**Day 24:**

**Utilization of VGroup and Other Grouping Methods**

**VGroup**

VGroup is used to group multiple vectorized mobjects (VMobjects) together. This allows you to apply transformations and animations to the entire group simultaneously.

**Example of VGroup:**

**Python**

from manim import \*

class VGroupExample(Scene):

def construct(self):

circle = Circle()

square = Square()

triangle = Triangle()

# Grouping the shapes

shapes = VGroup(circle, square, triangle)

# Arranging the shapes in a row

shapes.arrange(RIGHT, buff=1)

# Applying a transformation to the entire group

self.play(Create(shapes))

self.wait(1)

self.play(shapes.animate.shift(UP))

self.wait(1)

**Group**

Group is similar to VGroup but can include any type of mobject, not just VMobjects. However, Group does not support VMobject-specific methods like set\_stroke or set\_fill.

**Example of Group:**

**Python**

from manim import \*

class GroupExample(Scene):

def construct(self):

text = Text("Hello")

image = ImageMobject("path/to/image.png")

# Grouping the text and image

group = Group(text, image)

# Arranging the group

group.arrange(DOWN, buff=1)

# Applying a transformation to the entire group

self.play(FadeIn(group))

self.wait(1)

self.play(group.animate.shift(LEFT))

self.wait(1)

**Additional Grouping Methods**

* **arrange**: Arranges mobjects in a specified direction with a given buffer space.
* **arrange\_in\_grid**: Arranges mobjects in a grid layout.

**Example of arrange\_in\_grid:**

**Python**

from manim import \*

class GridExample(Scene):

def construct(self):

shapes = VGroup(\*[Square() for \_ in range(9)])

# Arranging shapes in a 3x3 grid

shapes.arrange\_in\_grid(rows=3, cols=3, buff=0.5)

self.play(Create(shapes))

self.wait(1)

**Custom Animation Techniques and Use of Updaters**

**Custom Animation Techniques**

Custom animations can be created by combining different animation methods and using updaters for dynamic behavior.

**Example of Custom Animation:**

**Python**

from manim import \*

class CustomAnimationExample(Scene):

def construct(self):

circle = Circle()

square = Square()

self.play(Create(circle))

self.wait(1)

self.play(Transform(circle, square))

self.wait(1)

self.play(Rotate(square, angle=PI/4))

self.wait(1)

self.play(FadeOut(square))

self.wait(1)

**Updaters**

Updaters are functions that continuously update the properties of mobjects during an animation. They are useful for creating dynamic and interactive animations.

**Example of Updaters:**

**Python**

from manim import \*

class UpdaterExample(Scene):

def construct(self):

dot = Dot()

text = Text("Moving Dot").next\_to(dot, UP)

# Updater function to keep text next to the dot

def update\_text(mob):

mob.next\_to(dot, UP)

text.add\_updater(update\_text)

self.add(dot, text)

self.play(dot.animate.shift(RIGHT \* 4), run\_time=4)

self.wait(1)

**ValueTracker**

ValueTracker is a utility class that helps in tracking and animating numerical values.

**Example of ValueTracker:**

**Python**

from manim import \*

class ValueTrackerExample(Scene):

def construct(self):

tracker = ValueTracker(0)

number = DecimalNumber().add\_updater(lambda d: d.set\_value(tracker.get\_value()))

self.add(number)

self.play(tracker.animate.set\_value(10), run\_time=5)

self.wait(1)