```
# File: metasurface_presentation.py
from manim import *

class TitleSlide2(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Title text with black color
        title = Text("Meta-Surface Design", font_size=64, color=BLACK)
        subtitle = Text("An Overview of Theory, Applications, and Simulations", font_size=32, color=BLACK)

        # Positioning text
        title.shift(UP*0.5)
        subtitle.next_to(title, DOWN*1.5, buff=0.5)

        # Display title and subtitle
        self.play(Write(title))
        #self.play(Circumscribe(title, color=DARK_BLUE))
        self.play(Write(subtitle))
        #self.play(Circumscribe(subtitle, color=DARK_BLUE))


        # Wait for a few seconds before transitioning to the next scene
        self.wait(3)

# File: metasurface_theory_section.py
from manim import *
```

```
class OutlineSlide(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Create the section title text
        section_title = Text("Outline", font_size=48, color=BLACK)
        section_title.to_edge(UP)

        # List of slides for the section
        slides = [
            "Theoretical Explanations",
            "Historical Parts",
            "Applications",
            "Python Code: Automated Structures in HFSS",
            "HFSS Simulations",
            "Results"
        ]

        # Create section title
        self.play(Write(section_title, run_time=0.75))
        self.wait(1)

        # Display each slide title with animations
        outline_items = []
        for i, slide in enumerate(slides):
            # Create a rectangle and number text
            rect = Rectangle(width=0.4, height=0.4, color=BLACK)
            number = Text(f"{i + 1}", font_size=24, color=BLACK)
```

```
    # Position the rectangle and number text
    if i == 0:
        rect.next_to(section_title, DOWN*2, aligned_edge=LEFT)
        rect.shift(LEFT*4)
    else:
        rect.next_to(outline_items[i-1][0], DOWN * 2, aligned_edge=LEFT)

    number.move_to(rect.get_center())

    # Create slide title text and align it to the right of the rectangle
    slide_text = Text(slide, font_size=24, color=BLACK)
    slide_text.next_to(rect, RIGHT, buff=0.5)

    # Add rectangle, number, and slide title to outline items for reference
    outline_items.append((rect, number, slide_text))

    # Animate the appearance of each rectangle and number followed by slide title
    self.play(Create(rect, shift=LEFT), FadeIn(number, shift=UP), run_time=0.5)
    self.play(Write(slide_text), run_time=0.8)
    self.add(slide_text)
    #self.play(Circumscribe(slide_text, color=DARK_BLUE))
    # Wait briefly before displaying the next slide
    self.wait(0.3)

# Wait for a few seconds to let the audience absorb the information
self.wait(2)
```

```
# File: metasurface_theory_section.py
from manim import *

class TheoreticalExplanationIntro(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Create the section title text
        section_title = Text("Theoretical Explanation", font_size=48, color=BLACK)
        section_title.to_edge(UP)

        # List of slides for the section
        slides = [
            "Background on Metasurfaces",
            "How to Calculate Elements Phases",
            "Formulas for Different Configurations of Meta-surface",
            "Examples for Different Configurations of Meta-surface"
        ]

        # Create section title
        self.play(Write(section_title, run_time=1.5))
        self.wait(1)

        # Display each slide title with animations
        outline_items = []
        for i, slide in enumerate(slides):
            # Create a rectangle and number text
            rect = Rectangle(width=0.5, height=0.5, color=BLACK)
```

```
    number = Text(f"{i + 1}", font_size=32, color=BLACK)

    # Position the rectangle and number text
    if i == 0:
        rect.next_to(section_title, DOWN*4, aligned_edge=LEFT)
        rect.shift(LEFT*2)
    else:
        rect.next_to(outline_items[i-1][0], DOWN * 2, aligned_edge=LEFT)

    number.move_to(rect.get_center())

    # Create slide title text and align it to the right of the rectangle
    slide_text = Text(slide, font_size=32, color=BLACK)
    slide_text.next_to(rect, RIGHT, buff=0.5)

    # Add rectangle, number, and slide title to outline items for reference
    outline_items.append((rect, number, slide_text))

    # Animate the appearance of each rectangle and number followed by slide title
    self.play(Create(rect, shift=LEFT), FadeIn(number, shift=UP), run_time=0.5)
    self.play(Write(slide_text), run_time=0.8)
    self.add(slide_text)
    #self.play(Circumscribe(slide_text, color=DARK_BLUE))
    # Wait briefly before displaying the next slide
    self.wait(0.3)

# Wait for a few seconds to let the audience absorb the information
self.wait(2)
```

```
# File: Background_on_Metasurfaces.py
from manim import *


class BackgroundonMetasurfaces(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE
        section_title = Text("Background on Metasurfaces", font_size=48, color=BLACK)
        image=ImageMobject(r"pictures\meta_surface.jpg").scale(0.8)
        # Create the text content
        paragraph = Paragraph(
            "Metasurfaces are artificially structured materials designed to control electromagnetic waves. ",
            "They consist of an array of elements (often called 'meta-atoms') that can independently manipulate ",
            "the amplitude, phase, and polarization of incident waves. The ability to steer and control waves ",
            "comes from introducing specific phase shifts at each element of the metasurface.",
            font_size=28,
            color=BLACK,
            line_spacing=2
        )


        section_title.to_edge(UP)
        paragraph.set_width(10)


        paragraph.next_to(section_title,DOWN*2,aligned_edge=RIGHT)
        image.next_to(paragraph, DOWN*4)
```

```
        # Add animations to display text and box
        self.play(Write(section_title), run_time=2)
