Philipps Universität Marburg

Advancing in Python and Pygame

Thursday, September 1 · 9:30 – 10:30am

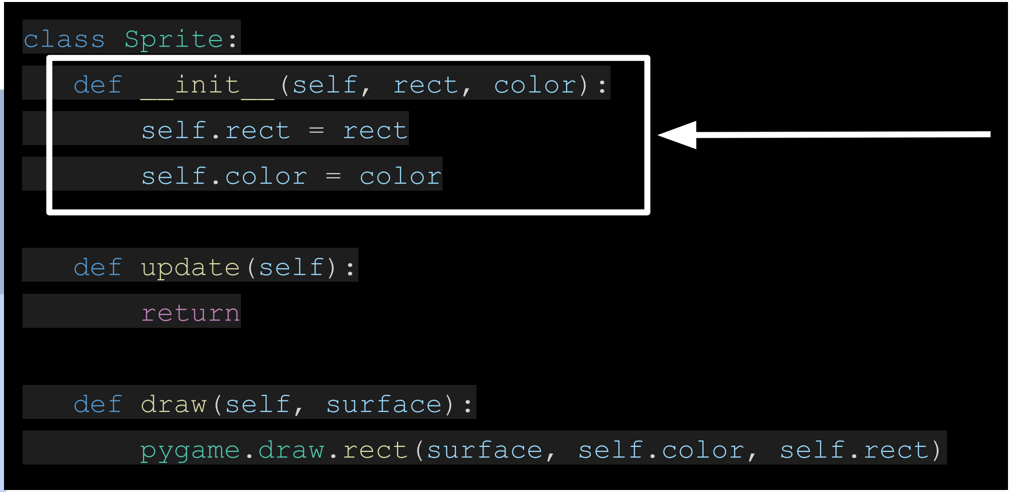
Part 1

Class inheritance

Making a Sprite base class

- What do the Player and Pepper class have in common?
  - Rect
  - Color
  - Update
  - Draw
- We can simplify the code even more by creating a parent class that Player and Pepper inherit from.

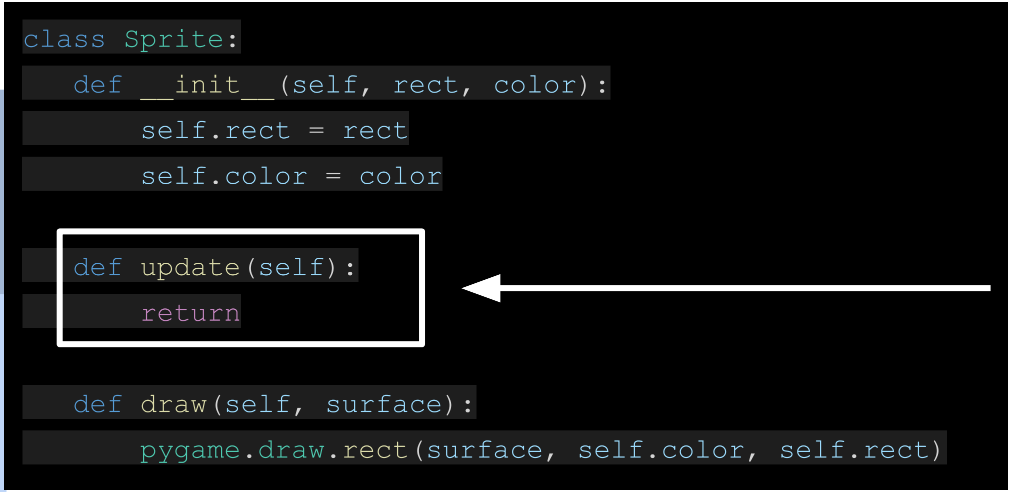
Step 1: Move commonalities to Sprite class



```
class Sprite:
    def __init__(self, rect, color):
        self.rect = rect
        self.color = color

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```

Constructor: Both Pepper and Player have a rect and a color. The rects and colors can be different, so we make them parameters that we can decide on when we create our objects.

Step 1: Move commonalities to Sprite class

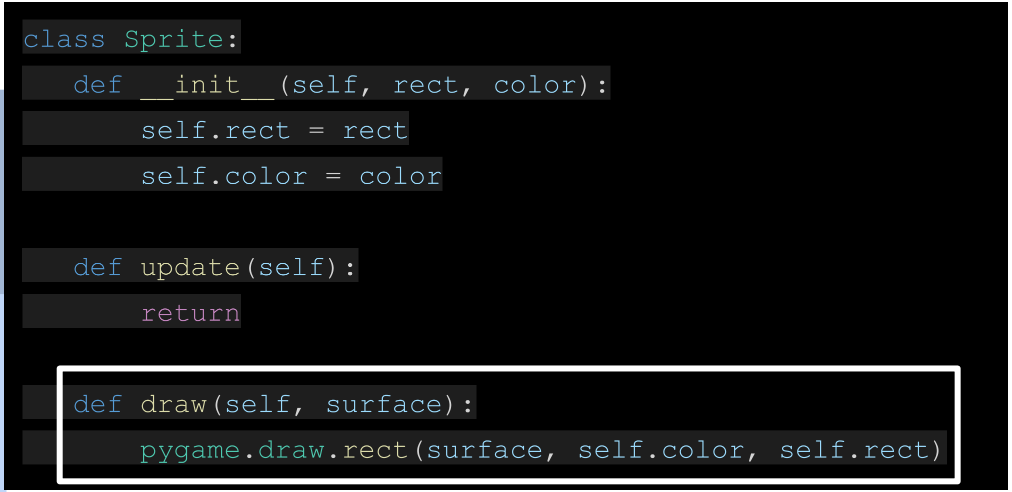


```python
class Sprite:
    def __init__(self, rect, color):
        self.rect = rect
        self.color = color

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```

Constructor: Pass in the desired rect and color when we create a Pepper or Player object.

Update method: Our sprites are updated in different ways, so we can leave this unimplemented and *override* it in the Pepper and Player classes

Step 1: Move commonalities to Sprite class



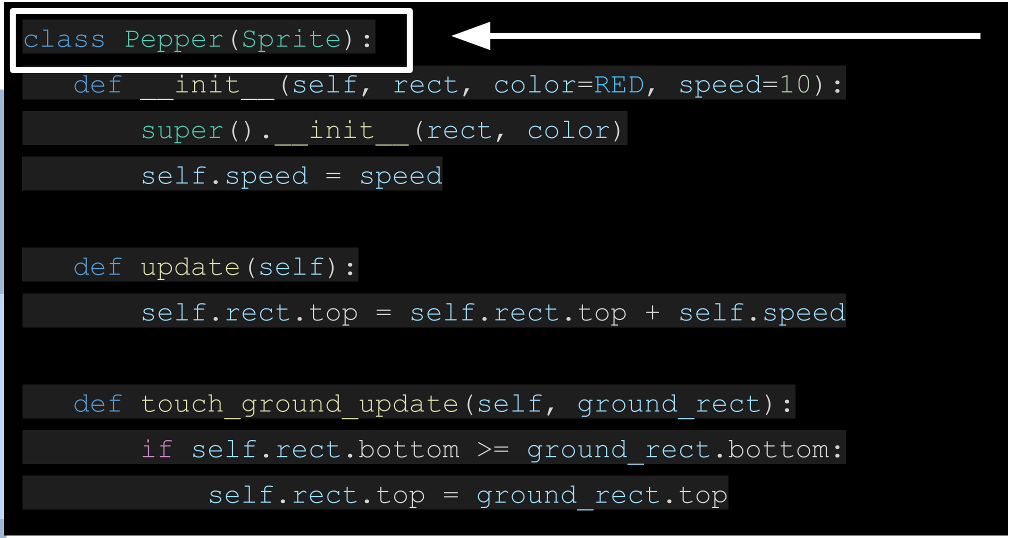
```
class Sprite:
    def __init__(self, rect, color):
        self.rect = rect
        self.color = color

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```

Constructor: Pass in the desired rect and color when we create a Pepper or Player object.

Update method: We will implement this individually for the child classes.

Draw method: In our game, the sprites are drawn the same way. So we can put the implementation in the base class and remove it from Pepper and Player.

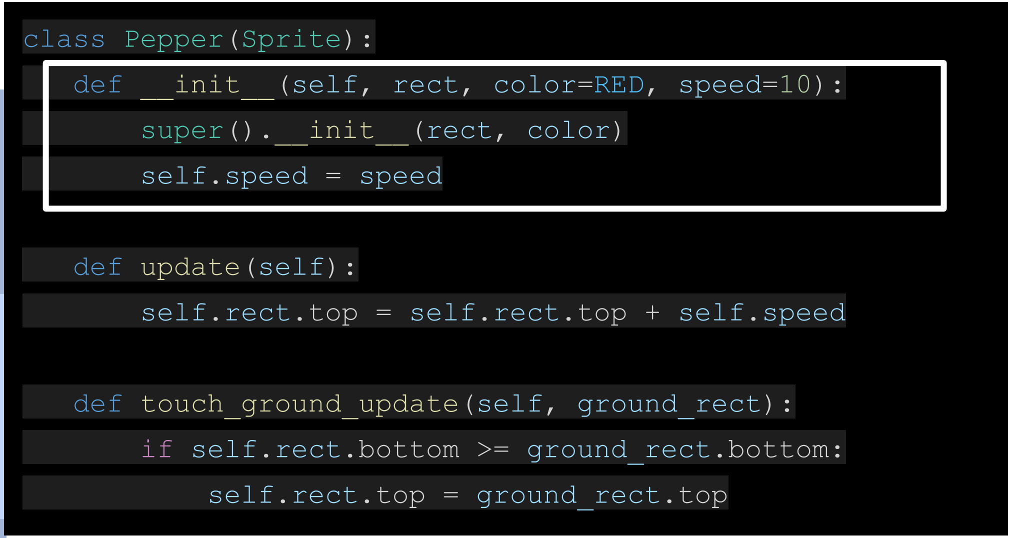
Step 2: Fix child classes



```python
class Pepper(Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def update(self):
        self.rect.top = self.rect.top + self.speed

    def touch_ground_update(self, ground_rect):
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top
```

Inherit Parent: We define the Pepper class like so. Then the Pepper inherits all the attributes and methods from the Sprite class.

Step 2: Fix child classes

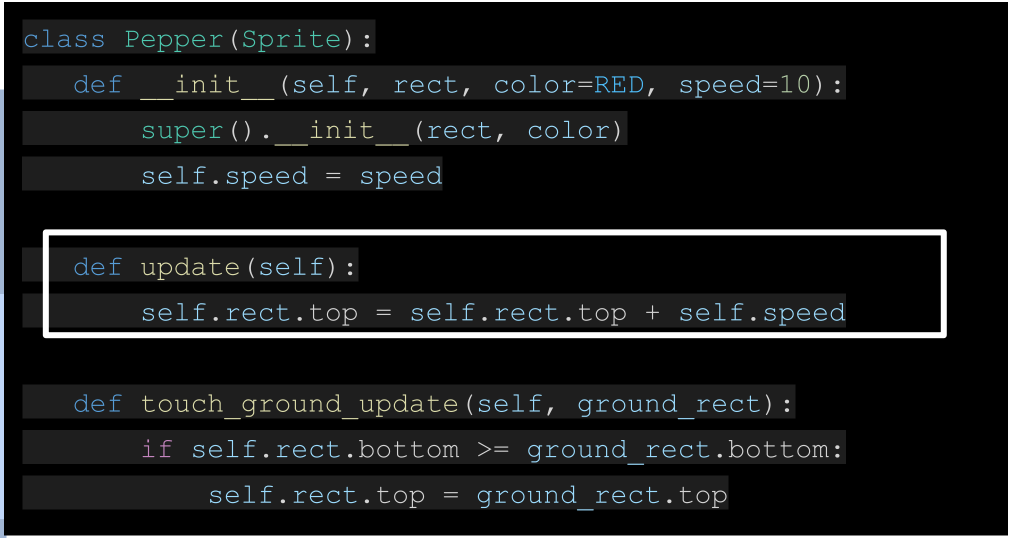


```
class Pepper(Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def update(self):
        self.rect.top = self.rect.top + self.speed

    def touch_ground_update(self, ground_rect):
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top
```

Inherit Parent: Pepper inherits from Sprite class.

Constructor: We can call super().__init__() which will use the constructor from the parent Sprite class. We pass in our desired values for rect and color. We also added a speed parameter. We make default values for the color and speed, so we can change them if we want to. But in general our peppers will be red and fall at the same speed.

Step 2: Fix child classes



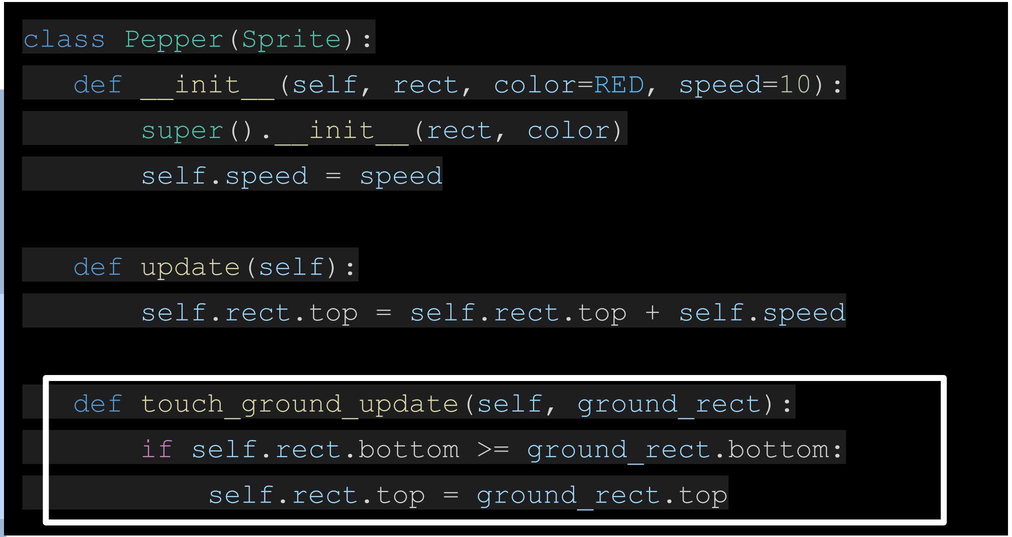
```python
class Pepper(Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def update(self):
        self.rect.top = self.rect.top + self.speed

    def touch_ground_update(self, ground_rect):
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top
```

Inherit Parent: Pepper inherits from Sprite class.

Constructor: Call super().__init__() to use the parent constructor. Add parameters if the child class has additional attributes to set.

Update method: Stays the same as before.

Step 2: Fix child classes



