the surface on which the line will be drawn
- **Color:** the color of the line
- **Start_point:** the starting coordinate of the line you want to draw
- **End_point:** the end coordinate of the line you want to draw
- **Width:** the thickness of the line, in pixels

Drawing a Line - Example

```python
import pygame
import sys

pygame.init()

screen = pygame.display.set_mode((800, 400))
screen.fill((200, 200, 200))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    pygame.draw.line(screen, (0, 0, 0), [100, 300], [500, 300], 5)
    pygame.display.flip()
```



Drawing a Rectangle

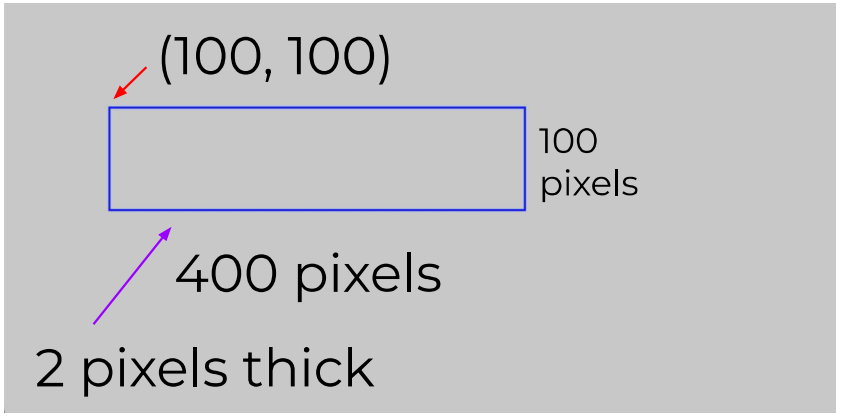
Syntax:
`pygame.draw.rect(surface, color, rectangle, *thickness)`

- **Surface:** the surface on which the rectangle will be drawn
- **Color:** the color of the rectangle, denoted in RGB format `(R, G, B)`
- **Rectangle:** denoted in square brackets with the following parameters: `[x, y, width, height]`
  - `x` and `y` refer to the coordinates of the **top-left corner** of the rectangle
- **Thickness:** an optional parameter, specifies how thick the border should be, in pixels.
  - If thickness is not specified, the rectangle is filled with the specified color

Drawing a Rectangle - Example

```python
import pygame
import sys

pygame.init()

screen = pygame.display.set_mode((800, 400))
screen.fill((200, 200, 200))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    pygame.draw.rect(screen, (0, 0, 255), [100, 100, 400, 100], 2)
    pygame.display.flip()
```

Initializing the surface before the while loop prevents screen flickering



Displaying Text (1)

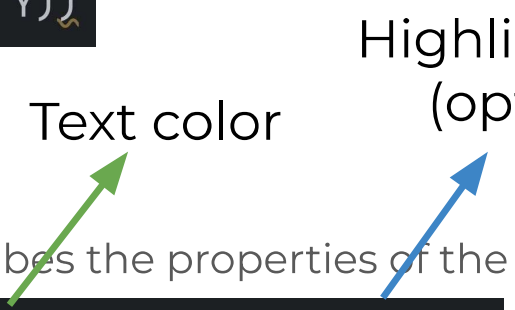
Create a display surface object using `display.set_mode()` function.

```
display_surface = pygame.display.set_mode((X, Y))
```

Create a Font object using `font.Font()` function.

```
font = pygame.font.Font('freesansbold.ttf', 32)
```

Create a Text object using `render()` function. This describes the properties of the text




```
text = font.render('CodeUp', True, (0, 255, 0), (0, 0, 128))
```

Create a rectangular object using `get_rect()` function. This allows the text to be displayed on the screen.

```
textRect = text.get_rect()
```

Displaying Text (2)

Set the position of the Rectangular object by setting the value of the center of Pygame rectangular object.

```
textRect.center = (200, 200)
```

Copying the Text surface object to the display surface object using `blit()` function.

```
display_surface.blit(text, textRect)
```

Show the display surface object on the Pygame window using the `display.update()` function.

