

# Beginner's Python Cheat Sheet - Pygame

## What is Pygame?

Pygame is a framework for making games using Python. Making games is fun, and it's a great way to expand your programming skills and knowledge. Pygame takes care of many of the lower-level tasks in building games, which lets you focus on the aspects of your game that make it interesting.

## Installing Pygame

*Pygame runs on all systems, but setup is slightly different on each OS. The instructions here assume you're using Python 3, and provide a minimal installation of Pygame. If these instructions don't work for your system, see the more detailed notes at <http://ehmatthes.github.io/pcc/>.*

### Pygame on Linux

```
$ sudo apt-get install python3-dev mercurial  
  libsdl-image1.2-dev libsdl2-dev  
  libsdl-ttf2.0-dev  
$ pip install --user  
  hg+http://bitbucket.org/pygame/pygame
```

### Pygame on OS X

*This assumes you've used Homebrew to install Python 3.*

```
$ brew install hg sdl sdl_image sdl_ttf  
$ pip install --user  
  hg+http://bitbucket.org/pygame/pygame
```

### Pygame on Windows

*Find an installer at <https://bitbucket.org/pygame/pygame/downloads/> or <http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame> that matches your version of Python. Run the installer file if it's a .exe or .msi file. If it's a .whl file, use pip to install Pygame:*

```
> python -m pip install --user  
  pygame-1.9.2a0-cp35-none-win32.whl
```

### Testing your installation

*To test your installation, open a terminal session and try to import Pygame. If you don't get any error messages, your installation was successful.*

```
$ python  
>>> import pygame  
>>>
```

## Starting a game

*The following code sets up an empty game window, and starts an event loop and a loop that continually refreshes the screen.*

### An empty game window

```
import sys  
import pygame as pg  
  
def run_game():  
    # Initialize and set up screen.  
    pg.init()  
    screen = pg.display.set_mode((1200, 800))  
    pg.display.set_caption("Alien Invasion")  
  
    # Start main loop.  
    while True:  
        # Start event loop.  
        for event in pg.event.get():  
            if event.type == pg.QUIT:  
                sys.exit()  
  
        # Refresh screen.  
        pg.display.flip()  
  
run_game()
```

### Setting a custom window size

*The display.set\_mode() function accepts a tuple that defines the screen size.*

```
screen_dim = (1200, 800)  
screen = pg.display.set_mode(screen_dim)
```

### Setting a custom background color

*Colors are defined as a tuple of red, green, and blue values. Each value ranges from 0-255.*

```
bg_color = (230, 230, 230)  
screen.fill(bg_color)
```

## Pygame rect objects

*Many objects in a game can be treated as simple rectangles, rather than their actual shape. This simplifies code without noticeably affecting game play. Pygame has a rect object that makes it easy to work with game objects.*

### Getting the screen rect object

*We already have a screen object; we can easily access the rect object associated with the screen.*

```
screen_rect = screen.get_rect()
```

### Finding the center of the screen

*Rect objects have a center attribute which stores the center point.*

```
screen_center = screen_rect.center
```

## Pygame rect objects (cont.)

### Useful rect attributes

*Once you have a rect object, there are a number of attributes that are useful when positioning objects and detecting relative positions of objects. (You can find more attributes in the Pygame documentation.)*

```
# Individual x and y values:  
screen_rect.left, screen_rect.right  
screen_rect.top, screen_rect.bottom  
screen_rect.centerx, screen_rect.centery  
screen_rect.width, screen_rect.height
```

```
# Tuples  
screen_rect.center  
screen_rect.size
```

### Creating a rect object

*You can create a rect object from scratch. For example a small rect object that's filled in can represent a bullet in a game. The Rect() class takes the coordinates of the upper left corner, and the width and height of the rect. The draw.rect() function takes a screen object, a color, and a rect. This function fills the given rect with the given color.*

```
bullet_rect = pg.Rect(100, 100, 3, 15)  
color = (100, 100, 100)  
pg.draw.rect(screen, color, bullet_rect)
```

## Working with images

*Many objects in a game are images that are moved around the screen. It's easiest to use bitmap (.bmp) image files, but you can also configure your system to work with jpg, png, and gif files as well.*

### Loading an image

```
ship = pg.image.load('images/ship.bmp')
```

### Getting the rect object from an image

```
ship_rect = ship.get_rect()
```

### Positioning an image

*With rects, it's easy to position an image wherever you want on the screen, or in relation to another object. The following code positions a ship object at the bottom center of the screen.*

```
ship_rect.midbottom = screen_rect.midbottom
```

## Python Crash Course

*Covers Python 3 and Python 2*

[nostarchpress.com/pythoncrashcourse](http://nostarchpress.com/pythoncrashcourse)



## Working with images (cont.)

### Drawing an image to the screen

Once an image is loaded and positioned, you can draw it to the screen with the `blit()` method. The `blit()` method acts on the screen object, and takes the image object and image rect as arguments.

```
# Draw ship to screen.  
screen.blit(ship, ship_rect)
```

### The `blitme()` method

Game objects such as ships are often written as classes. Then a `blitme()` method is usually defined, which draws the object to the screen.

```
def blitme(self):  
    """Draw ship at current location."""  
    self.screen.blit(self.image, self.rect)
```

## Responding to keyboard input

Pygame watches for events such as key presses and mouse actions. You can detect any event you care about in the event loop, and respond with any action that's appropriate for your game.

