Object-Oriented Programming

Classes in Python with Py5

Building a Controllable Spaceship 🚀

Course Topics:

- What are classes?
- Creating objects
- Methods and attributes
- Building a Spaceship class
- Multiple objects
- Inheritance

What You'll Learn

✅ Understand what classes and objects are
✅ Create your own classes
✅ Use constructors and methods
✅ Build a controllable spaceship
✅ Manage multiple objects
✅ Use inheritance to extend classes
✅ Organize code better

By the end: You'll have a working spaceship game! 🎮

Course Outline

1. **Introduction to OOP**
2. **Your First Class**
3. **The Spaceship Class**
4. **Movement & Rotation**
5. **Drawing the Spaceship**
6. **User Control**
7. **Multiple Objects**
8. **Inheritance**
9. **Complete Game**

Part 1: Introduction to OOP

What is Object-Oriented Programming?

OOP = Programming with "objects" that have:

- **Properties** (data/attributes)
- **Behaviors** (methods/functions)

Real World Example: A Car 🚗

- **Properties:** color, speed, position, fuel
- **Behaviors:** accelerate(), brake(), turn()

In code: We use **classes** to define these objects

Why Use Classes?

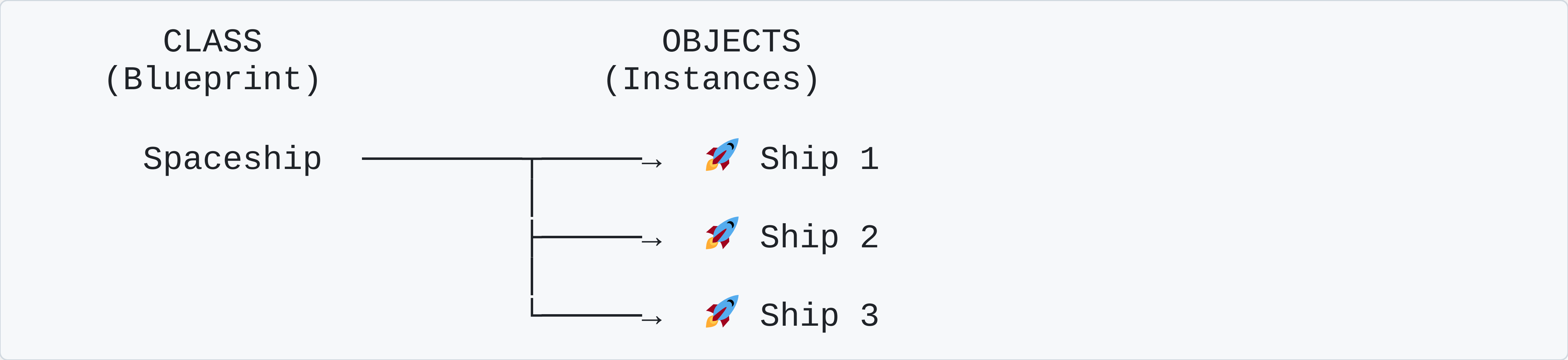
❌ **Without Classes**

- Variables everywhere
- Hard to organize
- Repetitive code
- Difficult to maintain

✅ **With Classes**

- Organized code
- Reusable objects
- Easy to understand
- Scalable

Classes vs Objects



```
     CLASS                   OBJECTS
  (Blueprint)              (Instances)

   Spaceship    ────────┬──────→  🚀 Ship 1
                        │
                        ├──────→  🚀 Ship 2
                        │
                        └──────→  🚀 Ship 3
```

Class = Cookie cutter

Object = The cookies made from it

Part 2: Your First Class

Basic Class Syntax

```python
# Define a class
class Dog:
    pass

# Create an object (instance)
my_dog = Dog()
```

Key Points:

- Class names use `PascalCase` (capitalize each word)
- Use `class` keyword
- Create instances by calling the class like a function

Adding Attributes

```python
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Create dogs with different attributes
my_dog = Dog("Buddy", 3)
your_dog = Dog("Max", 5)

print(my_dog.name)  # Output: Buddy
print(your_dog.age) # Output: 5
```

`__init__` = Constructor (runs when object is created)

