**PASCAL’S TRIANGLE EXPLANATION:**

from manim import \*

class PascalTriangleSymmetry(Scene):

def generate\_pascals\_triangle(self, rows):

triangle = [[1]]

for i in range(1, rows):

new\_row = [0] + triangle[-1] + [0] # Add surrounding 0s

new\_row = [new\_row[j] + new\_row[j + 1] for j in range(len(new\_row) - 1)]

triangle.append(new\_row)

return triangle

def construct(self):

rows = 6 # Stop at row 5 (1 4 6 4 1)

triangle = self.generate\_pascals\_triangle(rows)

# Title

title = Text("Symmetry in Pascal’s Triangle", font\_size=36)

title.to\_edge(UP, buff=0.5)

self.play(FadeIn(title, run\_time=1))

elements = []

y\_start = 1.5

for i, row in enumerate(triangle): # Show only up to row 5

x\_offset = -len(row) / 2

row\_elements = []

for j, num in enumerate(row):

num\_text = MathTex(str(num)).scale(1.2)

num\_text.move\_to(np.array([x\_offset + j, y\_start - i \* 0.7, 0]))

row\_elements.append(num\_text)

elements.append(row\_elements)

full\_group = VGroup(\*[num for row in elements for num in row])

self.play(FadeIn(full\_group, run\_time=0.8))

self.wait(0.3)

# Show first 4 rows step by step, then last row normally

for i in range(4):

row = elements[i]

for num in row:

self.play(Indicate(num, scale\_factor=1.2, run\_time=0.1))

if i > 0:

prev\_row = elements[i - 1]

for j in range(len(prev\_row)):

arrow = Arrow(prev\_row[j].get\_center(), row[j].get\_center(), buff=0.2, color=WHITE)

self.play(GrowArrow(arrow, run\_time=0.3))

self.wait(0.3) # Small pause before last row appears normally

self.play(\*[Indicate(num, scale\_factor=1.2, run\_time=0.1) for num in elements[4]])

# Apply highlights for symmetry

highlight\_anims = []

for i, row in enumerate(elements):

for j in range(len(row) // 2):

row[j].set\_color(BLUE)

row[-(j + 1)].set\_color(RED)

highlight\_anims.append(Indicate(row[j], scale\_factor=1.2, run\_time=0.1))

highlight\_anims.append(Indicate(row[-(j + 1)], scale\_factor=1.2, run\_time=0.1))

self.play(\*highlight\_anims, run\_time=0.6)

# Add "and so on..." text

etc\_text = Text("... and so on", font\_size=28, color=YELLOW)

etc\_text.next\_to(elements[-1][-2], DOWN, buff=0.5)

self.play(FadeIn(etc\_text), run\_time=0.8)

# Final Message

message = Text("Each number on the left mirrors the number on the right!", font\_size=24, color=YELLOW)

message.next\_to(full\_group, DOWN, buff=1.0)

self.play(Write(message), run\_time=0.8)

self.wait(1)

**EDGE CASE ZER0:**

from manim import \*

class PascalTriangleStart(Scene):

    def construct(self):

        # Define colors

        faded\_color = GRAY

        main\_color = WHITE

        # Create Pascal's Triangle first few rows

        pascal\_rows = [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]

        elements = []

        for i, row in enumerate(pascal\_rows):

            row\_elements = []

            for j, num in enumerate(row):

                tex = MathTex(str(num)).scale(1.5)

                tex.move\_to(RIGHT \* (j - i / 2) \* 1.2 + DOWN \* i \* 1.2)

                row\_elements.append(tex)

                self.add(tex)

            elements.append(row\_elements)

        self.wait(2)

        # Create a VGroup for the full Pascal's Triangle

        triangle\_group = VGroup(\*[num for row in elements for num in row])

        # Zoom into the top "1" while fading the background triangle

        one = elements[0][0]

        self.play(triangle\_group.animate.set\_opacity(0.2), one.animate.scale(3))  # Fade and zoom effect

        self.wait(1)

        # Position faded "0"s around it

        left\_zero = MathTex("0", color=faded\_color).scale(2).next\_to(one, LEFT, buff=0.7)

        right\_zero = MathTex("0", color=faded\_color).scale(2).next\_to(one, RIGHT, buff=0.7)

        # Animate faded "0"s appearing

        self.play(FadeIn(left\_zero, shift=LEFT), FadeIn(right\_zero, shift=RIGHT))

        # Explanation text about adding zeroes

        explanation\_text = Tex(

            "The zeroes act as placeholders, ensuring the addition rule stays consistent.\\\\"

            "Each number is the sum of the two above it.\\\\"

            "The edges stay 1 because 1 + 0 = 1 and 0 + 1 = 1."