        self.play(Write(paragraph), FadeIn (image), run_time=3)
        self.play(image.animate.shift(RIGHT*3),run_time=2)
        self.play(Circumscribe(image, color=BLACK))
        # Wait for a moment before ending the scene
        self.wait(3)



from manim import *


class How_to_Calculate_Elements_Phase(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE
        section_title = Text("How to Calculate Elements Phases", font_size=48, color=BLACK)
        section_title.to_edge(UP)


        # Multi-line Formula Breakdown
        formula_line1 = MathTex(
            r"\text{phase}(i_x, i_y) = k \left(",
            r"\underbrace{\sqrt{(x_{cor} - x_i)^2 + (y_{cor} - y_i)^2 + (z_{cor} - z_i)^2}}_{\text{Distance } R}",
            font_size=32,
            color=BLACK
        )


        formula_line2 = MathTex(
            r"- \underbrace{\sin(\theta_{dir}) \cdot \left( x_i \cos(\phi_{dir}) + y_i \sin(\phi_{dir})
\right)}_{\text{Elevation Adjustment}}",
```

```
        font_size=32,
        color=BLACK
    )

    formula_line3 = MathTex(
        r"- \underbrace{z_i \cos(\theta_{dir})}_{\text{Conformality Adjustment}} \right) +",
        font_size=32,
        color=BLACK
    )

    formula_line4 = MathTex(
        r"\underbrace{OAM_m \cdot \arctan\left(\frac{y_i}{x_i}\right)}_{\text{OAM Phase}}",
        font_size=32,
        color=BLACK
    )

    # Arrange the formula lines using VGroup for neat spacing
    formula = VGroup(formula_line1, formula_line2, formula_line3, formula_line4)
    formula.arrange(DOWN*1.2, aligned_edge=LEFT, buff=0.2).next_to(section_title, DOWN*2, buff=0.5, aligned_edge=LEFT)
    formula.shift(LEFT)

    # Distance Calculation (R)
    distance_title = Tex(r"\textbf{Distance Calculation (R):}", font_size=32, color=BLACK)
    distance_title.next_to(section_title, DOWN*4, aligned_edge=LEFT)

    distance_formula = MathTex(
        r"R = \sqrt{(x_{cor} - x_i)^2 + (y_{cor} - y_i)^2 + (z_{cor} - z_i)^2}",
        font_size=32,
```

```
        color=BLACK
    ).next_to(distance_title, DOWN, aligned_edge=LEFT, buff=0.4)

    distance_description = Tex(
        r"This term calculates the distance between the feed point $(x_{cor}, y_{cor}, z_{cor})$ and each element location $(x_i, y_i, z_i)$. ",
        font_size=32,
        color=BLACK
    ).next_to(distance_title, DOWN*3, aligned_edge=LEFT, buff=0.5)


    # Elevation Adjustment
    elevation_title = Tex(r"\textbf{Elevation Adjustment:}", font_size=32, color=BLACK)
    elevation_title.next_to(section_title, DOWN, aligned_edge=LEFT, buff=0.8)

    elevation_formula = MathTex(
        r"-\sin(\theta_{dir}) \cdot (x_i \cos(\phi_{dir}) + y_i \sin(\phi_{dir}))",
        font_size=32,
        color=BLACK
    ).next_to(elevation_title, DOWN, aligned_edge=LEFT, buff=0.4)

    elevation_description = Tex(
        r"This term accounts for the elevation angle $\theta_{dir}$.",
        font_size=32,
        color=BLACK
    ).next_to(elevation_formula, DOWN, aligned_edge=LEFT, buff=0.5)


    # Conformality Adjustment
```

```
conformality_title = Tex(r"\textbf{Conformality Adjustment:}", font_size=32, color=BLACK)
conformality_title.next_to(section_title, DOWN, aligned_edge=LEFT, buff=0.8)

conformality_formula = MathTex(
    r"-z_i \cos(\theta_{dir})",
    font_size=32,
    color=BLACK
).next_to(conformality_title, DOWN, aligned_edge=LEFT, buff=0.4)

conformality_description = Tex(
    r"This term adjusts for any element height $z_i$ and how it affects the steering of the beam. "
    r"If the metasurface is not perfectly flat, or if we intentionally introduce offsets in height, this term adjusts the phase accordingly.",
    font_size=32,
    color=BLACK
).next_to(conformality_formula, DOWN, aligned_edge=LEFT, buff=0.5)


# OAM Phase
oam_title = Tex(r"\textbf{OAM Phase:}", font_size=32, color=BLACK)
oam_title.next_to(section_title, DOWN, aligned_edge=LEFT, buff=0.8)

oam_formula = MathTex(
    r"+OAM_m \cdot \arctan\left(\frac{y_i}{x_i}\right)",
    font_size=32,
    color=BLACK
).next_to(oam_title, DOWN, aligned_edge=LEFT, buff=0.4)

oam_description = Tex(
```

```
        r"This final term introduces an OAM (Orbital Angular Momentum) mode, creating a spiral-like phase pattern. "
        r"When $OAM_m$ is non-zero, it causes the wavefront to carry a helical phase structure, leading to an OAM beam. ",
        font_size=32,
        color=BLACK
    ).next_to(oam_formula, DOWN, aligned_edge=LEFT, buff=0.5)



    # Animations
    self.play(Write(section_title), run_time=1.5)
    self.play(Write(formula), run_time=3)
    self.wait(3)

self.play(TransformMatchingShapes(formula,distance_formula),Write(distance_title),Write(distance_description))
    self.wait(3)

self.play(TransformMatchingShapes(distance_formula,elevation_formula),TransformMatchingShapes(distance_title,elevation_title),TransformMatchingShapes(distance_description,elevation_description))
    self.wait(3)

self.play(TransformMatchingShapes(elevation_formula,conformality_formula),TransformMatchingShapes(elevation_title,conformality_title),TransformMatchingShapes(elevation_description,conformality_description))
    self.wait(3)

self.play(FadeTransformPieces(conformality_formula,oam_formula),FadeTransformPieces(conformality_title,oam_title),FadeTransformPieces(conformality_description,oam_description))
    self.wait(3)



# File: configurations_applications.py
```

```
from manim import *

class ConfigurationsAndApplications(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Slide Title
        slide_title = Text("Formulas: Different Configurations of Meta-surface", font_size=40, color=BLACK)
        slide_title.to_edge(UP)

        # Subsection 1: Broadside Flat Metasurface
        subsection1_title = Tex(r"\subsection*{1. Broadside Flat Metasurface}", font_size=32, color=BLACK)
        subsection1_title.next_to(slide_title, DOWN*2, aligned_edge=LEFT)

        subsection1_text = Tex(
            r"In a broadside configuration the desired beam direction is directly perpendicular. "
