Class: A definition of a new type, a collection of functions (methods), create variables, and be the inheritance of another class.

Instance (object): a variable built using a class as a blueprint, has its own copy of non-static attributes in methods, and has access to the methods of the class. Self: Is the reference to the object that is calling this method. Method Examples (dunder): \_\_init\_\_ \_\_str\_\_ \_\_index\_\_

Mutability: can modify elements, add/remove elements. Dictionaries: an unordered mutable sequence

Parameter: A variable that will hold a value when called and goes away when the function ends.

Argument: A value that is passed to the function and used to initialize the parameter.

Example of class

class Particle(object):  
 *"""This is a simple image-based (color fadeout and shape) Particle."""*

Example of method:

def \_\_init\_\_(self, x, y, velocity\_x, velocity\_y, bounding\_radius, lifetime, gradient\_img, particle\_image

Constructor:

self.mImageReference = gradient\_img  
self.mPosition = [x, y]  
self.mVelocity = [velocity\_x, velocity\_y]  
self.mLifetime = lifetime  
self.mMaxLifetime = lifetime  
self.mBoundingRadius = bounding\_radius  
self.mDrawImage = particle\_image  
self.mGradientImage = gradient\_img  
self.mOrientation = random.randint(0, 360)  
self.mRotationSpeed = random.choice((1, -1)) \* random.uniform(10, 180)

Function example:

def update(self, dt, gravity\_acceleration, bounds):

Function Update information

self.mVelocity[1] += gravity\_acceleration \* dt  
self.mPosition[0] += self.mVelocity[0] \* dt  
self.mPosition[1] += self.mVelocity[1] \* dt  
self.mOrientation += self.mRotationSpeed \* dt  
self.mLifetime -= dt  
if self.mLifetime <= 0 or self.mPosition[0] < bounds[0] - self.mBoundingRadius or \  
 self.mPosition[0] > bounds[0] + bounds[2] + self.mBoundingRadius or \  
 self.mPosition[1] < bounds[1] - self.mBoundingRadius or \  
 self.mPosition[1] > bounds[1] + bounds[3] + self.mBoundingRadius:  
 return True  
return False

def draw(self, surf):  
 percent = self.mLifetime / self.mMaxLifetime  
 alpha = int(percent \* 255)  
 gw = self.mGradientImage.get\_width() - 1  
 gx = int((1.0 - percent) \* gw)  
 color = self.mGradientImage.get\_at((gx, 0))  
 temp\_img = pygame.transform.rotate(self.mDrawImage, self.mOrientation)  
 temp\_img.fill(color, special\_flags=pygame.BLEND\_RGB\_MULT)  
 temp\_img.set\_alpha(alpha)  
 surf.blit(temp\_img, (self.mPosition[0] - temp\_img.get\_width() / 2,  
 self.mPosition[1] - temp\_img.get\_height() / 2))  
def \_\_str\_\_(self):s = f"Particle [{(round(self.mPosition[0], 1), round(self.mPosition[1]), 1)}"  
 s += f" ori={round(self.mOrientation, 1)} life={round(self.mLifetime,1)}/"  
 s += f"{self.mMaxLifetime}]"  
 return s

Triangle Math and player example:

def polar\_to\_rectangular(radians, hypotenuse=1.0, inverted\_y=True):  
adjacent = hypotenuse \* math.cos(radians)  
 opposite = hypotenuse \* math.sin(radians)  
 if inverted\_y:  
 opposite = -opposite  
 return (adjacent, opposite)  
def polar\_to\_rectangularD(degrees, hypotenuse=1.0, inverted\_y=True):  
radians = math.radians(degrees)  
 return polar\_to\_rectangular(radians, hypotenuse, inverted\_y)  
def distance(x1, y1, x2, y2):  
a = x1 - x2  
 b = y1 - y2  
 return (a \*\* 2 + b \*\* 2) \*\* 0.5  
def angle\_towards(start\_x, start\_y, target\_x, target\_y, inverted\_y=True):  
adjacent = target\_x - start\_x  
 opposite = target\_y - start\_y  
 if inverted\_y:  
 opposite = -opposite  
 return math.atan2(opposite, adjacent)  
def angle\_towardsD(start\_x, start\_y, target\_x, target\_y, inverted\_y=True):  
return math.degrees(angle\_towards(start\_x, start\_y, target\_x, target\_y, inverted\_y))  
def tinted\_sprite(orig\_img, color\_multiplier, add\_alpha=True):  
new\_img = pygame.Surface(orig\_img.get\_size())  
 if add\_alpha:  
 new\_img = new\_img.convert\_alpha()  
 new\_img.fill((0, 0, 0, 0)) # Fills the (blank) copy with transparency  
 new\_img.blit(orig\_img, (0, 0)) # Now new\_img is a full copy of the original  
 new\_img.fill(color\_multiplier, special\_flags=pygame.BLEND\_RGB\_MULT)  
 return new\_img  
def draw\_hp\_bar(surf, area, percent, fill\_color, is\_horizontal = True, outline\_color = None):  
if outline\_color is not None:  
 pygame.draw.rect(surf, outline\_color, (area[0] - 1, area[1] - 1, area[2] + 2, area[3] + 2), 1)  
 if is\_horizontal:  
 pygame.draw.rect(surf, fill\_color, (area[0], area[1], area[2] \* percent, area[3]))  
 else:  
 pygame.draw.rect(surf, fill\_color, (area[0], area[1], area[2], area[3] \* percent))

Pygame setup Example:

import emitter  
import pygame  
pygame.init()  
win\_width = 800  
win\_height = 600  
win = pygame.display.set\_mode((win\_width, win\_height))  
clock = pygame.time.Clock()  
grad\_img = pygame.image.load("images/gradient.png")  
font\_obj = pygame.font.Font("fonts/PixelifySans-VariableFont\_wght.ttf", 22)  
pygame.quit()

event = pygame.event.poll()  
if event.type == pygame.QUIT:  
 game\_over = True  
elif event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_p:  
 paused = not paused  
if event.type == pygame.MOUSEBUTTONUP and event.button == 1:  
 player["cooldown"] = 0.0  
  
# polling  
keys\_pressed = pygame.key.get\_pressed()  
if keys\_pressed[pygame.K\_ESCAPE]:  
 game\_over = True  
mouse\_pressed = pygame.mouse.get\_pressed()  
mouse\_pos = pygame.mouse.get\_pos()  
move\_direction = [0, 0]  
if keys\_pressed[pygame.K\_a] or keys\_pressed[pygame.K\_LEFT]:  
 move\_direction[0] -= 1  
if keys\_pressed[pygame.K\_d] or keys\_pressed[pygame.K\_RIGHT]:  
 move\_direction[0] += 1  
if keys\_pressed[pygame.K\_w] or keys\_pressed[pygame.K\_UP]:  
 move\_direction[1] -= 1  
if keys\_pressed[pygame.K\_s] or keys\_pressed[pygame.K\_DOWN]:  
 move\_direction[1] += 1