        ).scale(0.8)

        explanation\_text.next\_to(one, UP, buff=1.5)

        self.play(Write(explanation\_text))

        self.wait(2)

**SYMMETRY PROPERTY:**

from manim import \*

class PascalTriangleSymmetry(Scene):

    def generate\_pascals\_triangle(self, rows):

        triangle = [[1]]

        for i in range(1, rows):

            new\_row = [1]

            for j in range(len(triangle[-1]) - 1):

                new\_row.append(triangle[-1][j] + triangle[-1][j + 1])

            new\_row.append(1)

            triangle.append(new\_row)

        return triangle

    def construct(self):

        rows = 6  # Keep only 6 rows

        triangle = self.generate\_pascals\_triangle(rows)

        # Title

        title = Text("Symmetry in Pascal’s Triangle", font\_size=36)

        title.to\_edge(UP, buff=0.5)  # Move title slightly down

        self.play(Write(title))

        elements = []

        y\_start = 1.5  # Move triangle downward

        for i, row in enumerate(triangle):

            x\_offset = -len(row) / 2  # Adjust to center each row

            row\_elements = []

            for j, num in enumerate(row):

                num\_text = Text(str(num), font\_size=30, color=WHITE, font="Times New Roman")  # Font adjusted

                num\_text.move\_to(np.array([x\_offset + j, y\_start - i \* 0.7, 0]))  # Better spacing

                row\_elements.append(num\_text)

            elements.append(row\_elements)

        full\_group = VGroup(\*[num for row in elements for num in row])

        self.play(FadeIn(full\_group, run\_time=2))  # Faded-in motion effect

        self.wait(1)

        # Apply highlights for symmetry (slower effect)

        highlight\_anims = []

        for i, row in enumerate(elements):

            for j in range(len(row) // 2):

                row[j].set\_color(BLUE)

                row[-(j + 1)].set\_color(RED)

                highlight\_anims.append(Indicate(row[j], scale\_factor=1.2, run\_time=0.8))

                highlight\_anims.append(Indicate(row[-(j + 1)], scale\_factor=1.2, run\_time=0.8))

        self.play(\*highlight\_anims, run\_time=2.5)  # Adjusted for better timing

        # Final Message (Properly aligned)

        message = Text("Each number on the left mirrors the number on the right!", font\_size=24, color=YELLOW)

        message.next\_to(full\_group, DOWN, buff=0.75)  # Ensuring it's placed BELOW the triangle

        self.play(Write(message))

        self.wait(2)  # Pause before ending

from manim import \*

from math import comb

class PascalBlackBackground(Scene):

    def construct(self):

        # Pure black background

        self.camera.background\_color = BLACK

        # Set Times New Roman for all text

        Text.set\_default(font="Times New Roman")

        # Explanation text (white on black)

        explanation = Text(

            "The resultant numbers in each row correspond to Pascal's Triangle,\n"

            "where each number represents the number of ways to reach\n"

            "that position from the top.",

            font\_size=30,

            color=WHITE,

            line\_spacing=1.2

        )

        explanation.to\_edge(UP, buff=0.6)

        self.play(FadeIn(explanation))

        self.wait(1)

        # Create Pascal's Triangle (5 rows)

        rows = 5

        triangle = VGroup()

        # Position numbers in triangle pattern

        for n in range(rows):

            for k in range(n+1):

                num = comb(n, k)

                num\_text = Text(str(num), font\_size=32, color=WHITE)

                position = DOWN\*0.5 + UP\*(2-n)\*0.7 + RIGHT\*(k-n/2)\*1.0

                num\_text.move\_to(position)

                triangle.add(num\_text)

        # Fade in triangle

        self.play(FadeIn(triangle, shift=UP\*0.3, lag\_ratio=0.1))

        self.wait(0.5)

        # Create slim yellow arrows (top-down)

        arrows = VGroup()

        for n in range(1, rows):

            for k in range(n+1):

                # Left parent connection

                if k > 0:

                    arrow = Arrow(

                        start=triangle[(n-1)\*n//2 + k-1].get\_bottom(),

                        end=triangle[n\*(n+1)//2 + k].get\_top(),

                        color=YELLOW,

                        buff=0.1,

                        stroke\_width=2,

                        tip\_length=0.12,

                        max\_tip\_length\_to\_length\_ratio=0.2

                    )

                    arrows.add(arrow)

                # Right parent connection

                if k < n:

                    arrow = Arrow(

                        start=triangle[(n-1)\*n//2 + k].get\_bottom(),

                        end=triangle[n\*(n+1)//2 + k].get\_top(),

                        color=YELLOW,

                        buff=0.1,

                        stroke\_width=2,

                        tip\_length=0.12,

                        max\_tip\_length\_to\_length\_ratio=0.2

                    )

                    arrows.add(arrow)

        # Animate arrows appearing from top to bottom

        self.play(LaggedStart(

            \*[GrowArrow(a) for a in sorted(arrows, key=lambda x: -x.get\_start()[1])],

            lag\_ratio=0.15,

            run\_time=2

        ))

        # Final highlight

        self.play(

            triangle.animate.set\_color(YELLOW),

            arrows.animate.set\_opacity(0.7),

            run\_time=1.5

        )

        self.wait(3)