### Responding to key presses

Pygame's main event loop registers a `KEYDOWN` event any time a key is pressed. When this happens, you can check for specific keys.

```
for event in pg.event.get():  
    if event.type == pg.KEYDOWN:  
        if event.key == pg.K_RIGHT:  
            ship_rect.x += 1  
        elif event.key == pg.K_LEFT:  
            ship_rect.x -= 1  
        elif event.key == pg.K_SPACE:  
            ship.fire_bullet()  
        elif event.key == pg.K_q:  
            sys.exit()
```

### Responding to released keys

When the user releases a key, a `KEYUP` event is triggered.

```
if event.type == pg.KEYUP:  
    if event.key == pg.K_RIGHT:  
        ship.moving_right = False
```

## Pygame documentation

The Pygame documentation is really helpful when building your own games. The home page for the Pygame project is at <http://pygame.org/>, and the home page for the documentation is at <http://pygame.org/docs/>.

The most useful part of the documentation are the pages about specific parts of Pygame, such as the `Rect()` class and the `sprite` module. You can find a list of these elements at the top of the help pages.

## Responding to mouse events

Pygame's event loop registers an event any time the mouse moves, or a mouse button is pressed or released.

### Responding to the mouse button

```
for event in pg.event.get():  
    if event.type == pg.MOUSEBUTTONDOWN:  
        ship.fire_bullet()
```

### Finding the mouse position

The mouse position is returned as a tuple.

```
mouse_pos = pg.mouse.get_pos()
```

### Clicking a button

You might want to know if the cursor is over an object such as a button. The `rect.collidepoint()` method returns true when a point is inside a rect object.

```
if button_rect.collidepoint(mouse_pos):  
    start_game()
```

### Hiding the mouse

```
pg.mouse.set_visible(False)
```

## Pygame groups

Pygame has a `Group` class which makes working with a group of similar objects easier. A group is like a list, with some extra functionality that's helpful when building games.

### Making and filling a group

An object that will be placed in a group must inherit from `Sprite`.

```
from pygame.sprite import Sprite, Group
```

```
def Bullet(Sprite):  
    ...  
    def draw_bullet(self):  
        ...  
    def update(self):  
        ...
```

```
bullets = Group()
```

```
new_bullet = Bullet()  
bullets.add(new_bullet)
```

### Looping through the items in a group

The `sprites()` method returns all the members of a group.

```
for bullet in bullets.sprites():  
    bullet.draw_bullet()
```

### Calling `update()` on a group

Calling `update()` on a group automatically calls `update()` on each member of the group.

```
bullets.update()
```

## Pygame groups (cont.)

### Removing an item from a group

It's important to delete elements that will never appear again in the game, so you don't waste memory and resources.

```
bullets.remove(bullet)
```

## Detecting collisions

You can detect when a single object collides with any member of a group. You can also detect when any member of one group collides with a member of another group.

### Collisions between a single object and a group

The `sprite.collideany()` function takes an object and a group, and returns True if the object overlaps with any member of the group.

```
if pg.sprite.spritecollideany(ship, aliens):  
    ships_left -= 1
```

### Collisions between two groups

The `sprite.groupcollide()` function takes two groups, and two booleans. The function returns a dictionary containing information about the members that have collided. The booleans tell Pygame whether to delete the members of either group that have collided.

```
collisions = pg.sprite.groupcollide(  
    bullets, aliens, True, True)
```

```
score += len(collisions) * alien_point_value
```

## Rendering text

You can use text for a variety of purposes in a game. For example you can share information with players, and you can display a score.

### Displaying a message

The following code defines a message, then a color for the text and the background color for the message. A font is defined using the default system font, with a font size of 48. The `font.render()` function is used to create an image of the message, and we get the rect object associated with the image. We then center the image on the screen and display it.

```
msg = "Play again?"  
msg_color = (100, 100, 100)  
bg_color = (230, 230, 230)
```

```
f = pg.font.SysFont(None, 48)  
msg_image = f.render(msg, True, msg_color,  
    bg_color)  
msg_image_rect = msg_image.get_rect()  
msg_image_rect.center = screen_rect.center  
screen.blit(msg_image, msg_image_rect)
```

More cheat sheets available at

[ehmatthes.github.io/pcc/](https://github.com/ehmatthes/pcc/)