Frequently Asked Questions Pygame

1.1 What is the minimum amount of code needed to make a Pygame window appear?
The following snippet of code will create a window 800 pixels wide by 600 pixels tall. Note that the `set_mode` function requires width and height be a tuple.

```
import pygame
pygame.init()
WIDTH = 800
HEIGHT = 600
screen = pygame.display.set_mode( (WIDTH, HEIGHT) )
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # flip the display updates user screen with images
    pygame.display.flip()
pygame.quit()
```

1.2 I want to make a full game project, what is a good main.py file startup?

```
import pygame
# these constants may be stored in a setting.py module
WIDTH = 800
HEIGHT = 600
FPS = 60
WHITE = (255, 255, 255)
###
class Application:
    def __init__(self):
        # create the screen
        self.screen = pygame.display.set_mode( (WIDTH, HEIGHT) )
        # create a group for all sprites
        self.sprites = pygame.sprite.Group()
        # create a clock object to manage frame rate and ticks
        self.clock = pygame.time.Clock()
        # a bool value to end the game loop
        self.running = True



    def gameloop(self):
```

```
        while self.running:
            # maintain FPS rate and measure delta time
            self.dt = self.clock.tick(FPS) / 1000
            # handle all user events since last frame
            self.update_events()
            # update sprite logic and positions
            self.sprites.update()
            # update the screen
            self.update_screen()

    def update_screen(self):
        # fill with background image or color
        self.screen.fill(WHITE)
        # draw sprites on screen
        self.sprites.draw(self.screen)
        # flip screen from previous frame to new frame
        pygame.display.flip()

    def update_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            # additional inputs can be handled here.


def main():
    pygame.init()
    app = Application()
    app.gameloop()
    pygame.quit()

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

2.1 How do I draw a rectangle on my screen?
Use `pygame.draw.rect` function.

```
pygame.draw.rect(surface, color, rect, width = 0)
```

surface is surface you want to draw on. If you are drawing on your screen, then this is your screen variable.

`color` is a tuple represeing RGB values. This is often a constant you have made.

`rect` is either a Rect object of a tuple of 4 values (x, y, width, height)

The optional argument `width` defaults to 0. When 0, the shape is filled in. Otherwise it is hollow.

There are several ways to use this function.

Example: draw a filled red rectangle on screen at (x,y) = (100, 200), width = 50, height = 25

```
RED = (255, 0, 0)
pygame.draw.rect(screen, RED, (100, 200, 50, 25))
```

Example: draw a hollow blue rect at x= 500, y = 400, width =100, height = 20 with edge width size of 1

```
BLUE = (0, 0, 255)
pygame.draw.rect(screen, BLUE, (500, 400, 100, 20), width = 1)
```

Example: create a Rect object, then draw that object

```
GREEN = (0, 255, 0)
my_rect = pygame.Rect(150, 550, 40, 40)
pygame.draw.rect(screen, GREEN, my_rect)
```

2.2 How do I draw a polygon on my screen?

```
pygame.draw.rect(surface, color, rect, width = 0)
```

surface is surface you want to draw on.
color is a tuple represeing RGB values.
points is a list or tuple of points which are in tuple form.
width `= 0` is default, making the polygon filled.

Example

```
GREEN = (0, 255, 0)
points = [(50, 100), (50, 200), (100, 150)]
```

```
pygame.draw.polygon(screen, GREEN, points) # filled
pygame.draw.polygon(screen, RED, points, width = 2) # hollow
```

The above example will make a green filled triangle that is outlined in red.

2.3 How do I draw a circle or an ellipse on my screen?
A circle can be drawin with either the circle or ellipse function.

```
pygame.draw.circle(surface, color, center, radius, width = 0)
```
surface is surface you want to draw on.
color is a tuple represeing RGB values.
center is a tuple for (x, y) or a vector.
radius is the radius of the circle.
width = 0 (optional) is used for line thickness. width = 0 is filled; width > 0 is hollow.