`self` = Refers to the instance itself

The `self` Parameter

```python
class Dog:
    def __init__(self, name):
        self.name = name  # Instance variable

    def bark(self):  # self is ALWAYS first parameter
        print(f"{self.name} says Woof!")

dog1 = Dog("Buddy")
dog1.bark()  # Buddy says Woof!
```

`self` lets each instance access its own data

Adding Methods

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        return f"{self.name} says Woof!"

    def birthday(self):
        self.age += 1
        return f"Happy birthday {self.name}! Now {self.age} years old."

my_dog = Dog("Buddy", 3)
print(my_dog.bark())        # Buddy says Woof!
print(my_dog.birthday())    # Happy birthday Buddy! Now 4 years old.
```

Part 3: The Spaceship Class

Planning Our Spaceship

What does a spaceship need?

PROPERTIES

- Position (x, y)
- Angle (direction)
- Speed
- Size

BEHAVIORS

- Move forward
- Move backward
- Rotate left
- Rotate right
- Draw itself

Basic Spaceship Class

```
class Spaceship:
    def __init__(self, x, y):
        self.x = x          # Position
        self.y = y
        self.angle = 0      # Direction (radians)
        self.speed = 0      # Current speed
        self.size = 20      # Size of ship

    def display(self):
        # Draw the spaceship (we'll implement this soon)
        pass

# Create a spaceship at center of screen
ship = Spaceship(400, 300)
```

Spaceship in Py5 Sketch

```python
import py5

class Spaceship:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.angle = 0
        self.speed = 0
        self.size = 20

ship = None

def setup():
    global ship
    py5.size(800, 600)
    ship = Spaceship(400, 300)  # Create ship at center

def draw():
    py5.background(0)
    # We'll draw the ship here

py5.run_sketch()
```

Part 4: Movement & Rotation

Adding Rotation Methods

```python
class Spaceship:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.angle = 0
        self.speed = 0
        self.size = 20
        self.rotation_speed = 0.1  # How fast to rotate

    def rotate_left(self):
        self.angle -= self.rotation_speed

    def rotate_right(self):
        self.angle += self.rotation_speed
```

Rotation changes the angle!

Adding Thrust (Forward Movement)

```
class Spaceship:
    # ... (previous code)

    def thrust(self):
        # Increase speed
        self.speed += 0.5
        # Limit maximum speed
        if self.speed > 5:
            self.speed = 5

    def reverse(self):
        # Decrease speed
        self.speed -= 0.5
        # Limit minimum speed
        if self.speed < -3:
            self.speed = -3
```

Update Method (Apply Movement)

```
import py5

class Spaceship:
    # ... (previous code)

    def update(self):
        # Move in the direction we're facing
        self.x += py5.cos(self.angle) * self.speed
        self.y += py5.sin(self.angle) * self.speed

        # Friction (gradually slow down)
        self.speed *= 0.99
```

Trigonometry:

- `cos(angle)` gives X direction
- `sin(angle)` gives Y direction

Screen Wrapping

```python
class Spaceship:
    # ... (previous code)

    def update(self):
        # Move
        self.x += py5.cos(self.angle) * self.speed
        self.y += py5.sin(self.angle) * self.speed

        # Friction
        self.speed *= 0.99

        # Wrap around screen edges
        if self.x > py5.width:
            self.x = 0
        elif self.x < 0:
            self.x = py5.width

        if self.y > py5.height:
            self.y = 0
        elif self.y < 0:
            self.y = py5.height
```

Part 5: Drawing the Spaceship

Drawing a Triangle Spaceship

```
class Spaceship:
    # ... (previous code)

    def display(self):
        py5.push_matrix()
        py5.translate(self.x, self.y)
        py5.rotate(self.angle)

        # Draw triangle pointing right
        py5.fill(255)
        py5.stroke(255)
        py5.stroke_weight(2)
        py5.triangle(self.size, 0,           # Nose (front)
                     -self.size, -self.size/2,  # Back left
                     -self.size, self.size/2)   # Back right

        py5.pop_matrix()
```

Spaceship Shape Diagram

```
            • (size, 0)
           /|
          / |
         /  |    ← Direction of travel
        /   |
       /    |
      •─────• (center at origin)
(-size, -size/2)  (-size, size/2)
```

Triangle points RIGHT in local coordinates

Rotation makes it face any direction

Adding Thrust Visual