```python
class Pepper(Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def update(self):
        self.rect.top = self.rect.top + self.speed

    def touch_ground_update(self, ground_rect):
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top
```

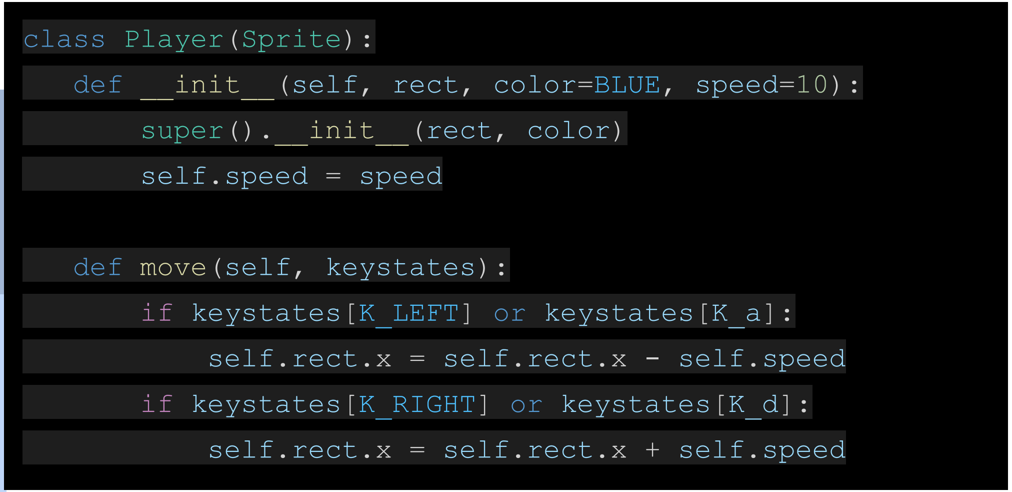
Inherit Parent: Pepper inherits from Sprite class.

Constructor: Call super().__init__() to use the parent constructor. Add parameters if the child class has additional attributes to set.

Update method: Stays the same as before.

Additional methods: The falling behavior is unique to the Pepper class, so we leave it in the class.

Step 2: Fix child classes



```
class Player(Sprite):
    def __init__(self, rect, color=BLUE, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def move(self, keystates):
        if keystates[K_LEFT] or keystates[K_a]:
            self.rect.x = self.rect.x - self.speed
        if keystates[K_RIGHT] or keystates[K_d]:
            self.rect.x = self.rect.x + self.speed
```

Do the same with the Player class…

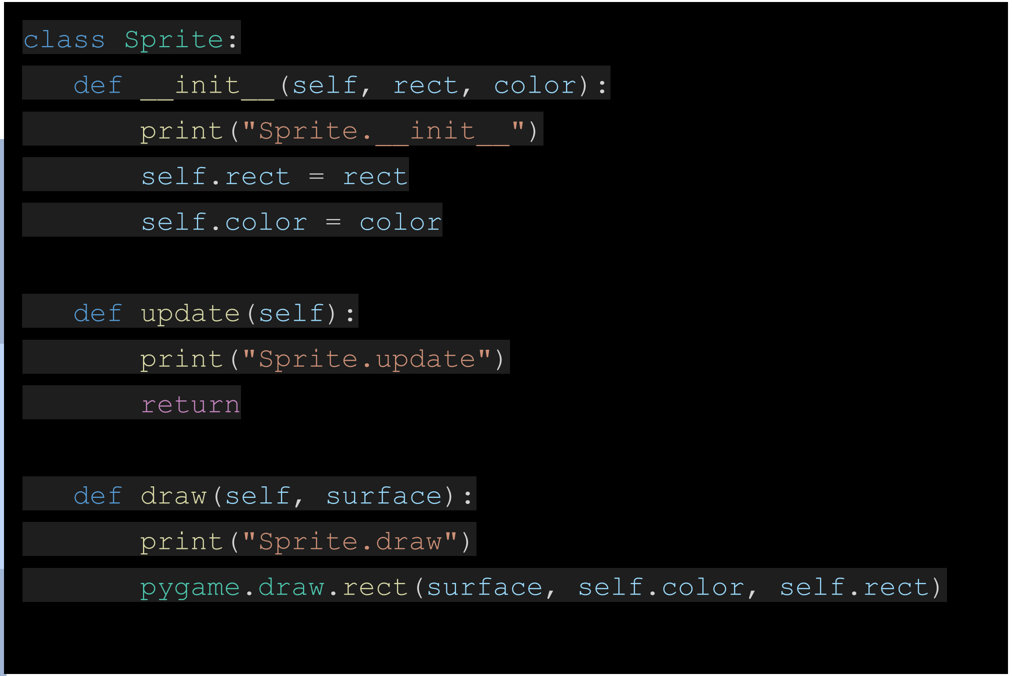
Inherit Parent: Player inherits from Sprite class.

Constructor: Call super().__init__() to use the parent constructor. Add parameters if the child class has additional attributes to set.

Update method: We don’t need a special update method for Player.

Additional methods: The player moving when keys are pressed is unique to the Player.

Observe



```
class Sprite:
    def __init__(self, rect, color):
        print("Sprite.__init__")
        self.rect = rect
        self.color = color

    def update(self):
        print("Sprite.update")
        return

    def draw(self, surface):
        print("Sprite.draw")
        pygame.draw.rect(surface, self.color, self.rect)
```

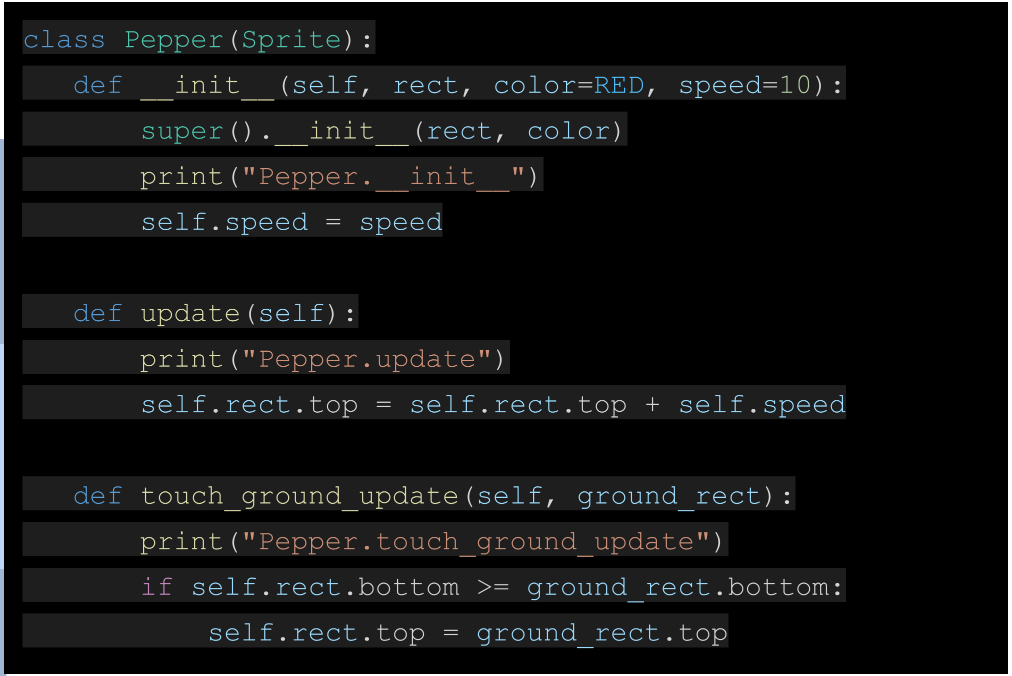
We are going to put print statements at the top of each method, and observe what prints at the command line.

Sprite.__init__

Sprite.update

Sprite.draw

Observe



```python
class Pepper(Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        print("Pepper.__init__")
        self.speed = speed

    def update(self):
        print("Pepper.update")
        self.rect.top = self.rect.top + self.speed

    def touch_ground_update(self, ground_rect):
        print("Pepper.touch_ground_update")
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top
```

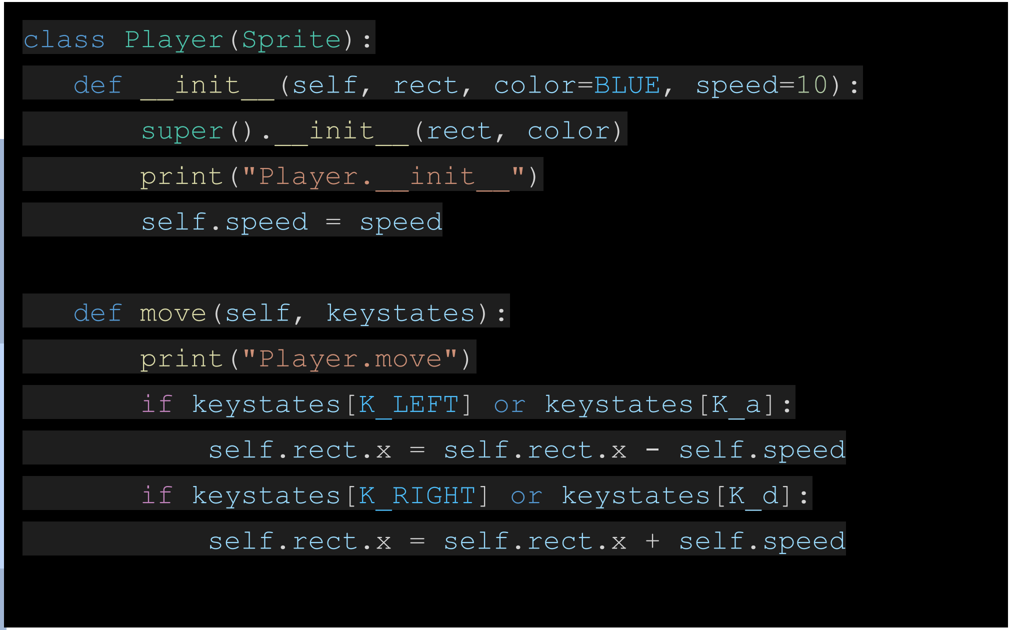
We are going to put print statements at the top of each method, and observe what prints at the command line.

Pepper.__init__ (after super())

Pepper.update

Pepper.touch_ground_update

Observe



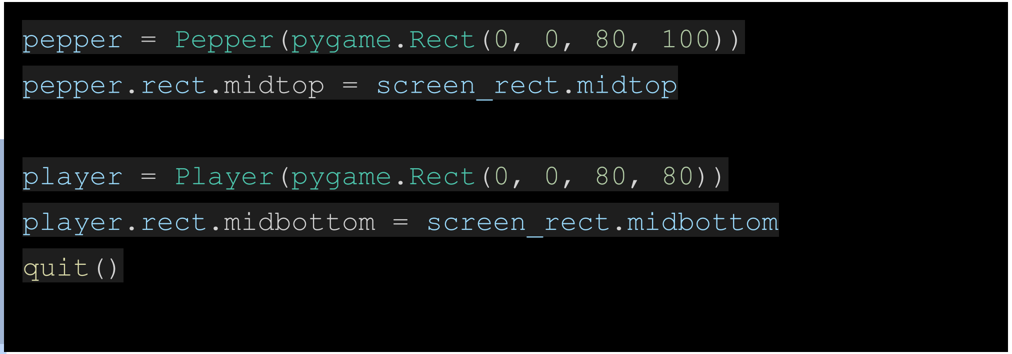
```
class Player(Sprite):
    def __init__(self, rect, color=BLUE, speed=10):
        super().__init__(rect, color)
        print("Player.__init__")
        self.speed = speed

    def move(self, keystates):
        print("Player.move")
        if keystates[K_LEFT] or keystates[K_a]:
            self.rect.x = self.rect.x - self.speed
        if keystates[K_RIGHT] or keystates[K_d]:
            self.rect.x = self.rect.x + self.speed
```

We are going to put print statements at the top of each method, and observe what prints at the command line.

Player.__init__ (after super())

Player.update

Observe



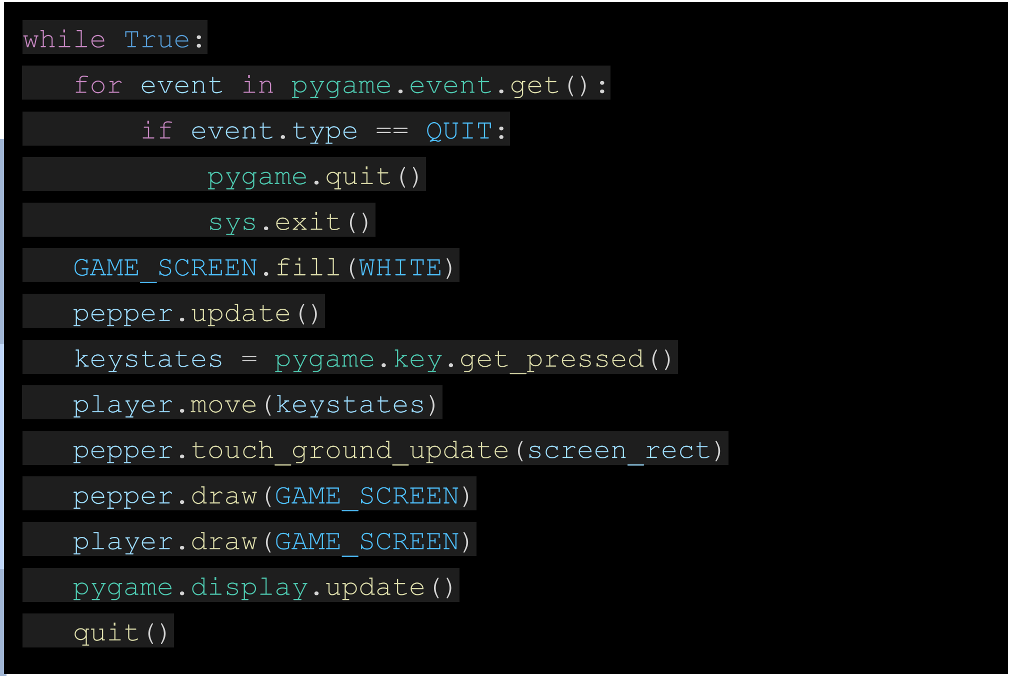
```
pepper = Pepper(pygame.Rect(0, 0, 80, 100))
pepper.rect.midtop = screen_rect.midtop

player = Player(pygame.Rect(0, 0, 80, 80))
player.rect.midbottom = screen_rect.midbottom
quit()
```

Before our game loop, let’s create a pepper object and a player object and then tell the program to quit.

This is what prints:

```
Sprite.__init__
Pepper.__init__
Sprite.__init__
Player.__init__
```

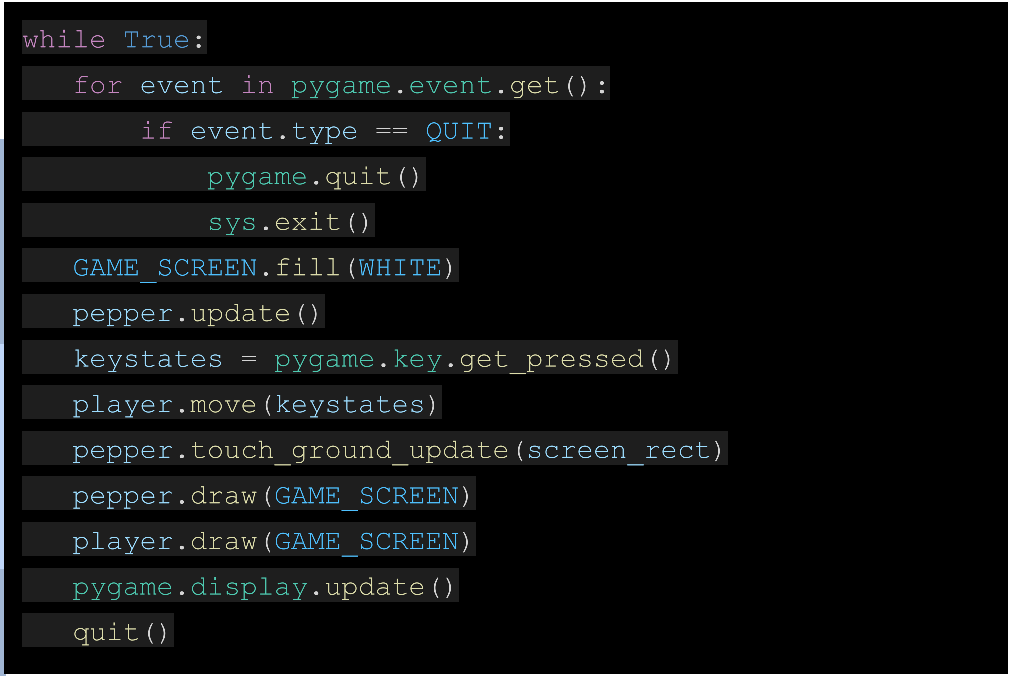
Observe



```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    GAME_SCREEN.fill(WHITE)
    pepper.update()
    keystates = pygame.key.get_pressed()
    player.move(keystates)
    pepper.touch_ground_update(screen_rect)
    pepper.draw(GAME_SCREEN)
    player.draw(GAME_SCREEN)
    pygame.display.update()
    quit()
```

Now let’s remove the quit() and run our game loop. Only using one pepper right now, no pepper_list. Quit after one iteration.

Observe



```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    GAME_SCREEN.fill(WHITE)
    pepper.update()
    keystates = pygame.key.get_pressed()
    player.move(keystates)
    pepper.touch_ground_update(screen_rect)
    pepper.draw(GAME_SCREEN)
    player.draw(GAME_SCREEN)
    pygame.display.update()
    quit()
```

This is what prints:

```
Sprite.__init__
Pepper.__init__
Sprite.__init__
Player.__init__
Pepper.update
Player.move
Pepper.touch_ground_update
Sprite.draw
Sprite.draw
```

Part 2

Pygame classes

Pygame classes and modules

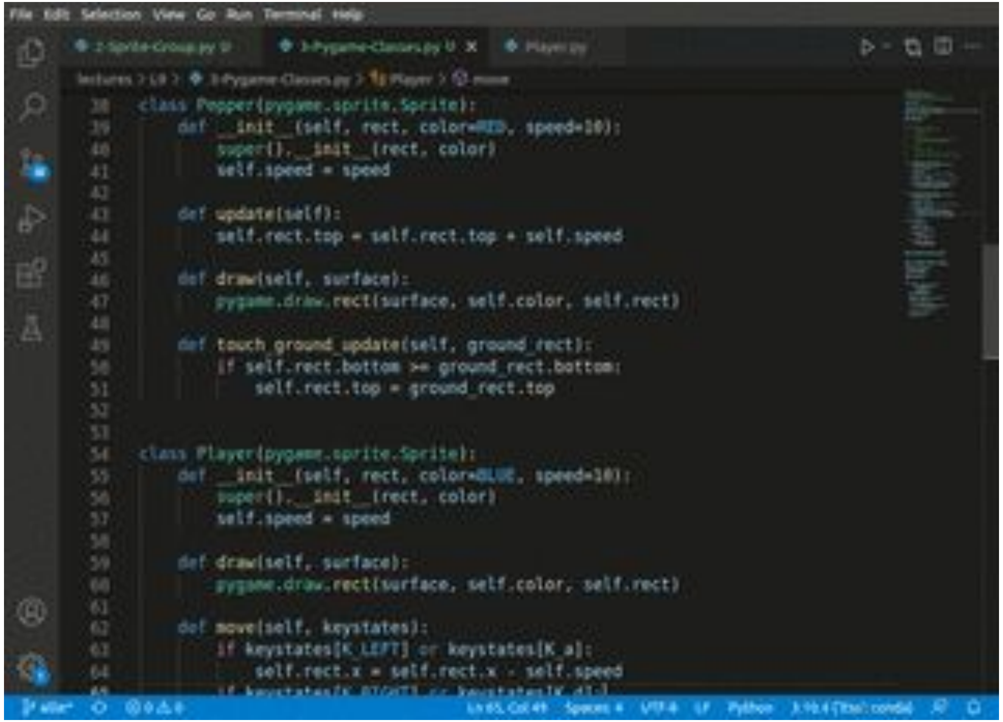
- We do not need to always create our own classes because Pygame provides many classes and modules
- They anticipate the typical programming needs for Game Programming and provide these to make game programming easier, faster, and cleaner
- It helps to understand how classes work so you can better understand how to use what Pygame has to offer!
- Now that you have learned a little bit about classes, let’s go through some useful tools provided by Pygame
- Don’t forget, you can always refer to their documentation! https://www.pygame.org/docs/

pygame.sprite.Sprite

- pygame.sprite.Sprite is a simple base class for visible game objects
- Let’s try to change the code we just worked on to use this class instead of the one we created

Tip

In VS Code, you can hover your mouse pointer over something you’ve imported and see a preview. You can also click to be shown its implementation. Then we can take a look about what’s different about your code.


File Edit Selection View Go Run Terminal Help
2-Sprite-Group.py U
3-Pygame-Classes.py U
Player.py
lectures > L8 > 3-Pygame-Classes.py > Player > move
Ln 65, Col 49 Spaces: 4 UTF-8 LF Python

```python
class Pepper(pygame.sprite.Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def update(self):
        self.rect.top = self.rect.top + self.speed

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)

    def touch_ground_update(self, ground_rect):
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top


class Player(pygame.sprite.Sprite):
    def __init__(self, rect, color=BLUE, speed=10):
        super().__init__(rect, color)
        self.speed = speed

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)

    def move(self, keystates):
        if keystates[K_LEFT] or keystates[K_a]:
            self.rect.x = self.rect.x - self.speed
        if keystates[K_RIGHT] or keystates[K_d]:
```

Let’s get you some starter code…

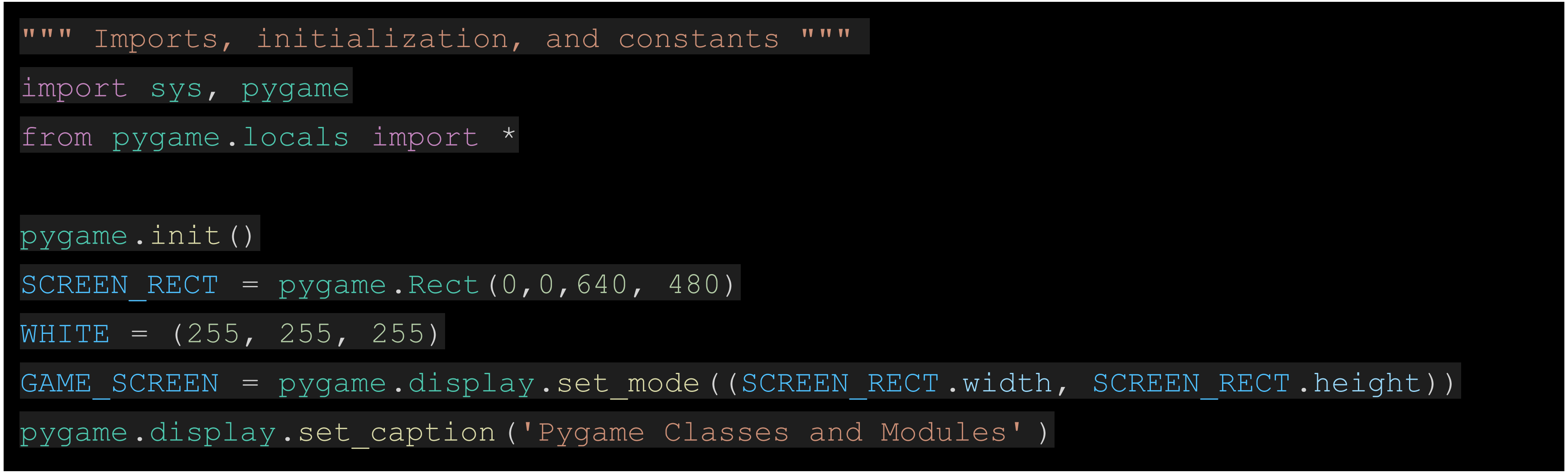
Instructions on where to get the starter code.

First, run the program and see what you can do in the game.

Then, let’s walk through how the code works.

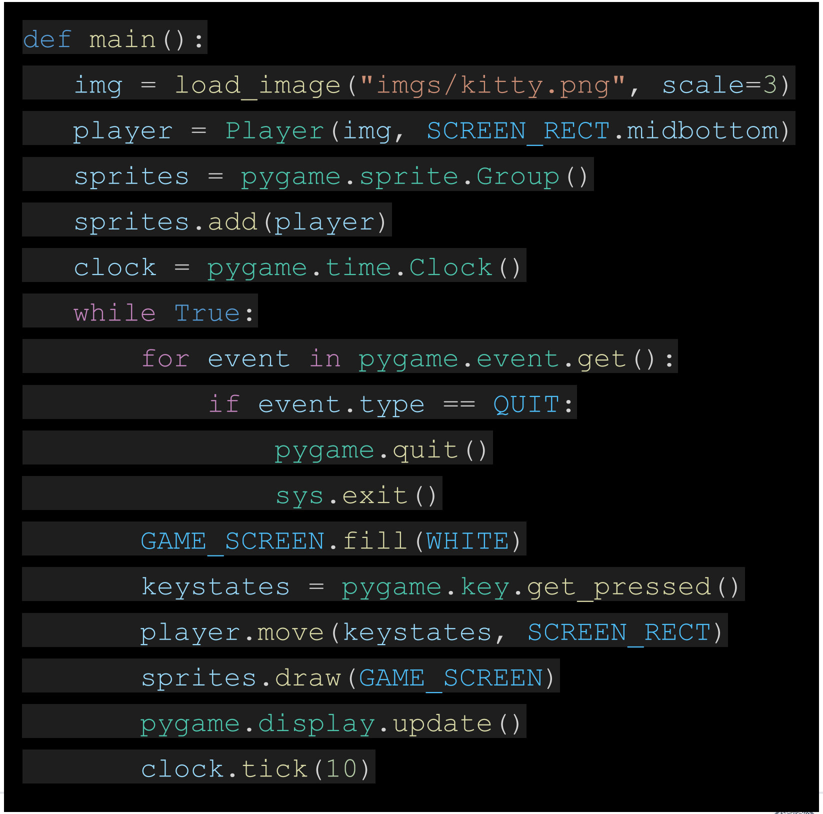
Imports, initialization, and constants

Starting at the top of the file, there’s nothing new here. These are typical for beginning the game program.