```
pygame.draw.ellipse(surface, color, rect, width = 0)
```
surface is surface you want to draw on.
color is a tuple represeing RGB values.
rect is either a Rect object or a tuple (x, y, width, height) for a bounding rectangle
width = 0 (optional) is used for line thickness. width = 0 is filled; width > 0 is hollow.

Examples

```
RED = (255, 0, 0)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
# draw a blue circle at (400, 300) with radius 50
pygame.draw.circle(screen, BLUE, (400, 300), 50)
# draw a hollow circle with red outline
pygame.draw.circle(screen, RED, (100, 400), 25, width = 1)

# creating  Rect object at (550, 200) with width = height = 75
rect1 = pygame.Rect(550, 200, 75, 75)
# drawing a circle that inscribes rect1
pygame.draw.ellipse(screen, GREEN, rect1)

# creating a Rect that is 100 pixels long but only 50 high
rect2 = pygame.Rect(200, 40, 100, 50)
# drawing the ellipse bound by the rect2 object.
pygame.draw.ellipse(screen, BLUE, rect2)
```

2.4 How do I draw a line on my screen?
Use the line function.

pygame.draw.line(surface, color, start, end, width = 1)
surface is surface you want to draw on.
color is a tuple represeing RGB values.
start is a tuple for (x, y) for one endpoint of the line
end is a tupe for the other endpoint of the line.
width = 1 (optional) is used for line thickness. width = 1 is default.

```
pygame.draw.line(screen, BLUE, (100, 200), (600, 50))
pygame.draw.line(screen, GREEN, (200, 300), (400, 400), width = 3)
```

2.5 How do I draw shapes to surface that is not my screen?
Shapes can be drawn onto any surface object in pygame.

You can create your own instances of the `Surface` class or, since an image is also a surface, load a surface with `pygame.image.load`.

In this example, we create an image object by making a surface, then drawing to the surface a circle.

```
RED = (255, 0, 0)

# create a screen surface
screen = pygame.display.set_mode( (WIDTH, HEIGHT) )

# create an image surface
image = pygame.Surface( (100, 100) )

# calculate center of our image surface
x, y = image.get_width() / 2, image.get_height() / 2

# draw red circle in center of the image
pygame.draw.circle(image, RED, (x, y), 20)

# draw the circle image on the screen.
screen.blit(image, (400, 300))
```

2.6 I drew a shape on a surface, now how can I make the outside of my shape transparent?
Use the `set_colorkey(`*`color`*`)` method that belongs to a Surface object to set the color you want to be transparent.

In this example, we make a red circle on an image surface and then make the part of the surface that is outside of the circle transparent (instead of the black color it is by default).

```
RED = (255, 0, 0)
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)

# create a screen
screen = pygame.display.set_mode((800, 600))
# make the screen a white background
screen.fill(WHITE)

# create a surface that will perfectly fit a circle with radius 50
radius = 50
image = pygame.Surface( (radius * 2, radius * 2) )
x, y = radius, radius
pygame.draw.circle(image, RED, (x, y), radius)

# make BLACK transparent
image.set_colorkey(BLACK)

# draw the circle image on the screen.
screen.blit(image, (400, 300))
```

2.7 How do I load an image in pygame?
Use the `pgyame.image.load(`*`image_file`*`)` function. Remember that you must also use *`image`*`.convert()` or *`image`*`.convert_alpha()` to convert the pixel format of the image to match the pixel format of the screen you have created.
The following little program assumes there is a PNG image called `game_guy.png` in the working directory.

```
import pygame
pygame.init()
WIDTH = 800
```

```
HEIGHT = 600
screen = pygame.display.set_mode( (WIDTH, HEIGHT) )
image = pygame.image.load('game_guy.png') # load image
image = image.convert_alpha() # convert pixel format to match screen
screen.blit(image, (400, 300)) # paint image at location (400, 300)
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # flip the display
    pygame.display.flip()
pygame.quit()
```

2.8 What does convert or convert_alpha do? Do I really need to use them?
Images are a big rectangular arrangement of RGB color and Alpha transparency values. The format of these pixels must be converted to match the pixel format of the user's screen.

If you do not convert the image to match the pixel format of the screen then every frame where the image is drawn the conversion must occur as its drawn. This is very slow and will slow down frame rate. Therefore, pygame recommends converting it once and for all and storing the converted images back on itself.

Example

```
image = pygame.image.load('game_guy.png') # load image
image = image.convert_alpha() # convert pixel format to match screen
```

The above two lines of code are often combined to single line:

```
image = pygame.image.load('game_guy.png').convert_alpha()
```

Using convert_alpha preserves the transparency properties of the PNG image.

2.9 Help! I have drawn a shape or blitted an image but I do not see it on my screen!
The most common cause of this problem is not calling the function `pygame.dispaly.flip` or `pygame.display.update`

The function `pygame.display.flip` will update the entire screen surface. The function `pygame.dispaly.update` allows you supply a list a `Rect` objects specifying the specific rectangle areas you wish to update. Since flip always updates the entire screen, it is typically a slower function.

2.10 How do I scale an image to a certain size?

Suppose you have an image called `img` which you want scaled to a certain `width` and `height`.