```python
class Spaceship:
    # ... (previous code)

    def display(self):
        py5.push_matrix()
        py5.translate(self.x, self.y)
        py5.rotate(self.angle)

        # Draw ship
        py5.fill(255)
        py5.triangle(self.size, 0,
                     -self.size, -self.size/2,
                     -self.size, self.size/2)

        # Draw flame when moving
        if abs(self.speed) > 0.5:
            py5.fill(255, 150, 0)  # Orange flame
            py5.triangle(-self.size, 0,
                         -self.size - 10, -5,
                         -self.size - 10, 5)

        py5.pop_matrix()
```

Part 6: User Control

Connecting Keyboard Input

```python
import py5

ship = None

def setup():
    global ship
    py5.size(800, 600)
    ship = Spaceship(400, 300)

def draw():
    py5.background(0)
    ship.update()
    ship.display()

def key_pressed():
    if py5.key == 'w' or py5.key_code == py5.UP:
        ship.thrust()
    elif py5.key == 's' or py5.key_code == py5.DOWN:
        ship.reverse()
    elif py5.key == 'a' or py5.key_code == py5.LEFT:
        ship.rotate_left()
    elif py5.key == 'd' or py5.key_code == py5.RIGHT:
        ship.rotate_right()

py5.run_sketch()
```

Continuous Control (Better!)

```
def draw():
    py5.background(0)

    # Check keys every frame for smooth control
    if py5.is_key_pressed:
        if py5.key == 'w' or py5.key_code == py5.UP:
            ship.thrust()
        elif py5.key == 's' or py5.key_code == py5.DOWN:
            ship.reverse()

        if py5.key == 'a' or py5.key_code == py5.LEFT:
            ship.rotate_left()
        elif py5.key == 'd' or py5.key_code == py5.RIGHT:
            ship.rotate_right()

    ship.update()
    ship.display()
```

Complete Spaceship Class (So Far)

```python
import py5

class Spaceship:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.angle = 0
        self.speed = 0
        self.size = 20
        self.rotation_speed = 0.1

    def rotate_left(self):
        self.angle -= self.rotation_speed

    def rotate_right(self):
        self.angle += self.rotation_speed

    def thrust(self):
        self.speed += 0.5
        if self.speed > 5:
            self.speed = 5
```

Complete Spaceship Class (Continued)

```
    def reverse(self):
        self.speed -= 0.5
        if self.speed < -3:
            self.speed = -3

    def update(self):
        self.x += py5.cos(self.angle) * self.speed
        self.y += py5.sin(self.angle) * self.speed
        self.speed *= 0.99

        # Screen wrapping
        if self.x > py5.width: self.x = 0
        elif self.x < 0: self.x = py5.width
        if self.y > py5.height: self.y = 0
        elif self.y < 0: self.y = py5.height
```

Complete Spaceship Class (Display)

```
    def display(self):
        py5.push_matrix()
        py5.translate(self.x, self.y)
        py5.rotate(self.angle)

        # Ship body
        py5.fill(255)
        py5.stroke(255)
        py5.stroke_weight(2)
        py5.triangle(self.size, 0,
                     -self.size, -self.size/2,
                     -self.size, self.size/2)

        # Thrust flame
        if abs(self.speed) > 0.5:
            py5.fill(255, 150, 0)
            py5.triangle(-self.size, 0,
                         -self.size - 10, -5,
                         -self.size - 10, 5)

        py5.pop_matrix()
```

Part 7: Multiple Objects

Why Multiple Objects?

One Class Definition → Many Objects

```
# Create multiple spaceships
ship1 = Spaceship(200, 300)
ship2 = Spaceship(600, 300)
ship3 = Spaceship(400, 150)

# Each has its own position, angle, speed, etc.
ship1.thrust()  # Only affects ship1
ship2.rotate_left()  # Only affects ship2
```

Each object is independent!

Asteroid Class Example