```
""" Imports, initialization, and constants """
import sys, pygame
from pygame.locals import *

pygame.init()
SCREEN_RECT = pygame.Rect(0,0,640, 480)
WHITE = (255, 255, 255)
GAME_SCREEN = pygame.display.set_mode((SCREEN_RECT.width, SCREEN_RECT.height))
pygame.display.set_caption('Pygame Classes and Modules' )
```

Main function



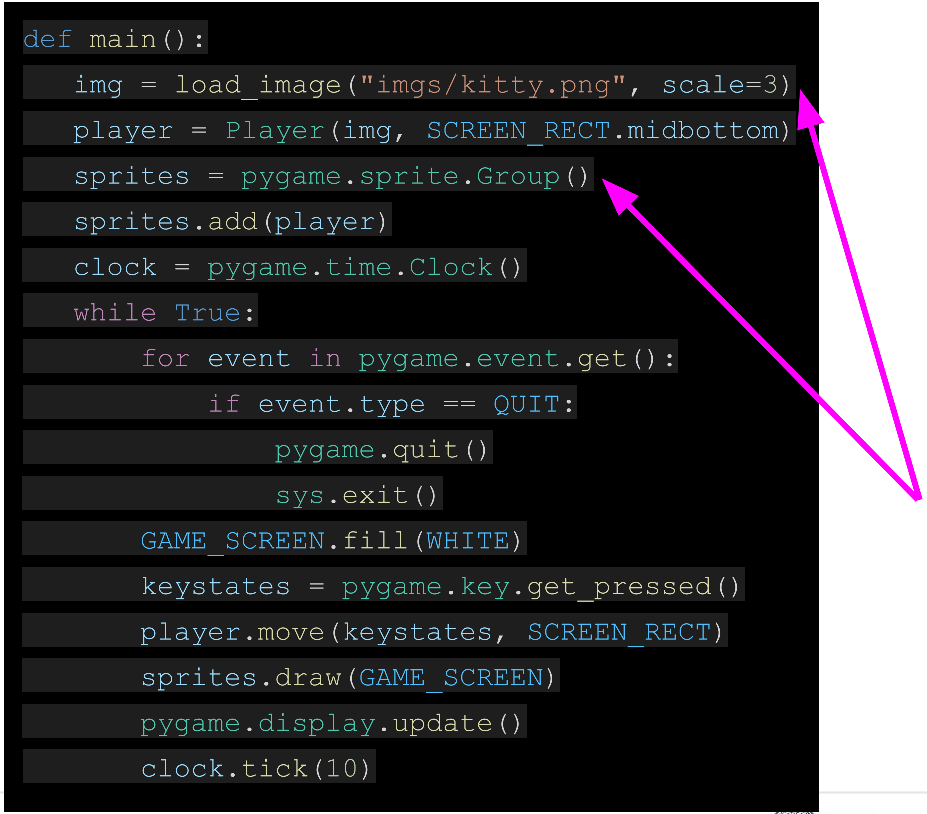
```python
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

Scroll toward the bottom of the file to find the main function.

Everything here should look quite familiar. We have our player object, the clock, and the game loop.

What’s new?

Main function



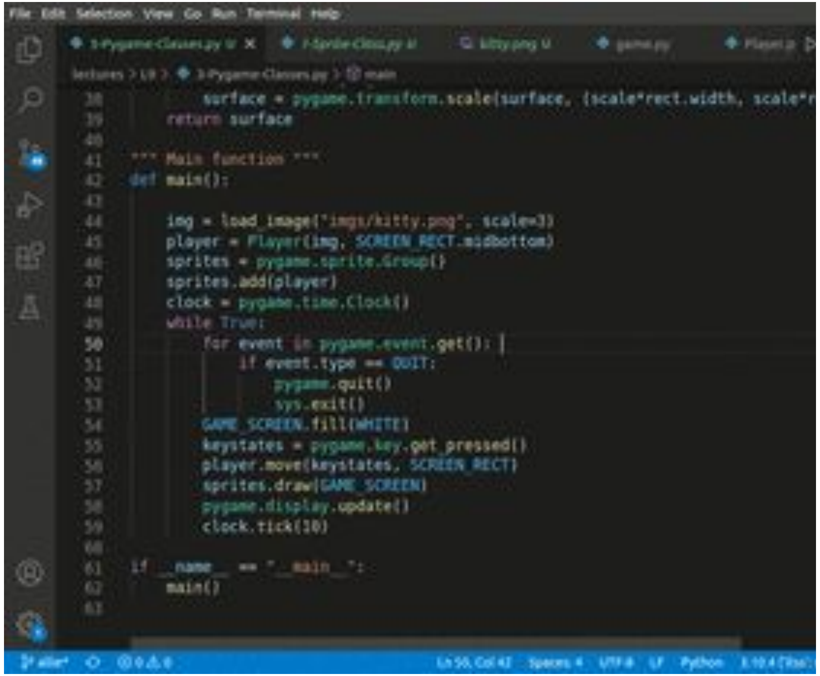
```python
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

Scroll toward the bottom of the file to find the main function.

Everything here should look quite familiar. We have our player object, the clock, and the game loop.

What’s new? **load_image()** and **pygame.sprite.Group().**

Load image


File Edit Selection View Go Run Terminal Help
3-Pygame-Classes.py U
7-Sprite-Class.py U
kitty.png U
game.py
Player.p
lectures > L9 > 3-Pygame-Classes.py > main
aller*
Ln 50, Col 42 Spaces: 4 UTF-8 LF Python 3.10.4

```
        surface = pygame.transform.scale(surface, (scale*rect.width, scale*r
    return surface

""" Main function """
def main():

    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)

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

Let’s take a look at the load_image function. In VS Code we can Control+Click on the function name and it will take us to its definition.

But it’s right above the main function :)

Load image

```python
""" Load image function """
def load_image(file_path, size=None, scale=None):
    """
    size: tuple (width, height)
    scale: integer value to scale the image
    """
    surface = pygame.image.load(file_path)
    if size:
        surface = pygame.transform.scale(surface, size)
    if scale:
        rect = surface.get_rect()
        surface = pygame.transform.scale(surface, (scale*rect.width, scale*rect.height))
    return surface
```

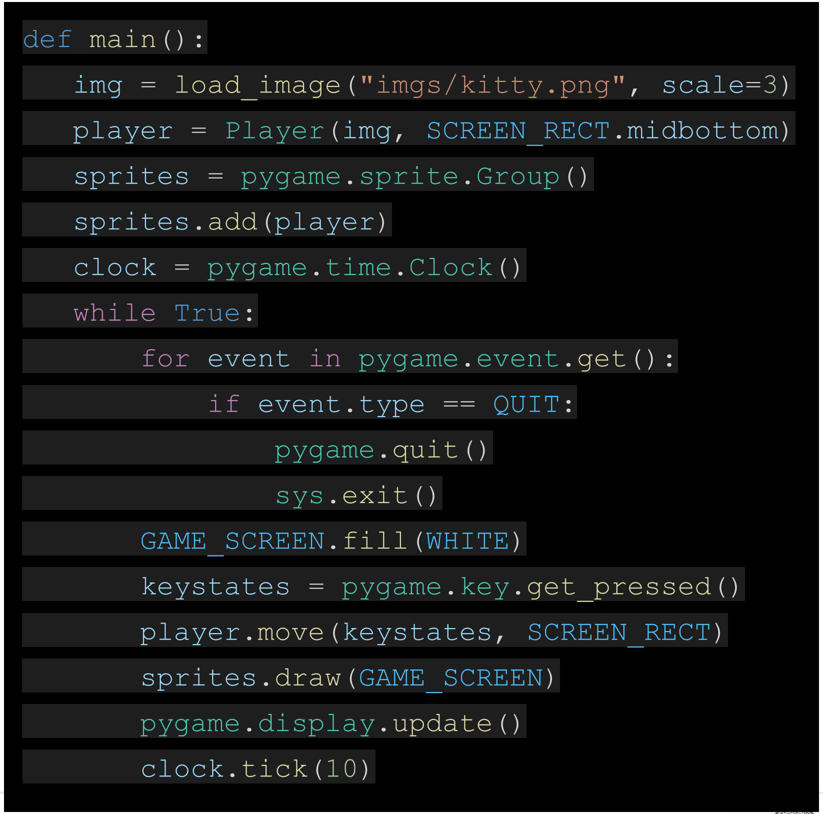
If you want to use an image for your sprites, you can simply use `pygame.image.load()` and pass in the path to your image file.

Load image

```python
""" Load image function """
def load_image(file_path, size=None, scale=None):
    """
    size: tuple (width, height)
    scale: integer value to scale the image
    """
    surface = pygame.image.load(file_path)
    if size:
        surface = pygame.transform.scale(surface, size)
    if scale:
        rect = surface.get_rect()
        surface = pygame.transform.scale(surface, (scale*rect.width, scale*rect.height))
    return surface
```

We wanted the ability to resize or rescale the image, so we made our own function with some extra steps to do that. Do you see how we do that?

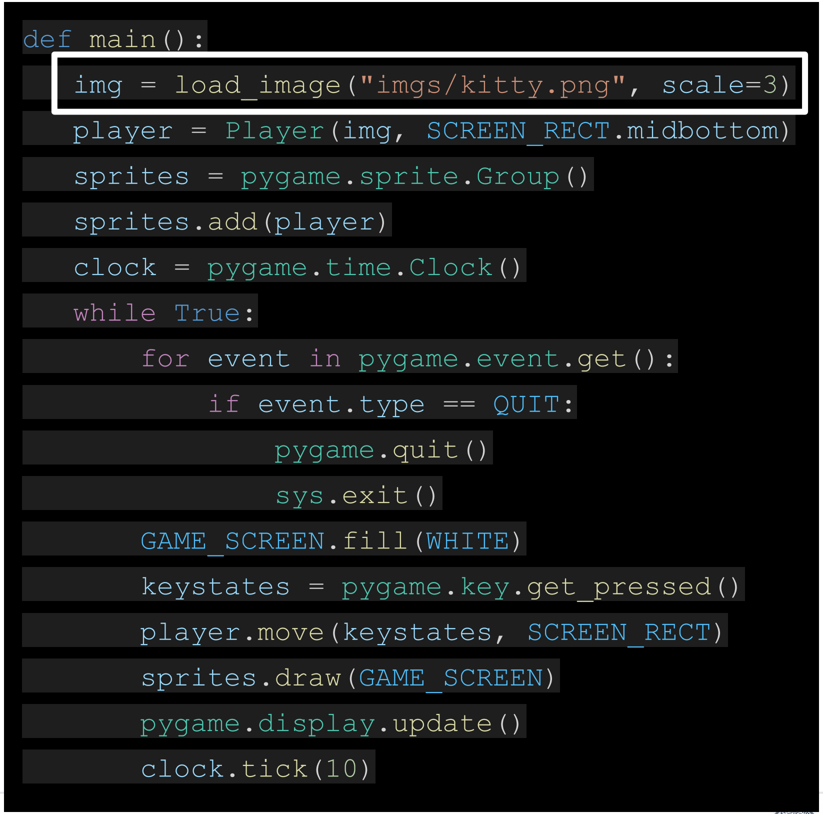
Main function



```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

Now, let’s go back to the main function and see how we use the load_image() function.

Main function

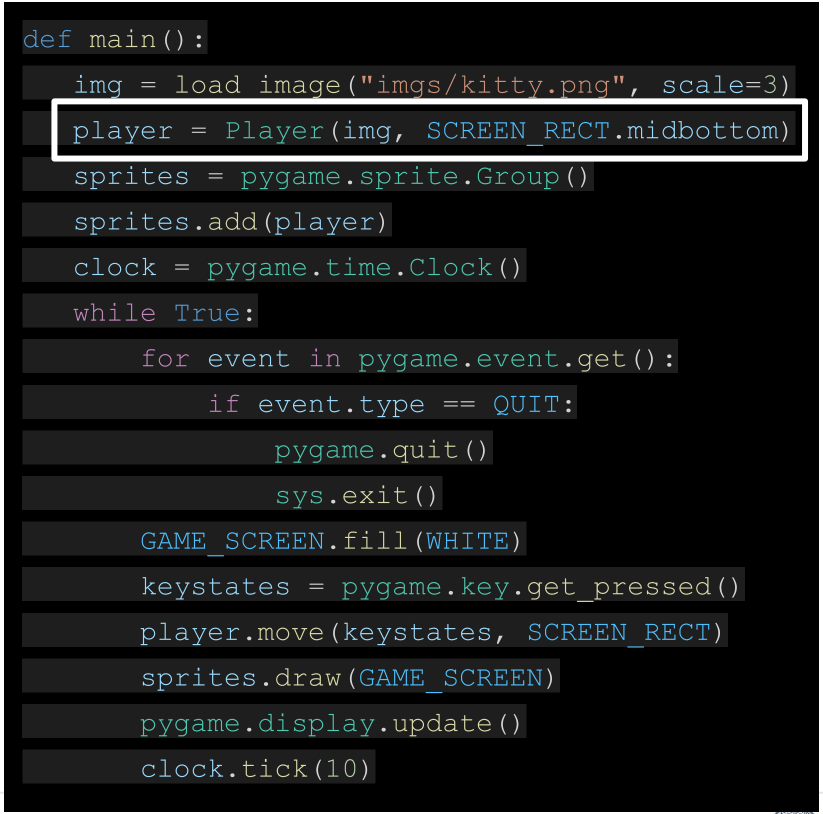


```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

We pass in the path to our image relative to where we are running the code.

We also set a value for scale, to tell the function that we want the image to be 3 times as large as its original size.