            r"Thus, the direction angles $\theta_{dir}$ and $\phi_{dir}$ are zero.",
            font_size=32,
            color=BLACK
        ).next_to(subsection1_title, DOWN, aligned_edge=LEFT, buff=0.5)

        subsection1_bullet1 = Tex(r"Desired direction angles: $\theta_{dir} = 0$ and $\phi_{dir} = 0$.", font_size=32, color=BLACK).next_to(subsection1_text, DOWN, aligned_edge=LEFT, buff=0.5)

        subsection1_formula = MathTex(
            r"\text{phase}(i_x, i_y) = k \cdot \sqrt{(x_{cor} - x_i)^2 + (y_{cor} - y_i)^2 + (z_{cor} - z_i)^2}",
            font_size=32,
            color=BLACK
```

```
).next_to(subsection1_bullet1, DOWN, aligned_edge=LEFT, buff=0.5)


# Subsection 2: Offset Metasurface

subsection2_title = Tex(r"\subsection*{2. Conformal broadside Metasurface}", font_size=32, color=BLACK)

subsection2_title.next_to(slide_title, DOWN*2, aligned_edge=LEFT)


subsection2_text = Tex(

    r"A conformal metasurface introduces a non-zero height. Each element needs a slightly different phase shift to achieve constructive interference in the braodside direction.",

    font_size=32,

    color=BLACK

).next_to(subsection2_title, DOWN, aligned_edge=LEFT, buff=0.5)


subsection2_bullet1 = Tex(r"The conformality adjustment term becomes critical here, as it corrects for differences in height $ z_i $.", font_size=32, color=BLACK).next_to(subsection2_text, DOWN, aligned_edge=LEFT, buff=0.5)


subsection2_formula = MathTex(

    r"\text{phase}(i_x, i_y) = k \left(\sqrt{(x_{cor} - x_i)^2 + (y_{cor} - y_i)^2 + (z_{cor} - z_i)^2} - z_i \right)",

    font_size=32,

    color=BLACK

).next_to(subsection2_bullet1, DOWN, aligned_edge=LEFT, buff=0.5)


# Subsection 3: Steered Beam at Different Angles

subsection3_title = Tex(r"\subsection*{3. Steered Beam at Different Angles}", font_size=32, color=BLACK)

subsection3_title.next_to(slide_title, DOWN*2, aligned_edge=LEFT)
```

```
    subsection3_text = Tex(
        r"When we want to steer the beam at specific angles:",
        font_size=32,
        color=BLACK
    ).next_to(subsection3_title, DOWN, aligned_edge=LEFT, buff=0.5)

    subsection3_bullet1 = Tex(r"Desired direction angles are now $\theta_{dir} \neq 0$ and/or $\phi_{dir} \neq 0$.", font_size=32, color=BLACK).next_to(subsection3_text, DOWN, aligned_edge=LEFT, buff=0.5)

    # Two-line formula for Steered Beam at Different Angles
    formula_line1_3 = MathTex(
        r"\text{phase}(i_x, i_y) = k \left(\sqrt{(x_{cor} - x_i)^2 + (y_{cor} - y_i)^2 + (z_{cor} - z_i)^2} - ",
        font_size=32,
        color=BLACK
    )

    formula_line2_3 = MathTex(
        r"\sin(\theta_{dir})\cdot (x_i \cos(\phi_{dir}) + y_i \sin(\phi_{dir})) - z_i \cos(\theta_{dir}) \right)",
        font_size=32,
        color=BLACK
    )

    # Arrange the formula lines using VGroup for neat spacing
    subsection3_formula = VGroup(formula_line1_3, formula_line2_3)
    subsection3_formula.arrange(DOWN, aligned_edge=LEFT, buff=0.2).next_to(subsection3_bullet1, DOWN*1.5, aligned_edge=LEFT, buff=0.5)

    # Subsection 4: OAM Modes for Metasurface
```

```
    subsection4_title = Tex(r"\subsection*{4. OAM Modes for Metasurface}", font_size=32, color=BLACK)
    subsection4_title.next_to(slide_title, DOWN*2, aligned_edge=LEFT)

    subsection4_text = Tex(
        r"Finally, to create beams with OAM:",
        font_size=32,
        color=BLACK
    ).next_to(subsection4_title, DOWN, aligned_edge=LEFT, buff=0.5)

    subsection4_bullet1 = Tex(r"We introduce the OAM phase term based on the coordinates of each element $(x_i, y_i)$.", font_size=32, color=BLACK).next_to(subsection4_text, DOWN, aligned_edge=LEFT, buff=0.5)

    # Multi-line formula for OAM Modes
    formula_line1_4 = MathTex(
        r"\text{phase}(i_x, i_y) = k \left(\sqrt{(x_{cor} - x_i)^2 + (y_{cor} - y_i)^2 + (z_{cor} - z_i)^2}",
        font_size=32,
        color=BLACK
    )

    formula_line2_4 = MathTex(
        r"- \sin(\theta_{dir}) \cdot (x_i \cos(\phi_{dir}) + y_i \sin(\phi_{dir}))",
        font_size=32,
        color=BLACK
    )

    formula_line3_4 = MathTex(
        r"- z_i \cos(\theta_{dir}) \right) + OAM_m \cdot \arctan\left(\frac{y_i}{x_i}\right)",
        font_size=32,
```

```
        color=BLACK
    )

    # Arrange the formula lines using VGroup for neat spacing
    subsection4_formula = VGroup(formula_line1_4, formula_line2_4, formula_line3_4)
    subsection4_formula.arrange(DOWN, aligned_edge=LEFT, buff=0.2).next_to(subsection4_bullet1, DOWN, aligned_edge=LEFT, buff=0.5)

    # Animations to display all sections and subsections
    self.play(Write(slide_title), run_time=1.5)
    self.wait(0.5)

    # Subsection 1 Animations
    self.play(Write(subsection1_title), run_time=1.5)
    self.play(Write(subsection1_text), run_time=2)
    self.play(Write(subsection1_bullet1), run_time=1.5)
    self.play(Write(subsection1_formula), run_time=1.5)
    self.wait(3)

    # Subsection 2 Animations
    self.play(TransformMatchingShapes(subsection1_title,subsection2_title), run_time=1.5)
    self.play(TransformMatchingShapes(subsection1_text,subsection2_text), run_time=2)
    self.play(TransformMatchingShapes(subsection1_bullet1,subsection2_bullet1), TransformMatchingShapes(subsection1_formula,subsection2_formula), run_time=1.5)
    self.wait(3)

    # Subsection 3 Animations
    self.play(TransformMatchingShapes(subsection2_title,subsection3_title), run_time=1.5)
    self.play(TransformMatchingShapes(subsection2_text,subsection3_text), run_time=2)
```

```
        self.play(TransformMatchingShapes(subsection2_bullet1,subsection3_bullet1), run_time=1.5)
        self.play(TransformMatchingShapes(subsection2_formula,subsection3_formula), run_time=1.5)
        self.wait(3)

        # Subsection 4 Animations
        self.play(TransformMatchingShapes(subsection3_title,subsection4_title), run_time=1.5)
        self.play(TransformMatchingShapes(subsection3_text,subsection4_text), run_time=2)
        self.play(TransformMatchingShapes(subsection3_bullet1,subsection4_bullet1), run_time=1.5)