```python
class Asteroid:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.size = py5.random(20, 50)
        self.vx = py5.random(-2, 2)  # Random velocity
        self.vy = py5.random(-2, 2)
        self.rotation = 0
        self.rotation_speed = py5.random(-0.1, 0.1)

    def update(self):
        self.x += self.vx
        self.y += self.vy
        self.rotation += self.rotation_speed

        # Wrap around screen
        if self.x > py5.width: self.x = 0
        elif self.x < 0: self.x = py5.width
        if self.y > py5.height: self.y = 0
        elif self.y < 0: self.y = py5.height
```

Drawing Asteroids

```python
class Asteroid:
    # ... (previous code)

    def display(self):
        py5.push_matrix()
        py5.translate(self.x, self.y)
        py5.rotate(self.rotation)

        py5.fill(100)
        py5.stroke(150)
        py5.stroke_weight(2)

        # Draw irregular polygon
        py5.begin_shape()
        for i in range(8):
            angle = py5.TWO_PI / 8 * i
            r = self.size * py5.random(0.8, 1.2)
            px = py5.cos(angle) * r
            py5_y = py5.sin(angle) * r
            py5.vertex(px, py5_y)
        py5.end_shape(py5.CLOSE)

        py5.pop_matrix()
```

Managing Multiple Objects with Lists

```
import py5

ship = None
asteroids = []

def setup():
    global ship, asteroids
    py5.size(800, 600)

    ship = Spaceship(400, 300)

    # Create 5 asteroids
    for i in range(5):
        x = py5.random(py5.width)
        y = py5.random(py5.height)
        asteroids.append(Asteroid(x, y))

def draw():
    py5.background(0)

    # Update and display all asteroids
    for asteroid in asteroids:
        asteroid.update()
        asteroid.display()

    ship.update()
    ship.display()

py5.run_sketch()
```

Adding Collision Detection

```python
class Spaceship:
    # ... (previous code)

    def hits(self, asteroid):
        # Calculate distance between ship and asteroid
        d = py5.dist(self.x, self.y, asteroid.x, asteroid.y)

        # Check if distance is less than sum of radii
        if d < self.size + asteroid.size:
            return True
        return False

# In draw():
def draw():
    py5.background(0)

    for asteroid in asteroids:
        asteroid.update()
        asteroid.display()

        if ship.hits(asteroid):
            py5.fill(255, 0, 0)
            py5.text("COLLISION!", 350, 300)

    ship.update()
    ship.display()
```

Part 8: Inheritance

What is Inheritance?

Create new classes based on existing ones

```
        Spaceship (Parent/Base)
              ↓
    ┌─────────┴─────────┐
    ↓                   ↓
PlayerShip          EnemyShip
(Child)            (Child)
```

Child classes inherit properties and methods from parent

Child classes can add new features or override existing ones

Basic Inheritance Syntax

```python
# Parent class
class Spaceship:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.speed = 0

    def move(self):
        self.x += self.speed

# Child class
class PlayerShip(Spaceship):  # Inherits from Spaceship
    def __init__(self, x, y):
        super().__init__(x, y)  # Call parent constructor
        self.lives = 3  # Add new property

    def lose_life(self):  # Add new method
        self.lives -= 1
```

Enemy Spaceship Example

```python
class EnemyShip(Spaceship):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.color = (255, 0, 0)  # Red
        self.ai_timer = 0

    def ai_update(self):
        # Simple AI: Random movements
        self.ai_timer += 1

        if self.ai_timer > 60:  # Every 60 frames
            choice = int(py5.random(4))
            if choice == 0:
                self.thrust()
            elif choice == 1:
                self.rotate_left()
            elif choice == 2:
                self.rotate_right()

            self.ai_timer = 0
```

Override Display Method

```
class EnemyShip(Spaceship):
    # ... (previous code)

    def display(self):
        # Different appearance than regular spaceship
        py5.push_matrix()
        py5.translate(self.x, self.y)
        py5.rotate(self.angle)

        # Red enemy ship
        py5.fill(255, 0, 0)
        py5.stroke(255, 100, 100)
        py5.stroke_weight(2)
        py5.triangle(self.size, 0,
                     -self.size, -self.size/2,
                     -self.size, self.size/2)

        py5.pop_matrix()
```

Overriding: Replace parent method with new version

Using Inheritance in Game