Main function

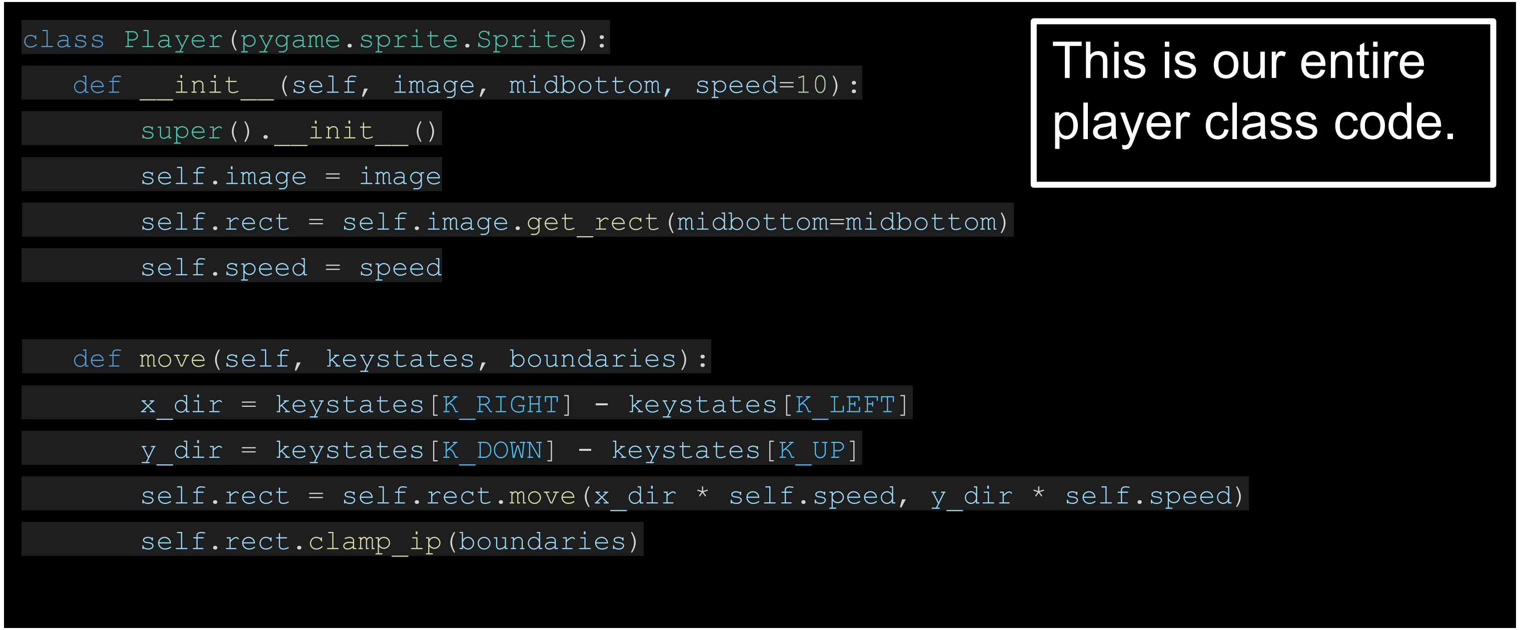


```
def main():
    img = load image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

Then, we take the output of load_image() and pass it to our initialization of our Player class object.

Now let’s jump to the Player class code to see what the class does with the image.

Player class



```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

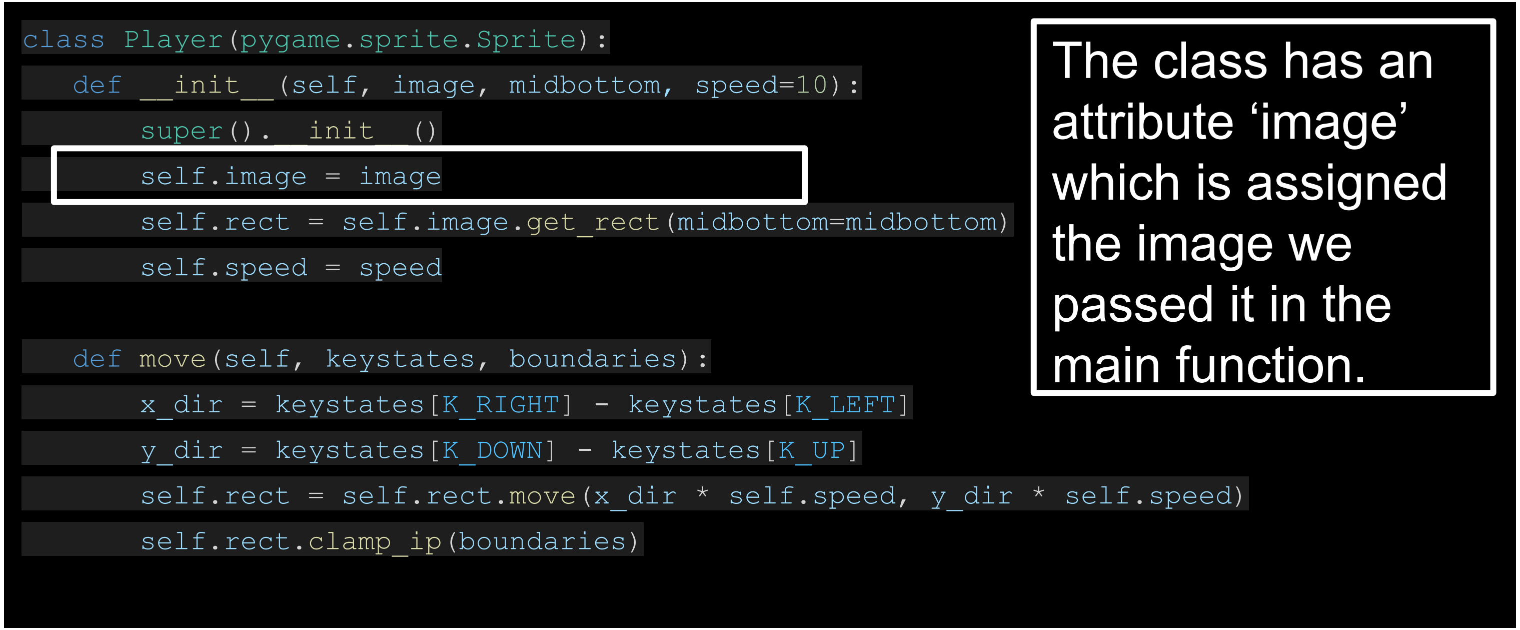
This is our entire player class code.

pygame.sprite.Sprite

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Notice that Player has a pygame parent class.

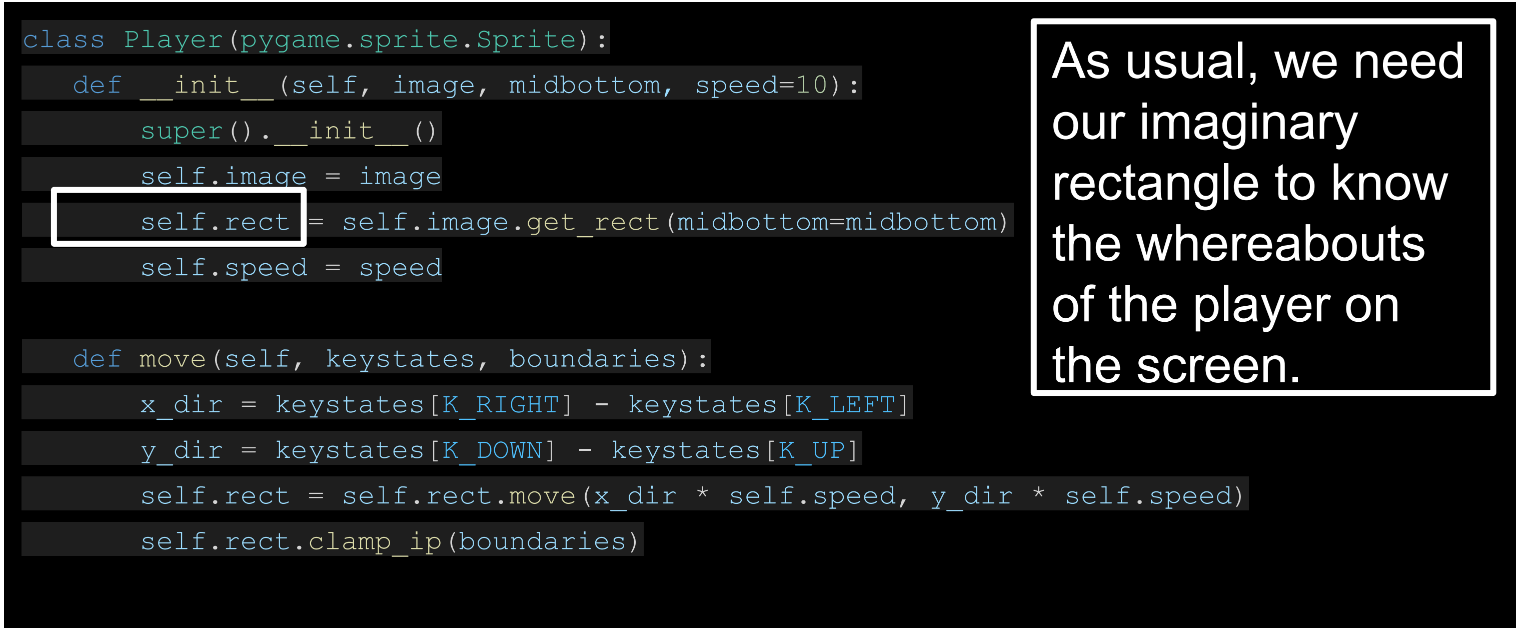
Player class



```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

The class has an attribute ‘image’ which is assigned the image we passed it in the main function.

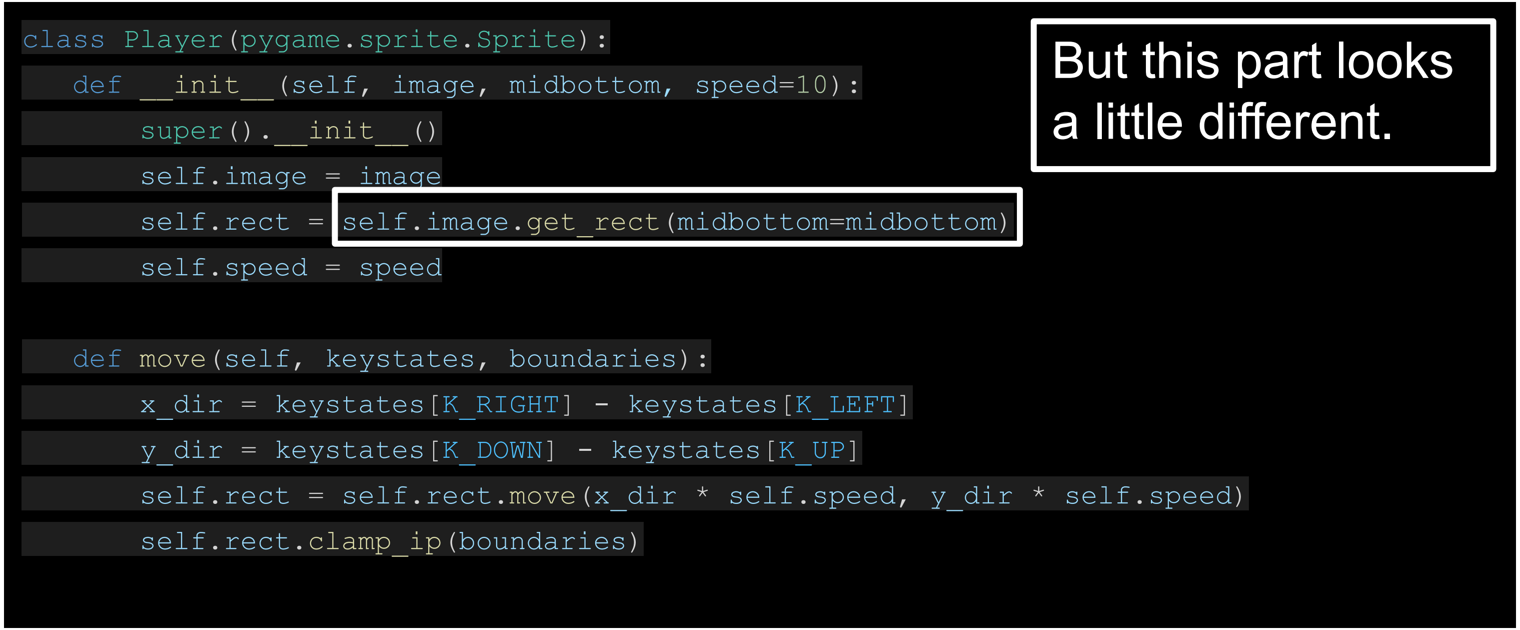
Player class



```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

As usual, we need our imaginary rectangle to know the whereabouts of the player on the screen.

Player class

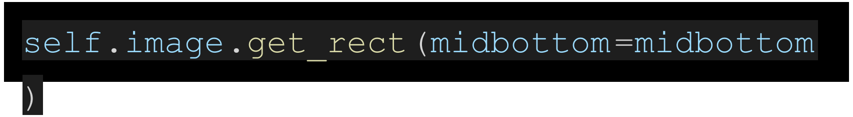


```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

But this part looks a little different.

pygame.Surface.get_rect()

The image is a **pygame.Surface** object.

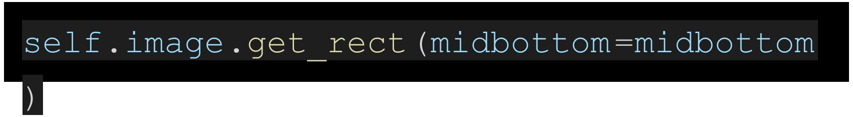