        self.play(TransformMatchingShapes(subsection3_formula,subsection4_formula), run_time=1.5)
        self.wait(3)



# File: configurations_applications.py
from manim import *

class ConfigurationsAndApplications2(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Title for the slide
        slide_title = Text("Examples: Different Configurations of Meta-surfaces", font_size=40, color=BLACK)
        slide_title.to_edge(UP)

        # Position of images and text
        image_position = ORIGIN
        text_position = DOWN * 3

        # Load all images
```

```
    flat_broadside = ImageMobject(r"pictures\FLAT_Broadside.png")
    concave_broadside = ImageMobject(r"pictures\Concave_Broadside.png")
    convex_broadside = ImageMobject(r"pictures\Convex_Broadside.png")
    steered_beam = ImageMobject(r"pictures\Convex_60deg.png")
    convex_oam3 = ImageMobject(r"pictures\Convex_broadside_OAM3.png")
    convex_oam10 = ImageMobject(r"pictures\Convex_broadside_OAM10.png")
    convex_oam20 = ImageMobject(r"pictures\Convex_broadside_OAM20.png")
    concave_oam3 = ImageMobject(r"pictures\Concave_broadside_OAM3.png")
    concave_oam10 = ImageMobject(r"pictures\Concave_broadside_OAM10.png")
    concave_oam20 = ImageMobject(r"pictures\Concave_broadside_OAM20.png")
    convex_oam_60deg = ImageMobject(r"pictures\Convex_60deg_OAM3.png")
    concave_60deg_oam = ImageMobject(r"pictures\Cocave_60deg_OAM3.png")

    # Position all images at the center
    for image in [flat_broadside,concave_broadside,convex_broadside,steered_beam,
convex_oam3,convex_oam10,
convex_oam20,concave_oam3,concave_oam10,concave_oam20,convex_oam_60deg,concave_60deg_o
am]:
        image.move_to(image_position)

    # Text explanations for each configuration
    flat_broadside_text = Tex(
        r"Broadside beam with flat metasurface .", font_size=32, color=BLACK
    ).move_to(text_position)



    concave_broadside_text = Tex(
        r"Broadside beam with a concave metasurface.", font_size=32, color=BLACK
    ).move_to(text_position)
```

```
    convex_broadside_text = Tex(
        r"Broadside beam with a convex metasurface.", font_size=32, color=BLACK
    ).move_to(text_position)


    steered_beam_text = Tex(
        r"Convex metasurface directing the beam at 60 degree.", font_size=32, color=BLACK
    ).move_to(text_position)

    convex_broadside_oam_text3 = Tex(
        r"Broadside beam with Orbital Angular Momentum (mode 3) in a convex metasurface.",
font_size=32, color=BLACK
    ).move_to(text_position)

    convex_broadside_oam_text10 = Tex(
        r"Broadside beam with Orbital Angular Momentum (mode 10) in a convex metasurface.",
font_size=32, color=BLACK
    ).move_to(text_position)

    convex_broadside_oam_text20 = Tex(
        r"Broadside beam with Orbital Angular Momentum (mode 20) in a convex metasurface.",
font_size=32, color=BLACK
    ).move_to(text_position)

    concave_oam_text3 = Tex(
        r"Broadside beam with Orbital Angular Momentum (mode 3) in a concave metasurface.",
font_size=32, color=BLACK
    ).move_to(text_position)

    concave_oam_text10 = Tex(
```

```
        r"Broadside beam with Orbital Angular Momentum (mode 10) in a concave metasurface.", font_size=32, color=BLACK
    ).move_to(text_position)


    concave_oam_text20 = Tex(
        r"Broadside beam with Orbital Angular Momentum (mode 20) in a concave metasurface.", font_size=32, color=BLACK
    ).move_to(text_position)


    convex_oam_60deg_text = Tex(
        r"Beam directed at 60 degree with Orbital Angular Momentum (mode 3) with a convex metasurface.", font_size=32, color=BLACK
    ).move_to(text_position)

    concave_60deg_oam_text = Tex(
        r"Beam directed at 60 degree with Orbital Angular Momentum (mode 3) with a concave metasurface.", font_size=32, color=BLACK
    ).move_to(text_position)

    # Display the title
    self.play(Write(slide_title), run_time=1.5)
    self.wait(0.5)

    # Display Flat Broadside case
    self.play(FadeIn(flat_broadside), Write(flat_broadside_text), run_time=2)
    self.wait(3)

    # Transition to Steered Beam case
```

```
        self.play(Transform(flat_broadside, convex_broadside), TransformMatchingShapes(flat_broadside_text, convex_broadside_text), run_time=2)

        self.wait(3)


            # Transition to Steered Beam case

        self.play(Transform(convex_broadside, concave_broadside), TransformMatchingShapes(convex_broadside_text, concave_broadside_text), run_time=2)

        self.wait(3)




        self.play(Transform(concave_broadside, steered_beam), TransformMatchingShapes(concave_broadside_text, steered_beam_text), run_time=2)

        self.wait(3)


        self.play(Transform(steered_beam, convex_oam3), TransformMatchingShapes(steered_beam_text, convex_broadside_oam_text3), run_time=2)

        self.wait(3)




        self.play(Transform(convex_oam3, convex_oam10), TransformMatchingShapes(convex_broadside_oam_text3, convex_broadside_oam_text10), run_time=2)

        self.wait(3)




        self.play(Transform(convex_oam10, convex_oam20), TransformMatchingShapes(convex_broadside_oam_text10, convex_broadside_oam_text20), run_time=2)

        self.wait(3)




        self.play(Transform(convex_oam20, concave_oam3), TransformMatchingShapes(convex_broadside_oam_text20, concave_oam_text3), run_time=2)
```

```
        self.wait(3)



        self.play(Transform(concave_oam3, concave_oam10),
TransformMatchingShapes(concave_oam_text3, concave_oam_text10), run_time=2)
        self.wait(3)


        self.play(Transform(concave_oam10, concave_oam20),
TransformMatchingShapes(concave_oam_text10, concave_oam_text20), run_time=2)
        self.wait(3)


        self.play(Transform(concave_oam20, convex_oam_60deg),
TransformMatchingShapes(concave_oam_text20, convex_oam_60deg_text), run_time=2)
        self.wait(3)


        self.play(Transform(convex_oam_60deg, concave_60deg_oam),
TransformMatchingShapes(convex_oam_60deg_text, concave_60deg_oam_text), run_time=2)
        self.wait(3)