```python
import py5

player = None
enemies = []

def setup():
    global player, enemies
    py5.size(800, 600)

    player = Spaceship(400, 300)  # Player ship

    # Create 3 enemy ships
    for i in range(3):
        x = py5.random(py5.width)
        y = py5.random(py5.height)
        enemies.append(EnemyShip(x, y))

def draw():
    py5.background(0)

    # Update enemies with AI
    for enemy in enemies:
        enemy.ai_update()
        enemy.update()
        enemy.display()

    # Player controls
    # ... (handle input)
    player.update()
    player.display()

py5.run_sketch()
```

Part 9: Complete Game

Game Architecture

Components:

1. **Spaceship class** - Player ship
2. **EnemyShip class** - AI-controlled ships
3. **Asteroid class** - Obstacles
4. **Bullet class** - Projectiles
5. **Game state** - Score, lives, level

Each is a separate class!

Bullet Class

```
class Bullet:
    def __init__(self, x, y, angle):
        self.x = x
        self.y = y
        self.vx = py5.cos(angle) * 10
        self.vy = py5.sin(angle) * 10
        self.lifespan = 60  # Frames before disappearing

    def update(self):
        self.x += self.vx
        self.y += self.vy
        self.lifespan -= 1

    def is_dead(self):
        return self.lifespan <= 0

    def display(self):
        py5.fill(255, 255, 0)
        py5.no_stroke()
        py5.circle(self.x, self.y, 5)
```

Shooting Method for Spaceship

```python
class Spaceship:
    # ... (previous code)

    def shoot(self):
        # Calculate bullet starting position (nose of ship)
        bullet_x = self.x + py5.cos(self.angle) * self.size
        bullet_y = self.y + py5.sin(self.angle) * self.size

        return Bullet(bullet_x, bullet_y, self.angle)

# In main code:
bullets = []

def key_pressed():
    if py5.key == ' ':  # Space to shoot
        bullets.append(ship.shoot())
```

Managing Bullets

```
def draw():
    py5.background(0)

    # Update and draw bullets
    for bullet in bullets[:]:  # Copy list to avoid issues
        bullet.update()
        bullet.display()

        # Remove dead bullets
        if bullet.is_dead():
            bullets.remove(bullet)

    # Check bullet collisions with asteroids
    for bullet in bullets[:]:
        for asteroid in asteroids[:]:
            d = py5.dist(bullet.x, bullet.y, asteroid.x, asteroid.y)
            if d < asteroid.size:
                bullets.remove(bullet)
                asteroids.remove(asteroid)
                # Add score, etc.
                break
```

Game State Class

```python
class Game:
    def __init__(self):
        self.score = 0
        self.lives = 3
        self.level = 1
        self.game_over = False

    def add_score(self, points):
        self.score += points

    def lose_life(self):
        self.lives -= 1
        if self.lives <= 0:
            self.game_over = True

    def display_hud(self):
        py5.fill(255)
        py5.text_size(20)
        py5.text(f"Score: {self.score}", 10, 30)
        py5.text(f"Lives: {self.lives}", 10, 60)
        py5.text(f"Level: {self.level}", 10, 90)
```

Complete Game Structure

```
import py5

# Game objects
game = None
player = None
enemies = []
asteroids = []
bullets = []

def setup():
    global game, player, enemies, asteroids
    py5.size(800, 600)

    game = Game()
    player = Spaceship(400, 300)

    # Initialize enemies and asteroids
    for i in range(3):
        enemies.append(EnemyShip(py5.random(py5.width),
                                  py5.random(py5.height)))

    for i in range(5):
        asteroids.append(Asteroid(py5.random(py5.width),
                                    py5.random(py5.height)))
```

Complete Game Loop