```
self.image.get_rect(midbottom=midbottom
)
```

pygame.Surface has a method **get_rect()** that returns the rectangular area of the surface object.

pygame.Surface.get_rect()

The image is a **pygame.Surface** object.



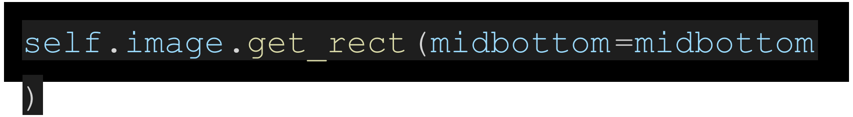
```
self.image.get_rect(midbottom=midbottom
)
```

pygame.Surface has a method **get_rect()** that returns the rectangular area of the surface object.

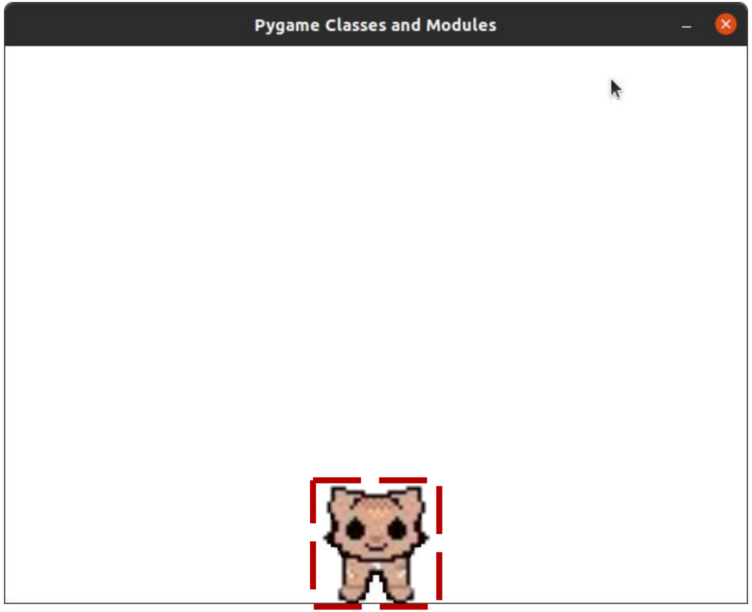
So we want to use this to get our imaginary rectangle around our sprite image.

pygame.Surface.get_rect()

Thanks to how pygame.Surface.get_rect() is implemented, we can set the starting point of our sprite by passing our desired (x,y) coordinate with a keyword, one of the pygame.Rect attributes.



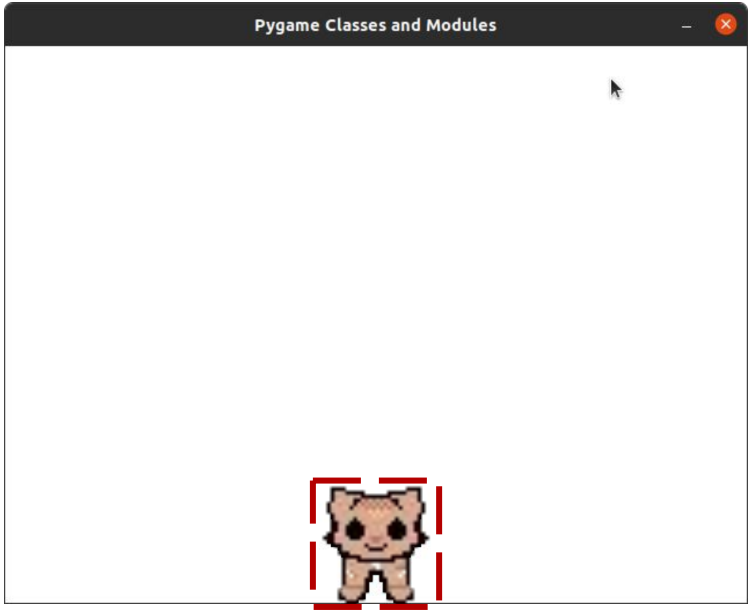
```
self.image.get_rect(midbottom=midbottom
)
```



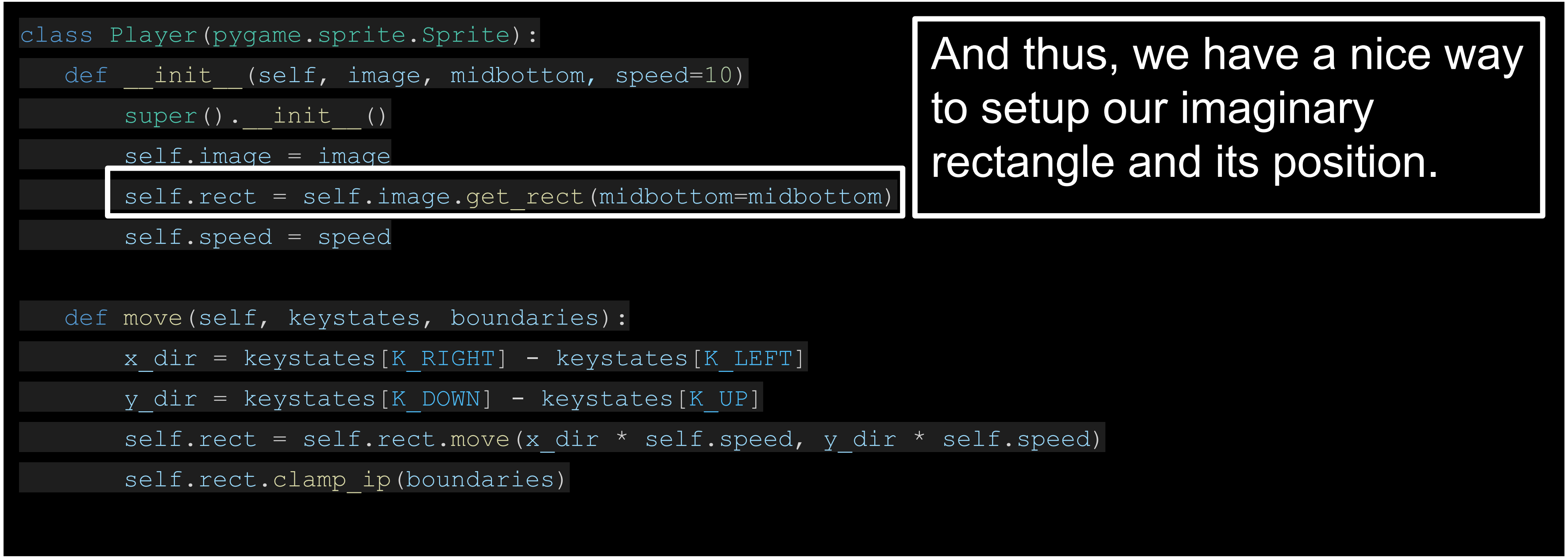
pygame.Surface

Don’t forget, you can always find out more about what you can do with pygame.Surface in the documentation!

https://www.pygame.org/docs/ref/surface.html



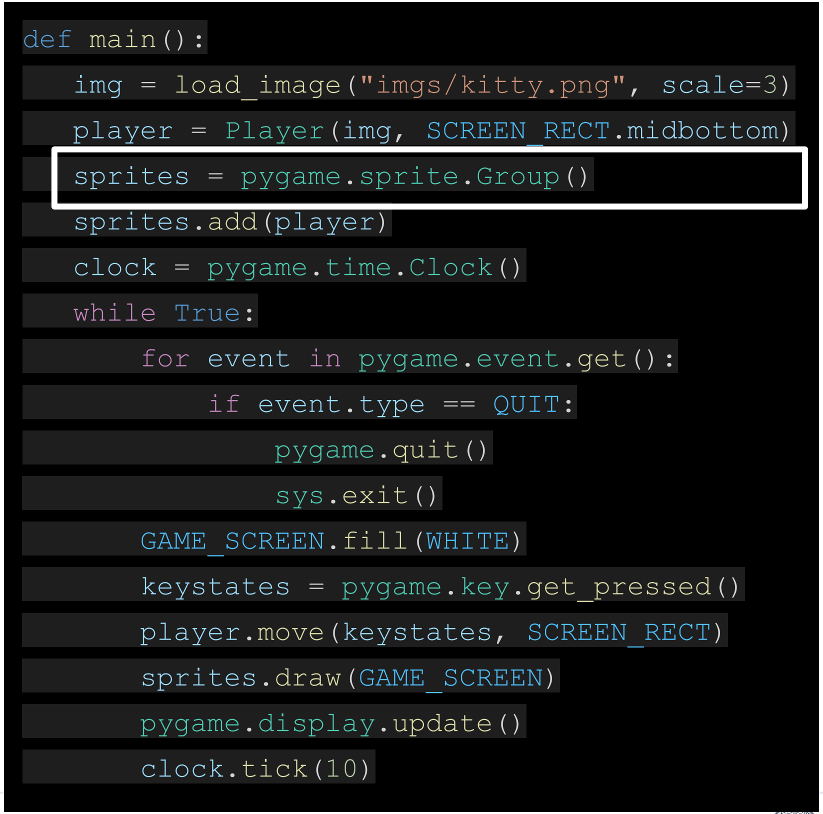
Player class



```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

And thus, we have a nice way to setup our imaginary rectangle and its position.

Main function



```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

Back to the main function.

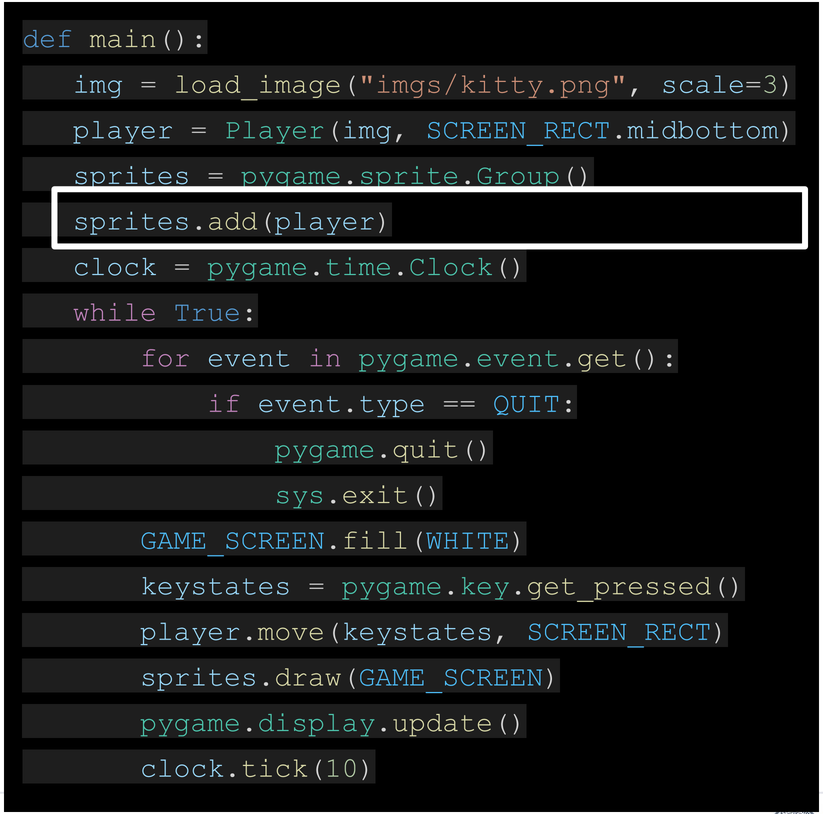
What is this ‘sprites’ thing?

It’s a pygame.sprite.Group object.

Documentation ;)
https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.Group

First, let’s see what we do with this object throughout the main function.

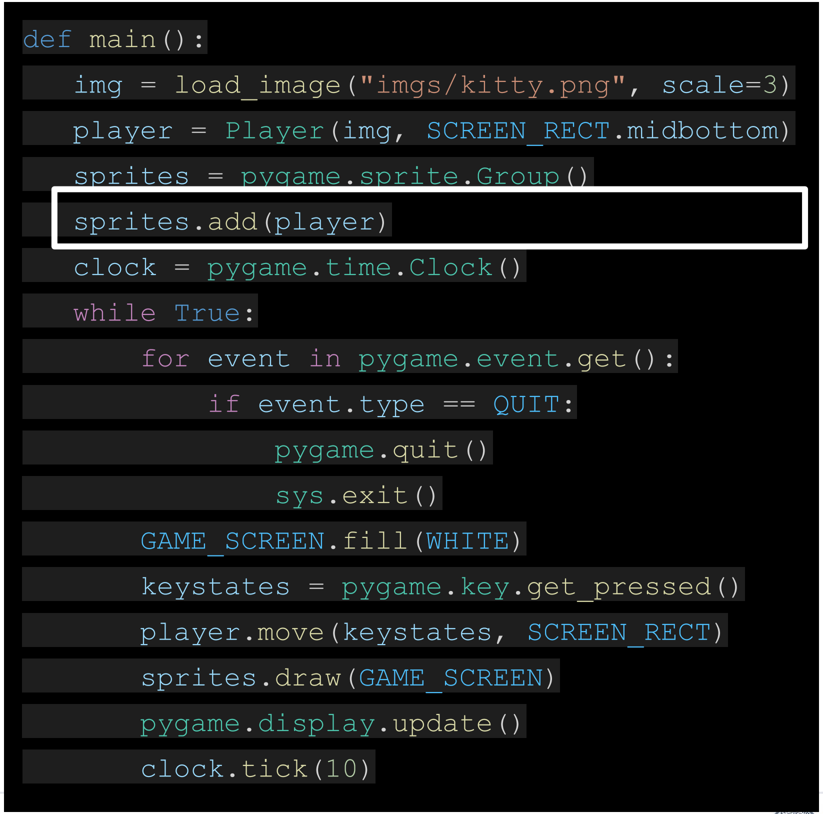
Main function



```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

1. We add the player to it.

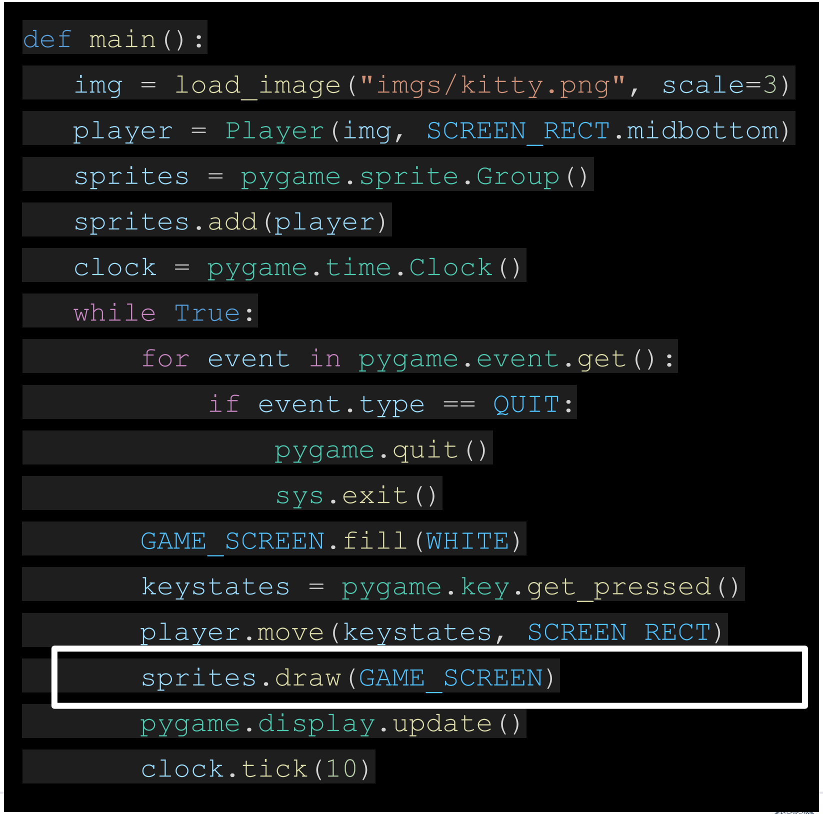
Main function



```python
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

1. We add the player to it. This tells us it might be some sort of container for sprite objects.

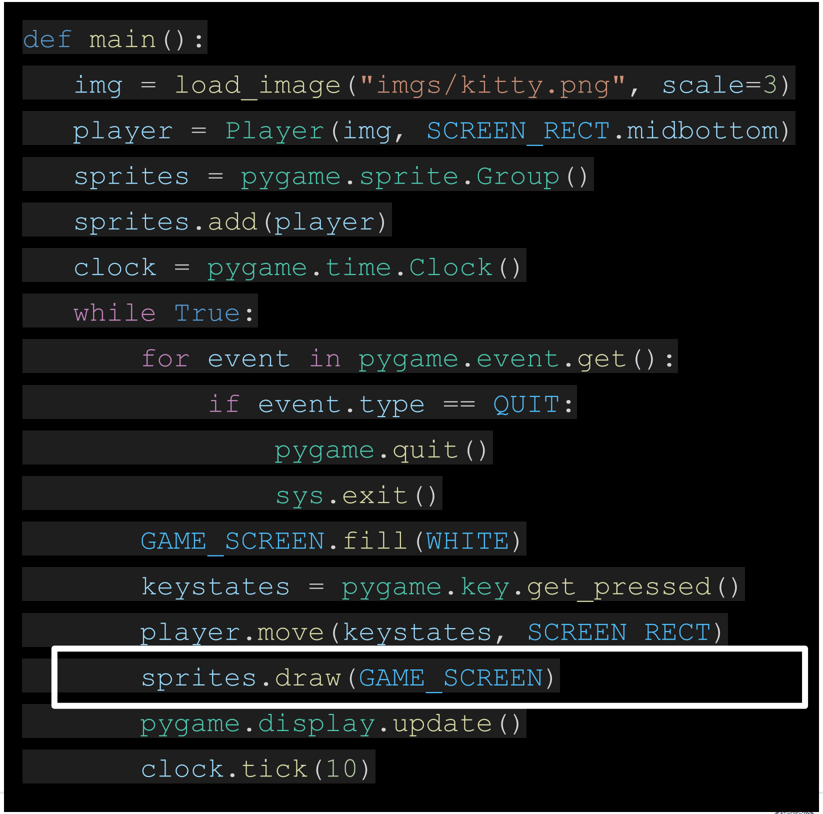
Main function



