

# Pygame Basics

## Load and Launch Pygame:

```
import pygame
pygame.init()
```

## Display

`screen = pygame.display.set_mode((width, height))` Initializes and creates the window where your game will run, and returns a Surface, here assigned to the name "screen." Note: you're passing a tuple, hence the double parenthesis.

`pygame.display.update()` Redraws the main display surface if argument list is empty. Optionally, you can pass it a list of Rects, and it will just redraw the portions of the screen indicated in the list.

`pygame.display.get_surface()` Returns a reference to the Surface instantiated with the `set_mode()` function. Use this if you forget to assign `set_mode()` to a name.

## Surfaces, Images, and Transformations

Note: "Surface" is the name of the class, so you'd use the name you assigned when you created the surface. For example, if your main display Surface was called "screen" (as it is above), you'd use `screen.blit()`, not

`Surface.blit()`

`Surface.blit(sourceSurface, destinationRect, optionalSourceRect)` Copies pixels from one Surface to another. Used to draw images to the screen. If you omit the third argument, the entire source Surface is copied to the area of the destination Surface specified by the Rect in the second argument

`Surface.fill(color)` Fills surface with a solid color. Argument is a tuple of RGB values. e.g. (255,0,255) for Magenta (maximum red and blue, no green)

`Surface.convert()` Changes pixel format of the Surface's image to the format used by the main display. Makes things faster. Use it.

`Surface.convert_alpha()` Same as above, but when the Surface's image has alpha transparency values to deal with.

`Surface.get_rect()` Returns a Rect that will tell you the dimensions and location of the surface.

`pygame.image.load(filename)` Loads image from disk and returns a Surface. Note that in Python, directories are indicated by a *forward slash*, unlike Windows

`pygame.transform.rotate(Surface, angle)` Rotates Surface counterclockwise by degrees

`pygame.transform.scale(Surface, (width, height))` Resizes Surface to new resolution

## Rects

`Rect.move(x, y)` Returns a Rect moved *x* pixels horizontally and *y* pixels vertically

`Rect.move_ip(x, y)` Moves the Rect *x* pixels horizontally and *y* pixels vertically

Assignable attributes (in most cases, a tuple of *x* and *y* values):

*top, left, bottom, right, topleft, bottomleft, topright, bottomright, midtop, midleft, midbottom, midright, center, centerx, centery, size, width, height*

## Time

`pygame.time.Clock()` Creates a Clock object (assign this to a name), which you can then call the `tick()` method on to find out how much time has passed since the last time you called `tick()`

`pygame.time.delay(milliseconds)` Pauses game for time specified

`pygame.time.get_ticks()` Returns the number of milliseconds passed since `pygame.init()` was called

## Joystick

```
my_joystick = pygame.joystick.Joystick(0)
```

```
my_joystick.init()
```

## Events

`pygame.event.get()` Call once per frame to get a list of events that occurred since the last time `pygame.event.get()` was called. Events can have the following type values, with associated attributes:

QUIT                   none

KEYDOWN               unicode, key, mod (if you import `pygame.locals`, compare to e.g. `K_a` for "a")

KEYUP                  key, mod

MOUSEMOTION           pos, rel, buttons

MOUSEBUTTONUP pos, button  
MOUSEBUTTONDOWN pos, button  
JOYAXISMOTION joy, axis, value

## Fonts

`f = pygame.font.Font(None, 32)` Creates a font object of size 32 using the default font. If you know where the .TTF file of the font you want to use is located, you can use the filename as the first argument  
`surf = f.render("Hello", 1, (255,0,255), (255,255,0))` Creates a surface of rendered text using the font of the font object. The first argument is the text itself, the second is whether the text is anti-aliased or not (0 for no), the third argument is a 3-item tuple that defines the RGB values of the color of the text, and the fourth (optional) argument is a 3-item tuple that defines the RGB values of the color of the background. If the fourth argument is not specified, the background will be transparent. This command creates a surface that has the word Hello in magenta on a yellow background, which can then be blitted to the screen like any surface. It's quite ugly.

## Audio

The default values for the sound channels are 22KHz frequency, 16-bit(signed), stereo sound with a 1K buffer. If you wish to change this, call `pygame.mixer.pre_init()`, BEFORE you call `pygame.init()`  
`pygame.mixer.pre_init(frequency, size, stereo, buffer)`  
size is negative if signed, stereo is boolean, buffer must be a power of 2

`kaboom = pygame.mixer.Sound(filename)` must be an uncompressed WAV or OGG  
`kaboom.play(loops=0, maxtime=0)`  
`kaboom.stop()`

For music, you do not create objects, since you can only have one music track running at any time. Music is streamed, never fully loaded at once. You can use MIDI files.

`pygame.mixer.music.load(filename)`  
`pygame.mixer.music.play(loops=0)` set loops to number of times to repeat after first run-through, -1 to repeat indefinitely  
`pygame.mixer.music.stop()`

## Sprites, Groups, and Collision Detection

```
class Monster(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load("monster.png")
        self.rect = self.image.get_rect()
    ...
```

```
monsters = pygame.sprite.RenderPlain((monster1, monster2, monster3))
monsters.update()
monsters.draw()
```

```
Rect.contains(Rect): return True or False
Rect.collidepoint(x, y): return True or False
Rect.colliderect(Rect): return True or False
Rect.collidelist(list): return index
```

```
pygame.sprite.spritecollide(sprite, group, dokill):
    return Sprite_list
```

```
pygame.sprite.groupcollide(group1, group2, dokill1,
    dokill2): return Sprite_dict
```