```
def draw():
    py5.background(0, 0, 20)  # Dark blue space

    if not game.game_over:
        # Update and display all objects
        update_bullets()
        update_enemies()
        update_asteroids()
        check_collisions()

        player.update()
        player.display()

        game.display_hud()
    else:
        display_game_over()

def display_game_over():
    py5.text_size(48)
    py5.fill(255, 0, 0)
    py5.text_align(py5.CENTER)
    py5.text("GAME OVER", py5.width/2, py5.height/2)
    py5.text_size(24)
    py5.text(f"Final Score: {game.score}", py5.width/2, py5.height/2 + 50)

py5.run_sketch()
```

Helper Functions

```python
def update_bullets():
    for bullet in bullets[:]:
        bullet.update()
        bullet.display()
        if bullet.is_dead():
            bullets.remove(bullet)

def update_enemies():
    for enemy in enemies:
        enemy.ai_update()
        enemy.update()
        enemy.display()

def update_asteroids():
    for asteroid in asteroids:
        asteroid.update()
        asteroid.display()

def check_collisions():
    # Check bullet-asteroid collisions
    for bullet in bullets[:]:
        for asteroid in asteroids[:]:
            if hit(bullet, asteroid):
                bullets.remove(bullet)
                asteroids.remove(asteroid)
                game.add_score(10)
                break
```

Part 10: Best Practices

Class Design Principles

DO ✅

- One class, one purpose
- Use meaningful names
- Keep methods small
- Use self consistently

DON'T ❌

- Mix unrelated features
- Make huge classes
- Forget self parameter
- Use globals excessively

Organizing Your Code

```
# Good structure:

# 1. Class definitions at top
class Spaceship:
    # ...

class Asteroid:
    # ...

class Bullet:
    # ...

# 2. Global variables
ship = None
asteroids = []

# 3. Setup and draw
def setup():
    # ...

def draw():
    # ...

# 4. Helper functions
def check_collisions():
    # ...

# 5. Event handlers
def key_pressed():
    # ...

py5.run_sketch()
```

Common Mistakes

1. Forgetting `self`

```
# Wrong:
class Spaceship:
    def move(self):
        x += 1  # ❌ What is x?

# Right:
class Spaceship:
    def move(self):
        self.x += 1  # ✅ Instance variable
```

2. Not calling `super().__init__()`

```
# Wrong:
class EnemyShip(Spaceship):
    def __init__(self, x, y):
        self.color = (255, 0, 0)  # ❌ Parent not initialized
```

Debugging Tips

```
# Add __repr__ for better printing
class Spaceship:
    def __repr__(self):
        return f"Spaceship(x={self.x:.1f}, y={self.y:.1f}, " \
               f"angle={self.angle:.2f})"

ship = Spaceship(100, 200)
print(ship)  # Spaceship(x=100.0, y=200.0, angle=0.00)

# Add debug display
class Spaceship:
    def display_debug(self):
        py5.fill(255)
        py5.text(f"Speed: {self.speed:.2f}", self.x, self.y - 30)
        py5.text(f"Angle: {self.angle:.2f}", self.x, self.y - 50)
```

Performance Tips

1. Don't create objects in draw():

```
# Bad:
def draw():
    ship = Spaceship(400, 300)  # ❌ Creates new object every frame

# Good:
ship = None
def setup():
    global ship
    ship = Spaceship(400, 300)  # ✅ Create once
```

2. Remove objects you don't need:

```
# Remove off-screen bullets
for bullet in bullets[:]:
    if bullet.x < 0 or bullet.x > py5.width:
        bullets.remove(bullet)
```

Advanced: Class Variables

```python
class Spaceship:
    # Class variable (shared by all instances)
    total_ships = 0

    def __init__(self, x, y):
        self.x = x  # Instance variable (unique to each)
        self.y = y
        Spaceship.total_ships += 1  # Increment class variable

    @classmethod
    def get_total_ships(cls):
        return cls.total_ships

# Usage:
ship1 = Spaceship(100, 100)
ship2 = Spaceship(200, 200)
print(Spaceship.get_total_ships())  # Output: 2
```

Advanced: Static Methods

```
class MathUtils:
    @staticmethod
    def distance(x1, y1, x2, y2):
        return ((x2 - x1)**2 + (y2 - y1)**2)**0.5

    @staticmethod
    def angle_between(x1, y1, x2, y2):
        return py5.atan2(y2 - y1, x2 - x1)

# Usage (no instance needed):
d = MathUtils.distance(0, 0, 100, 100)
angle = MathUtils.angle_between(50, 50, 200, 100)
```

Part 11: Extensions & Ideas

Power-Ups Class