```
img = pygame.transform.scale(img, (width, height))
```

3.1 How to I create a Sprite I can use in my game?

A sprite is an instance of a class that subclasses the built in pygame.sprite.Sprite class.

The following is an example of a very simply subclass of Sprite called Circle_Sprite.

```
class Circle_Sprite(pygame.sprite.Sprite):  # subclass
    def __init__(self, x, y):   # an init that takes x, y position
        super().__init__()
        self.image = pygame.Surface( (50, 50) )     # an image
        pygame.draw.circle(self.image, BLUE, (25, 25), 25)
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()           # a rect
        self.rect.x = x
        self.rect.y = y

    def update(self):
        self.rect.x += 1
```

All subclasses of Sprite should have a `self.image` and `self.rect` property. Usually, you will also want to include an `update` method as well.

3.2 How do I use a Sprite in a pygame program?

Sprites are best used in a `pygame.sprite.Group` or one of the other Group types.

The following code snippit is part of a program which creates an instance of the `Circle_Sprite` class from FAQ 3.1.

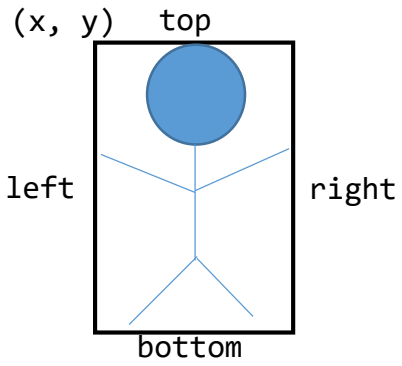
```
# … some imports, screen, and color constants …
screen = pygame.display.set_mode( (WIDTH, HEIGHT) )
sprites = pygame.sprite.Group()
sprite1 = Circle_Sprite(100, 200)  # create a Circle_Sprite object
sprites.add(sprite1)               # add sprite to our group
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # flip the display
    screen.fill(WHITE)
    sprites.update()  # calls update method for each sprite in group
    sprites.draw(screen)  # see note below
    pygame.display.flip()
pygame.quit()
```

A note on the draw method that belongs to Group objects:
`sprites.draw(screen)` works exactly the same as follows:

```
for sprite in sprites:
    screen.blit(sprite.image, sprite.rect)
```

So as you can see, each sprite's image in the group Sprites is blitted onto the screen using the sprites rect object for position. Therefore, each sprite you desire to draw on the screen should have a `self.image` and a `self.rect`.

3.3 How do I move a sprite directly from a main application?
Sprite locations can be changed by changing any of their rect location properties. The following is a summary of some of the many rect location properties.



In addition to these, you may find centerx and centery useful for locating the center of the sprite.

Consider the following example using an instance of a Mouse class that subclasses pygame.sprite.Sprite.

```
mouse = Mouse(30, 100)       # creates a Mouse object at (30, 100)
mouse.rect.x += 10               # moves mouse 10 units right
mouse.rect.y -= 5                # moves mouse 5 units up
```

3.4 How do I move a sprite through its update method?

Suppose a mouse sprite has been created and added to a sprite group called `all_sprites`. Then, when `all_sprites.update()` is called before each frame draw, the mouse sprite's update method will be called each frame.

In the `Mouse` class consider adding the following update method:

```
def update(self):
    self.rect.x += 2
    if self.rect.right > WIDTH:
        self.rect.right = WIDTH
```

In this example update method, each `Mouse` sprite created by the main application will move right 2 units a frame until the right edge starts to extend off the right side of the screen. At which point, we lock the right side to the width of the screen.

3.5 How do I detect collisions between sprites?

There are many different functions you can use to detect collisions between sprites. Which one is best depends on your game and the situation.

`pygame.sprite.spritecollide(sprite, group, dokill)`
This function takes a sprite you want to test, a group you want to test collisions of the sprite with, and a value of `True` if you want any sprite in the group collided with to be removed from the game or `False` otherwise. This method returns a list of all sprites in `group` that `sprite` is colliding with.

(Advanced) A fourth optional collide function argument can be used to customize how the collisions are detected.

pygame.sprite.collide_rect(sprite1, sprite2)
This function takes two sprites as arguments and returns True if the rect objects from sprite1 and sprite2 are overlapping.

`pygame.sprite.groupcollide(group1, group2, dokill1, dokill2)`
This function returns a dictionary of with every sprite from `group1` listed as a key that is mapped to a list of each sprite from `group2` it collides with. `dokill1` and `dokill2` can be `True` or `False` whether you want the colliding sprites removed upon collision detection.

This function also allows for an optional collide function as a fourth parameter to further customize collision detection.

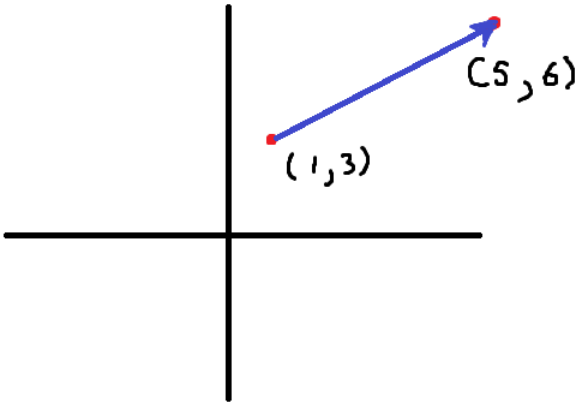
Consider the following game where a Sprite called explorer attempts to pick up coins for points. Coin sprites are saved in a group called coins_group. Since it is possible to get more than one coin at a time, the coder has opted to use the sprite_collide function.