        # File: historical_timeline.py
from manim import *


class HistoricalTimeline(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE


        # Main Title for the slide
        main_title = Tex(r"Historical Timeline of Metasurfaces", font_size=48, color=BLACK)
```

```
        main_title.to_edge(UP)

        # Position of timeline titles and text
        timeline_title_position = UP * 1.5
        text_position = DOWN * 1.5

        # Create timeline titles and texts
        timeline_1960_1970 = Tex(
            r"1960s - 1970s: Development of Reflectarrays", font_size=36, color=BLACK
        ).move_to(timeline_title_position)

        text_1960_1970 = Tex(
            r"Reflectarrays were originally conceived as an alternative to parabolic reflectors. "
            r"They consisted of a flat surface with a grid of resonant elements designed to introduce specific phase shifts, "
            r"effectively shaping and directing the reflected beam.",
            font_size=28,
            color=BLACK
        ).scale(0.8).move_to(text_position).set_width(12)

        timeline_1980_1990 = Tex(
            r"1980s - 1990s: Refinement of Reflectarrays", font_size=36, color=BLACK
        ).move_to(timeline_title_position)

        text_1980_1990 = Tex(
            r"This period saw improvements in reflectarray designs, including more sophisticated elements and better analytical models. "
            r"Reflectarrays became increasingly popular in satellite communications.",
            font_size=28,
```

```
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(12)

    timeline_2000 = Tex(
        r"2000s: Emergence of Electromagnetic Metamaterials", font_size=36, color=BLACK
    ).move_to(timeline_title_position)

    text_2000 = Tex(
        r"The concept of ``metamaterials'' gained traction, focusing on materials with properties not found in nature, such as negative refractive index. "
        r"Metasurfaces were introduced as a 2D analog of 3D metamaterials, allowing precise manipulation of electromagnetic waves using subwavelength-sized elements.",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(12)

    timeline_2010_2015 = Tex(
        r"2010 - 2015: Growth of Tunable and Active Metasurfaces", font_size=36, color=BLACK
    ).move_to(timeline_title_position)

    text_2010_2015 = Tex(
        r"Advances in materials science led to the development of tunable metasurfaces, which could dynamically control wavefronts through adjustable properties. "
        r"The idea of ``coding metasurfaces'' emerged, using digital codes to represent different phase responses, laying the foundation for programmable surfaces.",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(12)

    timeline_2016_2018 = Tex(
```

```
        r"2016 - 2018: Introduction of Intelligent Reflecting Surfaces (IRS)", font_size=36, color=BLACK
    ).move_to(timeline_title_position)

    text_2016_2018 = Tex(
        r"Intelligent Reflecting Surfaces (IRS) combined traditional reflectarrays with smart control electronics to dynamically shape and steer beams in real-time. "
        r"IRS technology adapted to changing environments and began being seen as a promising solution for optimizing wireless communication networks.",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(12)

    timeline_2019_present = Tex(
        r"2019 - Present: Advances in IRS Technology", font_size=36, color=BLACK
    ).move_to(timeline_title_position)

    text_2019_present = Tex(
        r"Researchers focused on enhancing IRS with better reconfigurability, energy efficiency, and integration with machine learning algorithms. "
        r"New materials such as tunable liquid crystals, graphene-based structures, and electronically controlled meta-atoms were explored to improve IRS performance.",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(12)

    # Display the main title
    self.play(Write(main_title), run_time=1.5)
    self.wait(0.5)

    # Display 1960s - 1970s Timeline
```

```
self.play(FadeIn(timeline_1960_1970), Write(text_1960_1970), run_time=2)
self.wait(2)

# Transition to 1980s - 1990s Timeline
self.play(
    TransformMatchingShapes(timeline_1960_1970, timeline_1980_1990),
    TransformMatchingShapes(text_1960_1970, text_1980_1990),
    run_time=2
)
self.wait(2)

# Transition to 2000s Timeline
self.play(
    TransformMatchingShapes(timeline_1980_1990, timeline_2000),
    TransformMatchingShapes(text_1980_1990, text_2000),
    run_time=2
)
self.wait(2)

# Transition to 2010 - 2015 Timeline
self.play(
    TransformMatchingShapes(timeline_2000, timeline_2010_2015),
    TransformMatchingShapes(text_2000, text_2010_2015),
    run_time=2
)
self.wait(2)

# Transition to 2016 - 2018 Timeline
self.play(
```

```
            TransformMatchingShapes(timeline_2010_2015, timeline_2016_2018),
            TransformMatchingShapes(text_2010_2015, text_2016_2018),
            run_time=2
        )
        self.wait(2)

        # Transition to 2019 - Present Timeline
        self.play(
            TransformMatchingShapes(timeline_2016_2018, timeline_2019_present),
            TransformMatchingShapes(text_2016_2018, text_2019_present),
            run_time=2
        )
        self.wait(2)


# File: applications_of_metasurfaces.py
from manim import *

# File: applications_of_metasurfaces_irs.py
from manim import *

class ApplicationsOfMetasurfacesAndIRS(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Main Title for the slide
        main_title = Tex(r"Applications of Metasurfaces and IRS", font_size=48, color=BLACK)
        main_title.to_edge(UP)
```

```
# Position of application titles and text
application_title_position = UP * 1.5
text_position = DOWN * 1.5

# Create application titles and texts
app_wireless_communication = Tex(
    r"Enhancing Wireless Communication Capacity", font_size=36, color=BLACK
).move_to(application_title_position)

text_wireless_communication = Tex(
    r"Beamforming and Beam Steering \\ MIMO Systems \\ Coverage Extension",