```python
class PowerUp:
    def __init__(self, x, y, type):
        self.x = x
        self.y = y
        self.type = type  # "speed", "shield", "weapon"
        self.size = 15
        self.lifetime = 300  # Frames

    def update(self):
        self.lifetime -= 1

    def is_expired(self):
        return self.lifetime <= 0

    def display(self):
        if self.type == "speed":
            py5.fill(0, 255, 0)
        elif self.type == "shield":
            py5.fill(0, 0, 255)
        elif self.type == "weapon":
            py5.fill(255, 255, 0)

        py5.circle(self.x, self.y, self.size * 2)
```

Particle System

```python
class Particle:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.vx = py5.random(-2, 2)
        self.vy = py5.random(-2, 2)
        self.life = 255  # Alpha value

    def update(self):
        self.x += self.vx
        self.y += self.vy
        self.life -= 5

    def is_dead(self):
        return self.life <= 0

    def display(self):
        py5.fill(255, 150, 0, self.life)
        py5.no_stroke()
        py5.circle(self.x, self.y, 5)

# Create explosion when asteroid is hit
def create_explosion(x, y):
    for i in range(20):
        particles.append(Particle(x, y))
```

Boss Enemy

```python
class BossShip(EnemyShip):
    def __init__(self, x, y):
        super().__init__(x, y)
        self.size = 40  # Bigger
        self.health = 10  # Multiple hits needed
        self.shoot_timer = 0

    def ai_update(self):
        # More aggressive AI
        # Calculate angle to player
        angle_to_player = py5.atan2(player.y - self.y,
                                     player.x - self.x)

        # Rotate toward player
        angle_diff = angle_to_player - self.angle
        if abs(angle_diff) > 0.1:
            if angle_diff > 0:
                self.rotate_right()
            else:
                self.rotate_left()

        # Shoot occasionally
        self.shoot_timer += 1
        if self.shoot_timer > 90:
            return self.shoot()  # Return bullet
            self.shoot_timer = 0
```

Project Ideas

Beginner:

- Simple shooter
- Asteroids clone
- Two-player game
- Obstacle course

Advanced:

- Tower defense
- Top-down racer
- Space exploration
- Strategy game

Enhancement Ideas

✨ **Add These Features:**

- Health bars above ships
- Different weapon types
- Upgradeable ships
- Sound effects
- Background music
- Particle trails
- Screen shake on collision
- Score multipliers
- Boss battles
- Level progression

Summary

What We Learned

✅ **Classes** organize code into objects with properties and methods

✅ `__init__` initializes objects when created

✅ `self` refers to the instance

✅ **Methods** are functions inside classes

✅ **Inheritance** creates specialized versions of classes

✅ **Multiple objects** from one class definition

✅ **OOP** makes code organized and reusable

Key Concepts Recap

Class Structure:

```
class ClassName:
    def __init__(self, param1, param2):  # Constructor
        self.property1 = param1           # Properties
        self.property2 = param2

    def method_name(self):                # Methods
        # Do something with self.properties
        pass
```

Our Spaceship Game Has:

Classes

Spaceship

EnemyShip

Asteroid

Bullet

Game

Concepts

Inheritance

Methods

Properties

Lists

Collisions

Features

Movement

Next Steps

Practice Building:

1. Modify the spaceship class
2. Add new enemy types
3. Create different weapons
4. Build a complete game
5. Share with friends!

Learn More:

- Composition over inheritance
- Design patterns
- Unit testing
- Documentation

Resources

- **Py5 Documentation:** https://py5coding.org
- **Python OOP Tutorial:** https://docs.python.org/3/tutorial/classes.html
- **Game Programming Patterns:** http://gameprogrammingpatterns.com

Books:

- "Python Crash Course" by Eric Matthes
- "Invent Your Own Computer Games with Python" by Al Sweigart

Thank You! 🚀

Questions?

Get Started:

```
pip install py5
```

Build your own spaceship game!

Control scheme:

- W/↑ = Forward
- S/↓ = Reverse
- A/← = Rotate left
- D/→ = Rotate right
- SPACE = Shoot