```
# get a list of all coins collided with, remove coins from game
coins = pygame.sprite.spritecollide(explorer, coins_group, True)

# increase score by the length of the coins collided with list
explorer.score += len(coins)
```

4.1 What is a vector?
A 2D vector is a set of instructions for how to move from one place to another.

Imagine you are at position (1, 3) and your lunch is at (5, 6). A vector tells you how far you must move and in which direction to reach your lunch.



The vector needed to travel from (1, 3) to (5, 6) can be described in two manners.

Method 1 Component Form

$$v =< 5-1, 6-3 >$$
$$v =< 4, 3 >$$

Method 2 Magnitude and Direction

Distance = $\sqrt{(5-1)^2 + (6-3)^2} = 5$

Direction = $\tan(\theta) = \frac{3}{4} \rightarrow \theta = 36.9°$

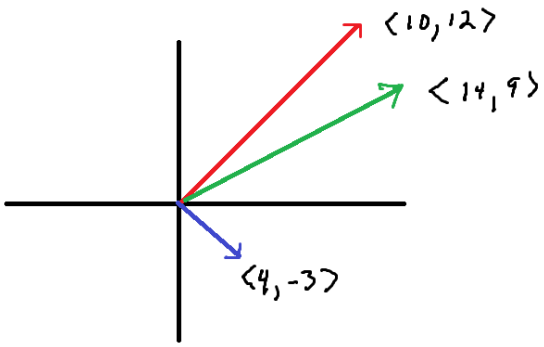
Each method has advantages. While the component form tells how much to move right and how much to move up and down and is easier to perform math operations with, the magnitude and direction tells exactly how far to move and in which direction.

4.2 How do you add or subtract two vectors?

Suppose throw a ball on a path represented by the vector $b =< 10, 12 >$. While in the air, however, the force of the wind affects it blowing along a path represented by the vector $w =< 4, -3 >$. What is the result of the throw?

To find the resulting path vector of the ball, simply add

$$b + w =< 10, 12 > +< 4, -3 > =< 14, 9 >$$



4.3 How do I create a vector in Pygame?
A vector is an instance of the pygame.math.Vector2 class.

To create a vector pass the constructor the x, y components of your vector.

```
v = pygame.math.Vector2(4, 5)
```

You can also create a copy of a vector from another vector.

```
w = pygame.math.Vector2(v)
```

4.4 What are some of the math operations I can do with vectors in pygame?

You can add and subtract vectors. You can also multiply a vector by a real number.

```
v = pygame.math.Vector2(4, 5)
w = pygame.math.Vector2(9, -1)

print(v + w)
print(v - w)
print(3 * v)
```

Output:

```
[13, 4]
[-5, 6]
[12, 15]
```

We can also use the shortcut `+=` and `-=` operators as well.

4.5 How do I get the x and y component of a vector in Pygame?

Simply access the x and y property as follows:

```
v = pygame.math.Vector2(3, 4)
print(v.x, v.y)
```

Output:

```
3.0 4.0
```

4.6 How do I find the magnitude (length) of vectors in Pygame?
Magnitude can be found with the magnitude or length methods.

```
v = pygame.math.Vector2(3, 4)

magnitude_v = v.magnitude()
length_v = v.length()     # alias for magnitude
print(magnitude_v, length_v)  # both 5.0
```

Note: Since the magnitude method uses the Pythagorean theorem it requires taking a square root. Square roots are very slow in computer science. Therefor, Pygame does not recommend using this method inside your game loop. It is better to use `magnitude_squared` or `length_squared` methods.

4.7 How do I find the direction angle of vectors in Pygame?

There is no specific method for the direction angle of a vector. Instead Pygame supplies us with the angle_to method to measure angle of rotation to a given vector.

vec1`.angle_to(`*vec2*`)` returns the degrees that vec1 must be rotated to match *vec2*. If *vec1* is on the positive x-axis then this angle is the direction angle for *vec2*.