    font_size=28,
    color=BLACK
).scale(0.8).move_to(text_position).set_width(10)

app_spectral_efficiency = Tex(
    r"Spectral and Energy Efficiency Improvements", font_size=36, color=BLACK
).move_to(application_title_position)

text_spectral_efficiency = Tex(
    r"Energy-efficient Communication \\ Interference Management",
    font_size=28,
    color=BLACK
).scale(0.8).move_to(text_position).set_width(10)

app_security_privacy = Tex(
    r"Security and Privacy in Communications", font_size=36, color=BLACK
).move_to(application_title_position)
```

```
    text_security_privacy = Tex(
        r"Physical Layer Security \\ Directional Control for Secure Transmissions",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(10)

    app_sensing_imaging = Tex(
        r"Sensing and Imaging Applications", font_size=36, color=BLACK
    ).move_to(application_title_position)

    text_sensing_imaging = Tex(
        r"Smart Environment Sensing \\ Metasurface-based Holography \\ Imaging through Obstructions",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(10)

    app_reconfigurable_systems = Tex(
        r"Reconfigurable Optical and Microwave Systems", font_size=36, color=BLACK
    ).move_to(application_title_position)

    text_reconfigurable_systems = Tex(
        r"Reconfigurable Optics \\ Reconfigurable Antenna Design",
        font_size=28,
        color=BLACK
    ).scale(0.8).move_to(text_position).set_width(10)

    app_power_transfer = Tex(
```

```
    r"Wireless Power Transfer and Energy Harvesting", font_size=36, color=BLACK
).move_to(application_title_position)

text_power_transfer = Tex(
    r"Wireless Power Transfer (WPT) \\ Energy Harvesting",
    font_size=28,
    color=BLACK
).scale(0.8).move_to(text_position).set_width(10)

app_5g_beyond = Tex(
    r"5G and Beyond: Enabling Intelligent Network Architectures", font_size=36, color=BLACK
).move_to(application_title_position)

text_5g_beyond = Tex(
    r"5G and 6G Wireless Networks \\ Dynamic Network Optimization",
    font_size=28,
    color=BLACK
).scale(0.8).move_to(text_position).set_width(10)

# Display the main title
self.play(Write(main_title), run_time=1.5)
self.wait(0.5)

# Display Enhancing Wireless Communication Capacity application
self.play(FadeIn(app_wireless_communication), Write(text_wireless_communication), run_time=2)
self.wait(2)

# Transition to Spectral and Energy Efficiency Improvements application
self.play(
```

```
    TransformMatchingShapes(app_wireless_communication, app_spectral_efficiency),
    TransformMatchingShapes(text_wireless_communication, text_spectral_efficiency),
    run_time=2
)
self.wait(2)

# Transition to Security and Privacy in Communications application
self.play(
    TransformMatchingShapes(app_spectral_efficiency, app_security_privacy),
    TransformMatchingShapes(text_spectral_efficiency, text_security_privacy),
    run_time=2
)
self.wait(2)

# Transition to Sensing and Imaging Applications
self.play(
    TransformMatchingShapes(app_security_privacy, app_sensing_imaging),
    TransformMatchingShapes(text_security_privacy, text_sensing_imaging),
    run_time=2
)
self.wait(2)

# Transition to Reconfigurable Optical and Microwave Systems
self.play(
    TransformMatchingShapes(app_sensing_imaging, app_reconfigurable_systems),
    TransformMatchingShapes(text_sensing_imaging, text_reconfigurable_systems),
    run_time=2
)
self.wait(2)
```

```
        # Transition to Wireless Power Transfer and Energy Harvesting
        self.play(
            TransformMatchingShapes(app_reconfigurable_systems, app_power_transfer),
            TransformMatchingShapes(text_reconfigurable_systems, text_power_transfer),
            run_time=2
        )
        self.wait(2)

        # Transition to 5G and Beyond: Enabling Intelligent Network Architectures
        self.play(
            TransformMatchingShapes(app_power_transfer, app_5g_beyond),
            TransformMatchingShapes(text_power_transfer, text_5g_beyond),
            run_time=2
        )
        self.wait(2)




# File: metasurface_theory_section.py
from manim import *

class PythonCodeAutomatedHFSS1(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Create the section title text
```

```
section_title = Text(r"Python Code: Automated Structures in HFSS", font_size=48, color=BLACK)
section_title.to_edge(UP)

# List of slides for the section
slides = [
    "Jason_data_creator.py Code Explanation",
    "generate_hfss_script.py Code Explanation",
    "Plotter.py Code Explnation"
]

# Create section title
self.play(Write(section_title, run_time=1.5))
self.wait(1)

# Display each slide title with animations
outline_items = []
for i, slide in enumerate(slides):
    # Create a rectangle and number text
    rect = Rectangle(width=0.5, height=0.5, color=BLACK)
    number = Text(f"{i + 1}", font_size=32, color=BLACK)

    # Position the rectangle and number text
    if i == 0:
        rect.next_to(section_title, DOWN*4, aligned_edge=LEFT)
        rect.shift(RIGHT*2)
    else:
        rect.next_to(outline_items[i-1][0], DOWN * 2, aligned_edge=LEFT)

    number.move_to(rect.get_center())
```

```
            # Create slide title text and align it to the right of the rectangle
            slide_text = Text(slide, font_size=32, color=BLACK)
            slide_text.next_to(rect, RIGHT, buff=0.5)

            # Add rectangle, number, and slide title to outline items for reference
            outline_items.append((rect, number, slide_text))

            # Animate the appearance of each rectangle and number followed by slide title
            self.play(Create(rect, shift=LEFT), FadeIn(number, shift=UP), run_time=0.5)
            self.play(Write(slide_text), run_time=0.8)
            self.add(slide_text)
            #self.play(Circumscribe(slide_text, color=DARK_BLUE))
            # Wait briefly before displaying the next slide
            self.wait(0.3)

        # Wait for a few seconds to let the audience absorb the information
        self.wait(2)