```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

1. We add the player to it. This tells us it might be some sort of container for sprite objects.
2. Then, we call sprites.draw().

Main function



```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

1. We add the player to it. This tells us it might be some sort of container for sprite objects.
2. Then, we call sprites.draw(). Does this remind you of something?

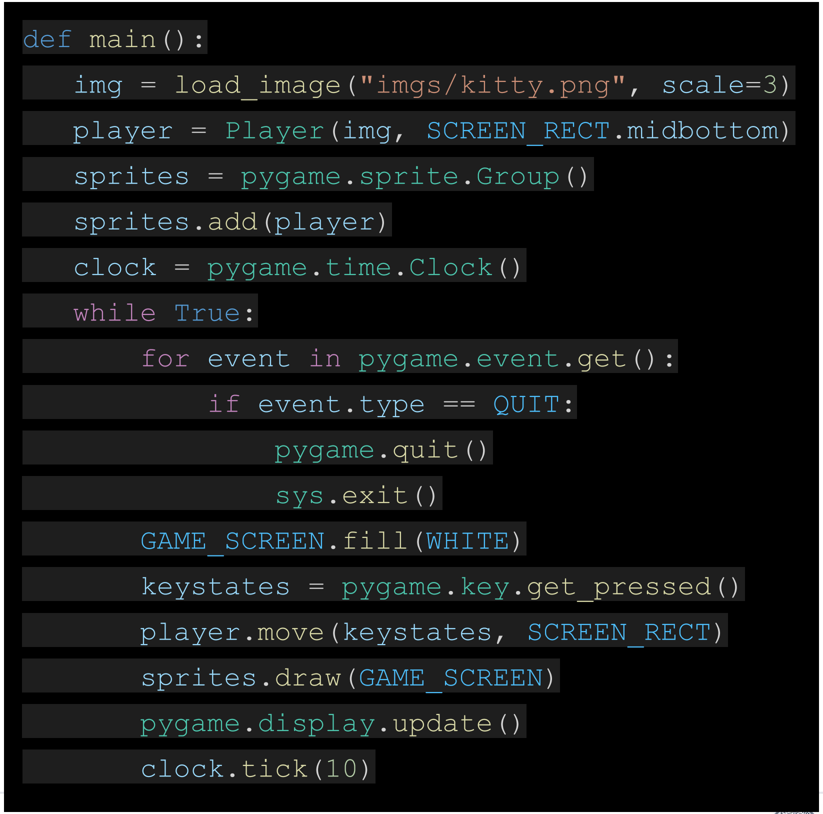
pygame.sprite.Group

```
class PepperGroup:
    def __init__(self):
        self.items = []

    def add(self, item):
        self.items.append(item)

    def update(self):
        for item in self.items:
            item.update()

    def draw(self, surface):
        for item in self.items:
            item.draw(surface)
```

Pygame provides classes similar to our PepperGroup we created in an earlier tutorial. We don’t have to make our own!

Documentation:
https://www.pygame.org/docs/ref/sprite.html

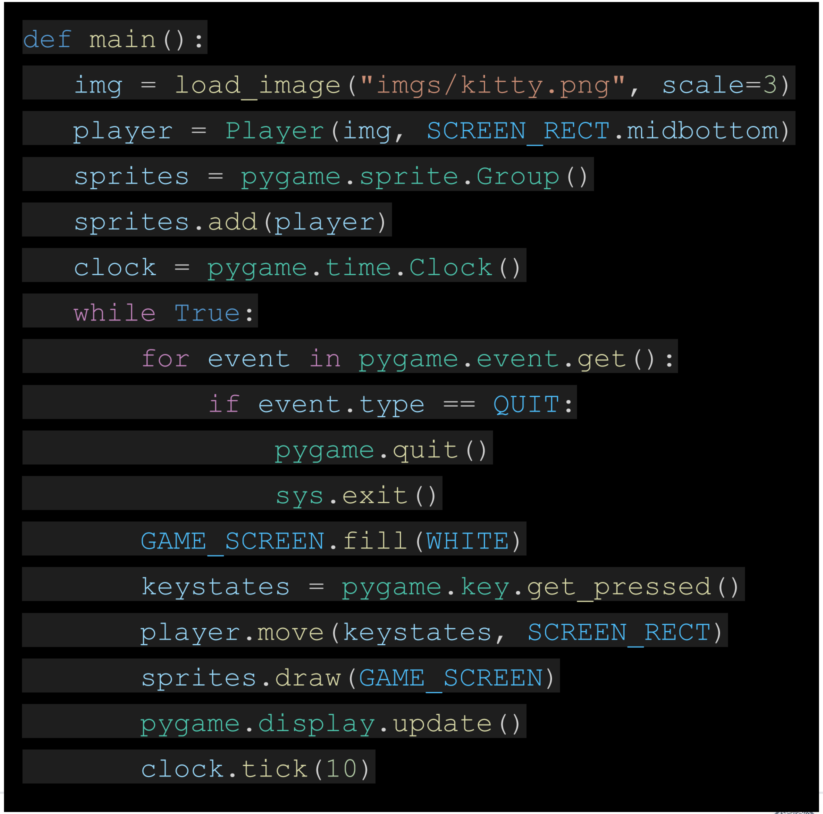
Main function



```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

- Currently, the sprite group only has one sprite: the player.
- If we wanted to add more objects to the game, we could make another pygame.sprite.Sprite object and add it to the sprite group.
- Then, the object will be drawn.
- If we want to add behavior, such as falling as time passes, what do we need to add to the main function?

Main function

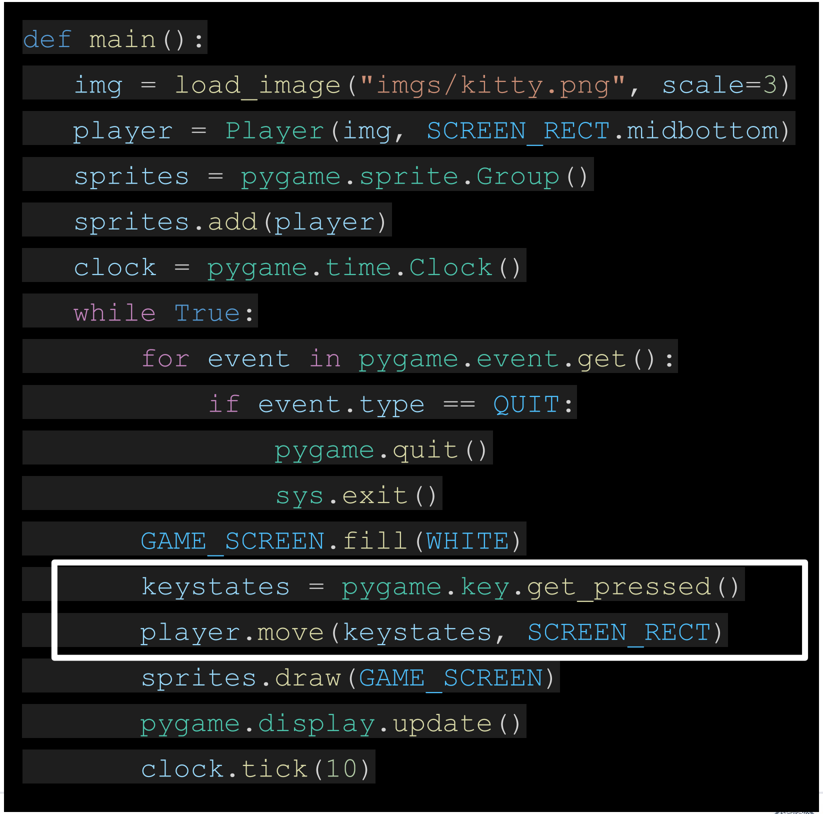


```
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

If we want to add behavior, such as falling as time passes, what do we need to add to the main function?

We need to call sprites.update(). We also need to implement the new object’s update method.

Main function



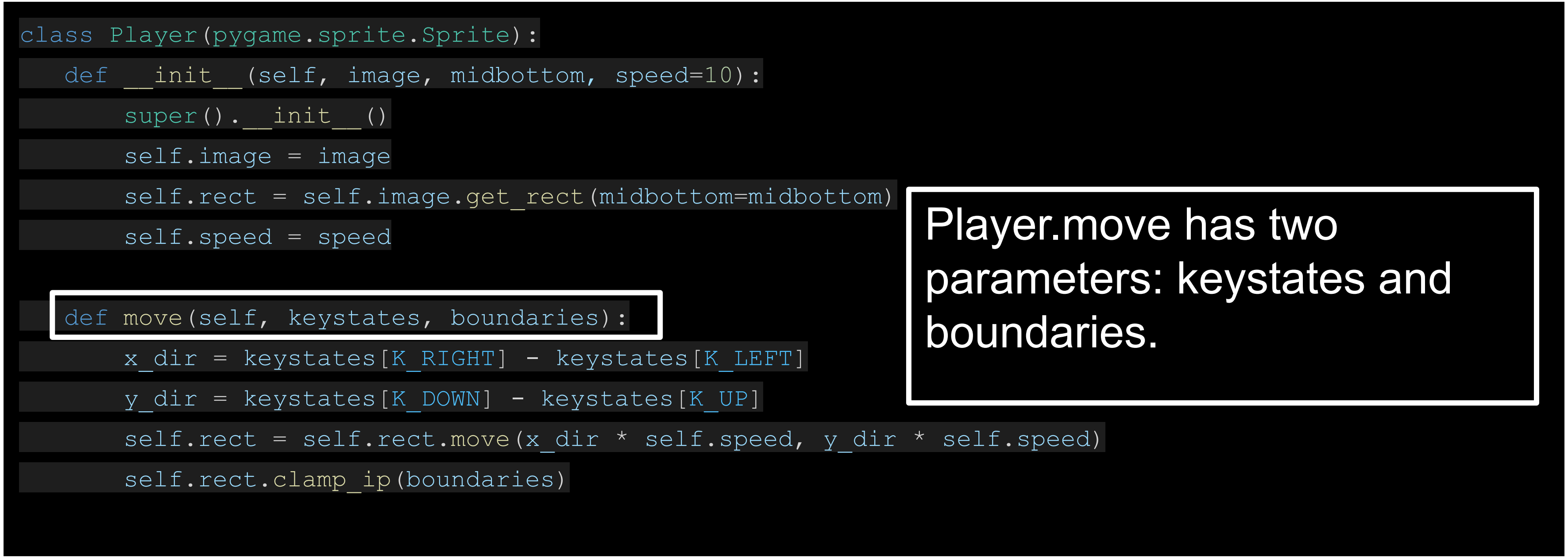
```python
def main():
    img = load_image("imgs/kitty.png", scale=3)
    player = Player(img, SCREEN_RECT.midbottom)
    sprites = pygame.sprite.Group()
    sprites.add(player)
    clock = pygame.time.Clock()
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
        GAME_SCREEN.fill(WHITE)
        keystates = pygame.key.get_pressed()
        player.move(keystates, SCREEN_RECT)
        sprites.draw(GAME_SCREEN)
        pygame.display.update()
        clock.tick(10)
```

Now let’s think about Player.move.

In the main function, we get the keystates and pass them to Player.move like we did in our previous tutorials.

We also pass in SCREEN_RECT. Let’s go to the implementation of the method to see what we do with it.

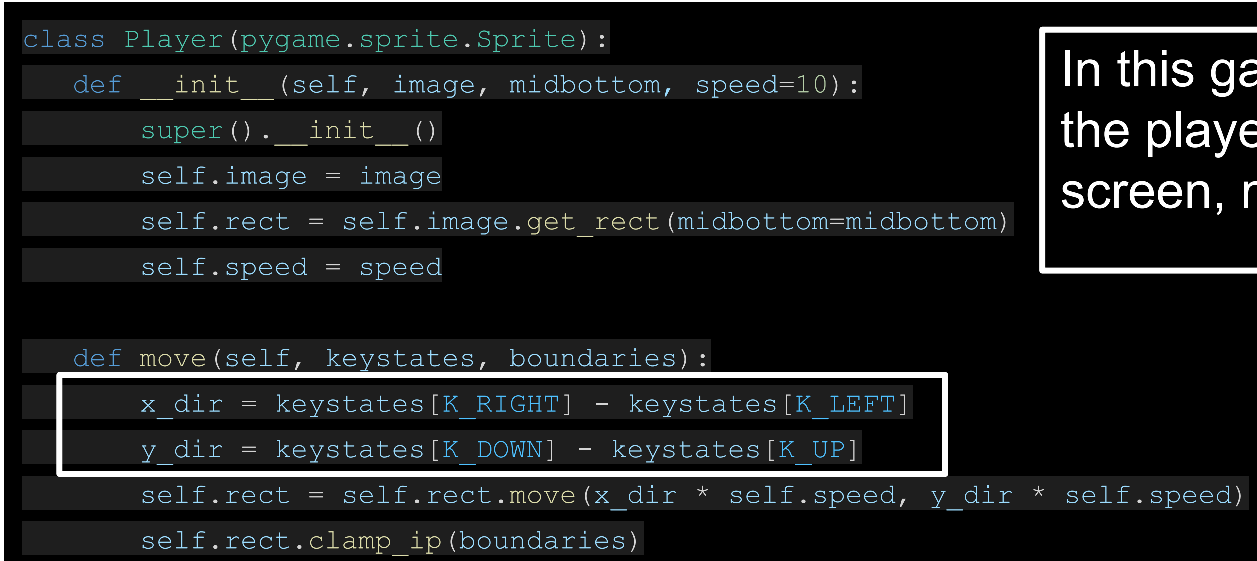
Player.move