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

Displaying Text: Exemplar

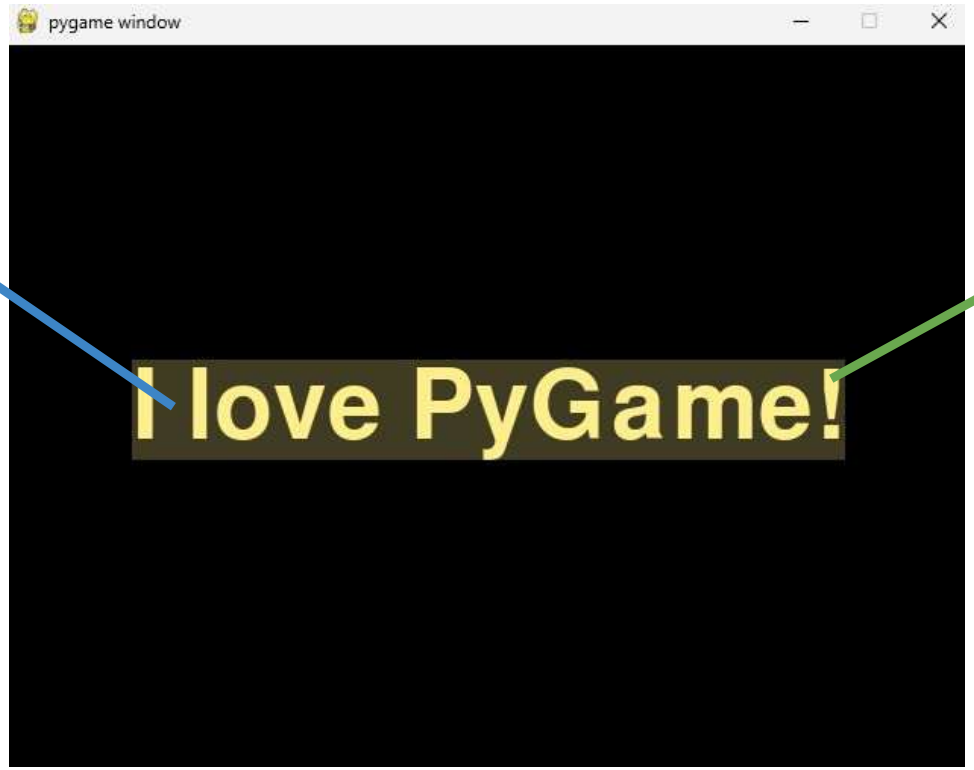
Displaying highlighted text: “I Love PyGame!”

```python
import pygame

def main():  1 usage
    pygame.init()
    display_surface = pygame.display.set_mode((640, 480))
    font = pygame.font.Font( name: 'freesansbold.ttf',  size: 64)

    text = font.render( text: "I love PyGame!",  antialias: True,  color: (255, 238, 140),  background: (63, 60, 35))
    textRect = text.get_rect()

    textRect.center = (320, 240)
    display_surface.blit(text, textRect)
    pygame.display.update()

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
    pygame.quit()

if __name__ == "__main__":
    main()
```

Displaying Text: Exemplar

Displaying highlighted “I Love PyGame!”

The text highlight



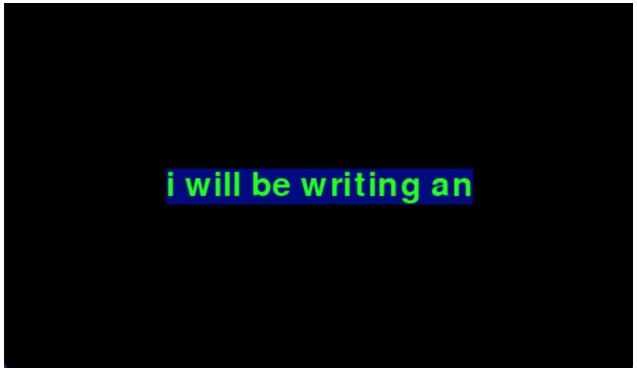
The text for this program is “I love PyGame!”

Activity

Create a program that will display a string on the Pygame window. The program should take in user input, and cut off any additional characters if the string exceeds 20 characters. The string should be displayed in the center of the screen. The font, colour, and highlights can be whatever, based on your preferences.

Note: make sure you use the `input()` function **before** the while loop

```
pygame 2.6.1 (SDL 2.28.4, Python 3.9.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter a sentence: i will be writing an extremely long sentence and it will get cut off
```



Solution

The program is basically the same, except you get the user to input a string, then use string splicing to limit the number of characters. Note that other factors may vary based on your preferences.

```
import sys
import pygame

pygame.font.init()

X = 600
Y = 350

display_surface = pygame.display.set_mode((X, Y))
font = pygame.font.Font("freesansbold.ttf", 32)
string = input("Enter a sentence: ")
text = font.render(string[0:20], True, (0, 255, 0), (0, 0, 128))
textRect = text.get_rect()
textRect.center = (300, 175)
display_surface.blit(text, textRect)
pygame.display.update()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
```

Detecting Collisions

- `collidelist()` is a method of the `pygame.Rect` object and is used for collision detection
- You pass in a **list** of `pygame.Rect` objects and it loops through the list **in order** and checks for overlap with the Rect it is called on
  - **Ex.**
    - `rect.collidelist(list_of_rects)`
    - This checks if any rect in `list_of_rects` overlaps with `rect`
- The method returns the **index** of the **first** overlapping rect from the list OR a **-1** if no rect overlaps

Detecting Collisions: Example

```
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 800))
rect_list = [(300, 300, 100, 100), (600, 600, 50, 70)]
pygame.draw.rect(screen, (255, 0, 0), rect_list[0])
pygame.draw.rect(screen, (0, 0, 255), rect_list[1])
pygame.display.flip()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            pos = pygame.mouse.get_pos()
            mouse = pygame.Rect(pos[0], pos[1], 1, 1)
            idx = mouse.collidelist(rect_list)
            if idx != -1:
                pygame.draw.rect(screen, (255, 255, 255), rect_list[idx])
                pygame.display.flip()
```