# File: python_code_automated_hfss.py
from manim import *

class PythonCodeAutomatedHFSS2(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE
```

```
        # Main Title for the slide
        main_title = Text(r"Jason_data_creator.py Code Explanation", font_size=48, color=BLACK)
        main_title.to_edge(UP)

        # Position for text and code
        explanation_position = UP * 2
        code_position = UP* 0.5

        # Display the main title
        self.play(Write(main_title), run_time=1.5)
        self.wait(0.5)

        # Section: Explanation of Required Packages
        section_title1 = Tex(r"Packages Required:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation1 = Tex(
            r"numpy, pandas, scipy, matplotlib", font_size=28, color=BLACK
        ).next_to(section_title1, DOWN)

        code1 = Code(
            code="pip install numpy pandas scipy matplotlib",
            tab_width=4,
            language="bash",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).move_to(code_position)

        self.play(Write(section_title1), run_time=1)
```

```
        self.play(Write(explanation1), run_time=1)
        self.play(Create(code1), run_time=1.5)
        self.wait(2)



            # Subtitle for the section
        section_title2 = Tex(r"1 - Importing Packages:", font_size=36,
color=BLACK).move_to(explanation_position)

        # Explanation for each package
        explanation2 = Tex(
            r"\textbf{numpy (np)}: Handles numerical operations and array manipulations.\\"
            r"\textbf{pandas (pd)}: Used for handling tabular data, reading and saving CSV/JSON files.\\"
            r"\textbf{scipy.constants.speed\_of\_light}: Provides the speed of light constant, which is required
for calculating wavelength.",
            font_size=28,
            color=BLACK
        ).next_to(section_title2, DOWN)

            # Code Block for Importing Packages
        code2 = Code(
            code="""import numpy as np
import pandas as pd
from scipy.constants import speed_of_light""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
```

```
        ).next_to(explanation2, DOWN)


        self.play(TransformMatchingShapes(section_title1,section_title2), run_time=1)
        self.play(TransformMatchingShapes(explanation1,explanation2),
TransformMatchingShapes(code1,code2), run_time=1)
        self.wait(2)



            # Subtitle for the second section
        section_title3 = Tex(r"2 - Defining load\_unit\_cell\_data Function:", font_size=36,
color=BLACK).move_to(explanation_position)



        # Explanation for the second section
        explanation3 = Tex(
            r"\textbf{Purpose}: Loads reflection phase vs. size data from a CSV file into two arrays: phases and
sizes.\\"
            r"\textbf{pd.read\_csv}: Reads the CSV file located at file\_path.\\"
            r"\textbf{.to\_numpy()}: Converts the columns to numpy arrays for easier manipulation.",
            font_size=28,
            color=BLACK
        ).next_to(section_title3, DOWN)


        # Code Block for load_unit_cell_data Function
        code3 = Code(
            code="""def load_unit_cell_data(file_path):
df = pd.read_csv(file_path)
phases = df['Phase'].to_numpy()
sizes = df['Size'].to_numpy()
return phases, sizes""",
```

```
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation3, DOWN)

        # Transition to the second section
        self.play(TransformMatchingShapes(section_title2, section_title3), run_time=1)
        self.play(TransformMatchingShapes(explanation2, explanation3), TransformMatchingShapes(code2, code3), run_time=1)
        self.wait(2)

        # Subtitle for the third section
        section_title4 = Tex(r"3 - Defining calculate\_patch\_size Function:", font_size=36, color=BLACK).move_to(explanation_position)

        # Explanation for the third section
        explanation4 = Tex(
            r"\textbf{Purpose}: Determines the appropriate size of the patch based on the calculated target phase.\\"
            r"\textbf{Process}: Finds the index (idx) where the difference between the calculated target\_phase and existing phases is smallest. Returns the corresponding size from the sizes array.",
            font_size=28,
            color=BLACK
        ).next_to(section_title4, DOWN)

        # Code Block for calculate_patch_size Function
```

```
        code4 = Code(
            code="""def calculate_patch_size(target_phase, phases, sizes):
idx = (np.abs(phases - target_phase)).argmin()
return sizes[idx]""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation4, DOWN)

        # Transition to the third section
        self.play(TransformMatchingShapes(section_title3, section_title4), run_time=1)
        self.play(TransformMatchingShapes(explanation3, explanation4), TransformMatchingShapes(code3, code4), run_time=1)
        self.wait(2)



        # Subtitle for the fourth section
        section_title5 = Tex(r"4 - Defining pre\_calculate\_data Function:", font_size=36, color=BLACK).move_to(explanation_position)

        # Explanation for the fourth section
        explanation5 = Tex(
            r"\textbf{Purpose}: Main function to calculate phase and patch data, and save them in files.\\"
            r"\textbf{Parameters}: output\_file - The name of the JSON file where patch data will be saved.\\"
            r"phase\_data\_file - The name of the CSV file where phase data will be saved.",
            font_size=28,
```

```
            color=BLACK
        ).next_to(section_title5, DOWN)

        # Code Block for pre_calculate_data Function
        code5 = Code(
            code="""def pre_calculate_data(output_file="patch_data.json",
phase_data_file="phase_data.csv"):""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation5, DOWN)

        # Transition to the fourth section
        self.play(TransformMatchingShapes(section_title4, section_title5), run_time=1)
        self.play(TransformMatchingShapes(explanation4, explanation5), TransformMatchingShapes(code4,