```
class Player(pygame.sprite.Sprite):
   def __init__(self, image, midbottom, speed=10):
       super().__init__()
       self.image = image
       self.rect = self.image.get_rect(midbottom=midbottom)
       self.speed = speed

   def move(self, keystates, boundaries):
       x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
       y_dir = keystates[K_DOWN] - keystates[K_UP]
       self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
       self.rect.clamp_ip(boundaries)
```

Player.move has two parameters: keystates and boundaries.

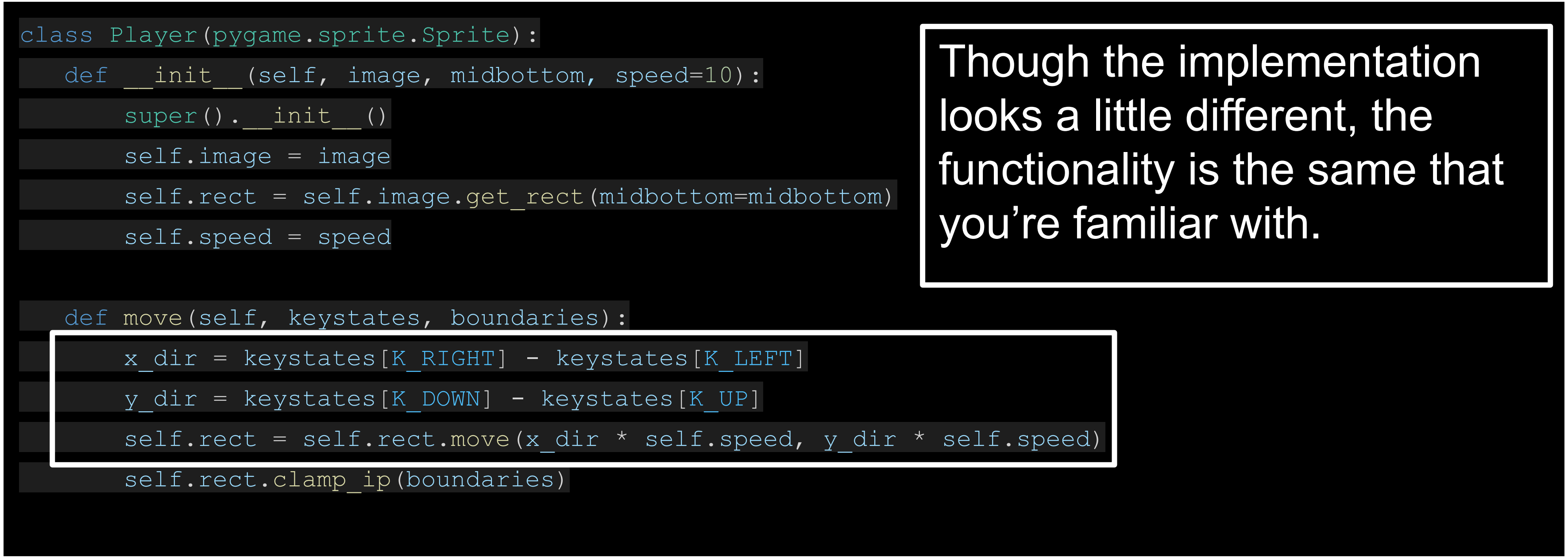
Player.move



```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

In this game, we can move the player all around the screen, not just left to right.

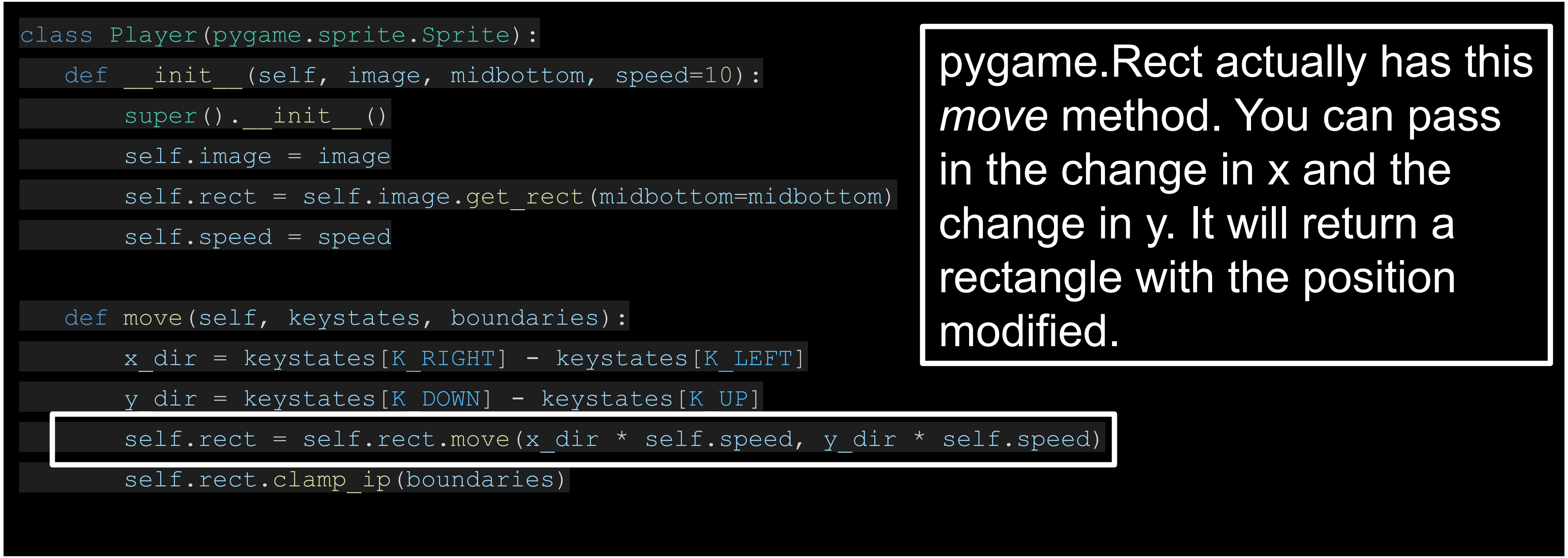
Player.move



```python
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Though the implementation looks a little different, the functionality is the same that you’re familiar with.

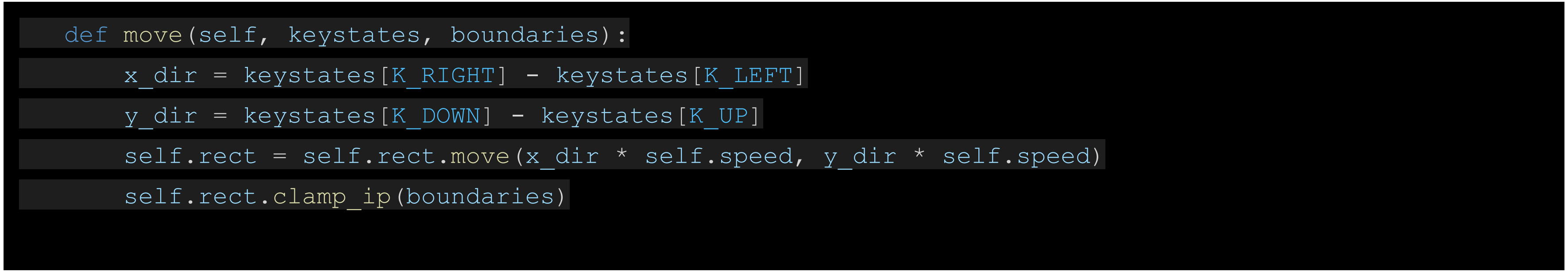
Player.move



```python
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

pygame.Rect actually has this *move* method. You can pass in the change in x and the change in y. It will return a rectangle with the position modified.

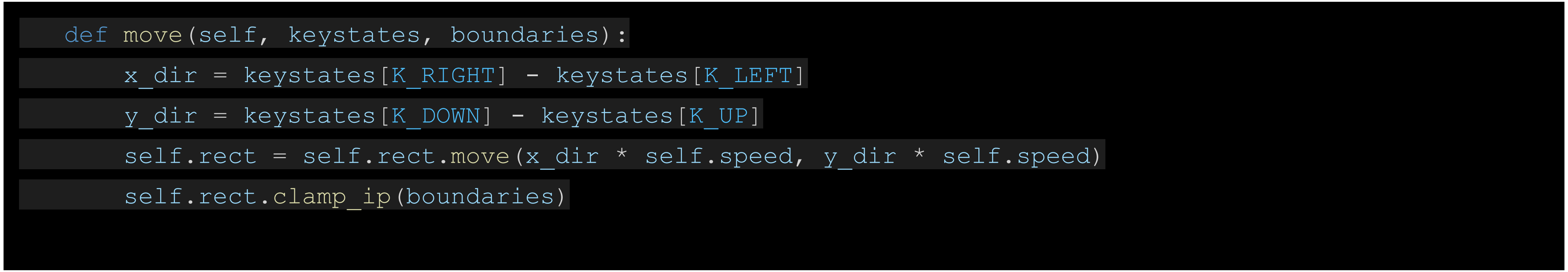
Player.move: Exercise.



```
    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

1. What is ***x_dir*** when only the ***right arrow key*** is pressed?
2. What is ***x_dir*** when only the ***left arrow key*** is pressed?
3. What is ***x_dir*** when both the ***left and right arrow keys*** are pressed?
4. What does the value of ***y_dir*** need to be in order for the player to move up the screen?

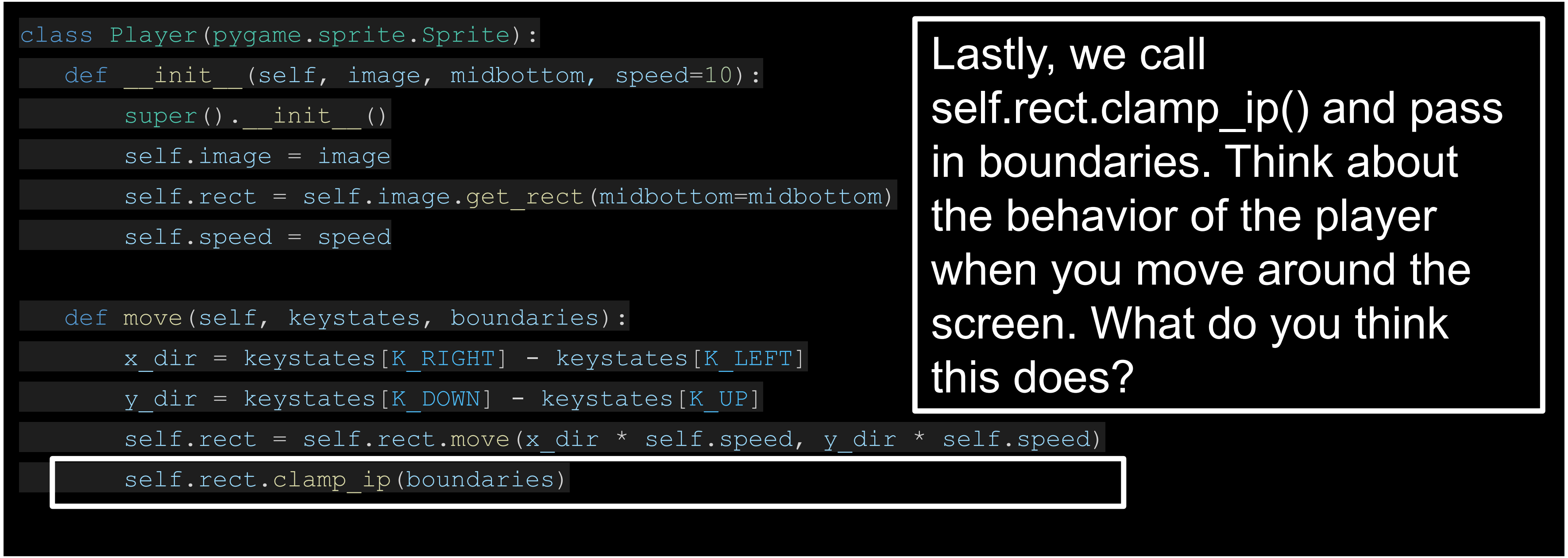
Player.move: Solution.



```python
    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

1. What is ***x_dir*** when only the ***right arrow key*** is pressed? **1**
2. What is ***x_dir*** when only the ***left arrow key*** is pressed? **-1**
3. What is ***x_dir*** when both the ***left and right arrow keys*** are pressed? **0**
4. What does the value of ***y_dir*** need to be in order for the player to move up the screen? **-1**

Player.move



```python
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Lastly, we call self.rect.clamp_ip() and pass in boundaries. Think about the behavior of the player when you move around the screen. What do you think this does?

Part 3

Making the game more animated