```
v = pygame.math.Vector2(3, 4)
x_vec = pygame.math.Vector2(1, 0) # a normal vector on x-axis
angle = x_vec.angle_to(v)
print('direction angle of v', angle)
```

Remember that the direction of a vector is measured from the positive x-axis in a clockwise-rotation. (In math class it is a counter-clockwise, but is backward in pygame because the positive y-axis goes downward in pygame.)

4.8 How do I use vectors to control the postion, velocity, and acceleration of a sprite projectile (like a tossed ball)?

Sir Isaac Newton developed his laws of motion many years ago. The laws of motion give us very useful equations for predicting the location of an object at different moments in time.

Assume that acceleration constant over a time interval ΔT – read 'delta t'. Δ means "change in"

$\Delta V = A * \Delta T$ The change in a sprites velocity is its acceleration times change in time.
$\Delta P = \frac{1}{2} * A * \Delta T^2 + V * \Delta T$ The change in position is dependent on acceleration and velocity.

To use these formulas we simply supply our sprites with an an acceleration, velocity, and position vector.

The ΔT should be the change in time between frames. We get this using the `tick` method from a `pygame.time.Clock` object.

The following program application demonstrates how to use position, velocity, and acceleration vectors to model the motion of a ball tossed at a 45 degree angle upwards (-45 degrees) with velocity of 50 pixels per millisecond. The ball is under constant downward acceleration of 20 pixels per millisecond squared due to gravity.

```
import pygame
WIDTH = 600
HEIGHT = 600

GRAVITY = 20

BLUE = (0, 0, 255)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

class Circle_Sprite(pygame.sprite.Sprite):
    def __init__(self, app, x, y, speed, angle):
        super().__init__()
        self.game = app
        self.image = pygame.Surface( (50, 50) )
        pygame.draw.circle(self.image, BLUE, (25, 25), 25)
        self.image.set_colorkey(BLACK)
        self.rect = self.image.get_rect()
        self.pos = pygame.math.Vector2(x, y)
        self.vel = pygame.math.Vector2(speed, 0)
        self.vel = self.vel.rotate(angle)
```

```
        self.acc = pygame.math.Vector2(0, GRAVITY)
        self.rect.center = self.pos

    def update(self):
        dt = self.game.dt
        self.vel += self.acc * dt
        self.pos += 0.5 * self.acc * dt ** 2 + self.vel * dt
        self.rect.center = self.pos

class Application:
    def __init__(self):
        self.screen = pygame.display.set_mode( (WIDTH, HEIGHT) )
        self.sprites = pygame.sprite.Group()
        circle = Circle_Sprite(self, 100, 200, 50, -45)
        self.sprites.add(circle)
        self.running = True
        self.clock = pygame.time.Clock()

    def gameloop(self):
        self.dt = self.clock.tick(60) / 1000
        while self.running:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
            # flip the display
            self.screen.fill(WHITE)
            self.sprites.update()
            self.sprites.draw(self.screen)
            pygame.display.flip()

def main():
    pygame.init()
    app = Application()
    app.gameloop()
    pygame.quit()

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

5.1 How do I detect input from a users keyboard?
There are a couple of popular ways to do this.

Method 1:
You can track KEYDOWN and KEYUP events that are fired when the user presses and releases a key. Consider the following example update_events method in an application class. This method is called once per frame in the gameloop.

```
    def update_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    print('up')
                elif event.key == pygame.K_DOWN:
                    print('down')
                elif event.key == pygame.K_LEFT:
                    print('left')
                elif event.key == pygame.K_RIGHT:
                    print('right')
```

Advanced:
You can make a call to the function `pygame.key.set_repeat(`*`interval`*`)` to force pygame to consider a held down key to be repeatedly pressed every *`interval`* of milliseconds.

Method 2:
You can get a dictionary of all key states during a sprite's update.

Consider the following update method in a Sprite class. Recall that update method will be called once per frame for all sprites in a group if `group.update()` is called.

```
    def update(self):
        # keys will be a dictionary of key : bool.
        # for example if up is pressed then keys[pygame.K_UP] == True
        keys = pygame.key.get_pressed()
        if keys[pygame.K_UP]:
            self.rect.y -= 1
        if keys[pygame.K_DOWN]:
            self.rect.y += 1
        if keys[pygame.K_LEFT]:
            self.rect.x -= 1
        if keys[pygame.K_RIGHT]:
            self.rect.x += 1
```