code5), run_time=1)
        self.wait(2)

        # Subtitle for the fifth section with corrected LaTeX escaping
        section_title6 = Tex(r"5 - Load Reflection Data:", font_size=36,
color=BLACK).move_to(explanation_position)

        # Explanation for the fifth section with proper LaTeX escaping
        explanation6 = Tex(
            r"\textbf{Purpose}: Load the reflection phase vs. size data from the CSV file
``phase\_vs\_size.csv''.",
            font_size=28,
            color=BLACK
```

```
    ).next_to(section_title6, DOWN)

    # Code Block for loading reflection data
    code6 = Code(
        code="""phases, sizes = load_unit_cell_data("phase_vs_size.csv")""",
        tab_width=4,
        language="python",
        font="Monospace",
        background="window",
        insert_line_no=False
    ).next_to(explanation6, DOWN)

    # Transition to the fifth section
    self.play(TransformMatchingShapes(section_title5, section_title6), run_time=1)
    self.play(TransformMatchingShapes(explanation5, explanation6), TransformMatchingShapes(code5, code6), run_time=1)
    self.wait(2)


# Subtitle for the sixth section
    section_title7 = Tex(r"6 - Defining Parameters:", font_size=36, color=BLACK).move_to(explanation_position)

    # Explanation for the sixth section with corrected LaTeX formatting
    explanation7 = Tex(
        r"\textbf{Frequency (freq)}: Set to 10 GHz as an example.\\"
        r"\textbf{Wavelength Calculation}: Converts the speed of light (in m/s) to mm/s and divides by frequency to obtain the wavelength.\\"
        r"\textbf{Wave Number (k)}: Calculated as 2 $\cdot$ $\pi$ / wavelength.",
        font_size=28,
```

```
            color=BLACK
        ).next_to(section_title7, DOWN*0.5)

        # Code Block for Defining Parameters
        code7 = Code(
            code="""freq = 10e9  # Frequency in Hz (example value)
wavelength = (speed_of_light * 1000) / freq
k = 2 * np.pi / wavelength""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation7, DOWN)

        # Transition to the sixth section
        self.play(TransformMatchingShapes(section_title6, section_title7), run_time=1)
        self.play(TransformMatchingShapes(explanation6, explanation7), TransformMatchingShapes(code6, code7), run_time=1)
        self.wait(2)

        # Subtitle for the Example Values section
        section_title8 = Tex(r"7 - Design Values:", font_size=36, color=BLACK).move_to(explanation_position)
        section_title8.shift(UP*0.5)
        # Explanation for the Example Values section with proper LaTeX escaping
        explanation8 = Tex(
            r"\textbf{Direction Angles (the\_dir, phi\_dir)}: Theta and Phi directions in degrees.\\"
            r"\textbf{Feed Coordinates (x\_cor, y\_cor, z\_cor)}: Position of the feed antenna.\\"
```

```
            r"\textbf{Diameter (rad)}: The overall diameter of the array in mm.\\"
            r"\textbf{Cell Spacing (uedim)}: The distance between unit cells in the array.\\"
            r"\textbf{OAM Mode (OAM\_m)}: Orbital Angular Momentum mode.\\"
            r"\textbf{F/D Ratio (FOD)}: Ratio of focal length to diameter.\\"
            r"\textbf{Focal Length (f)}: Calculated as FOD $\times$ rad.",
            font_size=28,
            color=BLACK
        ).next_to(section_title8, DOWN*0.5)

        # Code Block for Example Values
        code8 = Code(
            code="""the_dir, phi_dir, pha_zer = 0
x_cor, y_cor, z_cor = 0, 0, 250
rad = 500
uedim = 4.6
OAM_m = 3
FOD = 0.4
f = FOD * rad  """,
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).to_edge(DOWN)

        # Transition to the Example Values section
        self.play(TransformMatchingShapes(section_title7, section_title8), run_time=1)
        self.play(TransformMatchingShapes(explanation7, explanation8), TransformMatchingShapes(code7, code8), run_time=1)
```

```
        self.wait(2)



            # Subtitle for the Height Parameter section
        section_title9 = Tex(r"8 - Calculate the Height Parameter:", font_size=36, color=BLACK).move_to(explanation_position)

        # Explanation for the Height Parameter section
        explanation9 = Tex(
            r"\textbf{Calculates the height parameter} using the formula $(\text{rad}^2) / (16 \cdot \text{f})$.",
            font_size=28,
            color=BLACK
        ).next_to(section_title9, DOWN)

        # Code Block for Calculating the Height Parameter
        code9 = Code(
            code="""h = (rad ** 2) / (16 * f)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation9, DOWN)

        # Transition to the Height Parameter section
        self.play(TransformMatchingShapes(section_title8, section_title9), run_time=1)
        self.play(TransformMatchingShapes(explanation8, explanation9), TransformMatchingShapes(code8, code9), run_time=1)
        self.wait(2)
```

```
        # Subtitle for the Set Conformal Mode section
        section_title10 = Tex(r"9 - Set Conformal Mode:", font_size=36, color=BLACK).move_to(explanation_position)

        # Explanation for the Set Conformal Mode section
        explanation10 = Tex(
            r"\textbf{Determines whether the surface is convex or concave} based on the value of \textit{sign\_z}.",
            font_size=28,
            color=BLACK
        ).next_to(section_title10, DOWN)

        # Code Block for Setting Conformal Mode
        code10 = Code(
            code="""sign_z = -1  # 1 for Convex, -1 for Concave (example value)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation10, DOWN)

        # Transition to the Set Conformal Mode section
        self.play(TransformMatchingShapes(section_title9, section_title10), run_time=1)
        self.play(TransformMatchingShapes(explanation9, explanation10), TransformMatchingShapes(code9, code10), run_time=1)
        self.wait(2)
```

```
        # Subtitle for the Generate Coordinate Grid section (now with the correct section number)
        section_title11 = Tex(r"10 - Generate Coordinate Grid:", font_size=36, color=BLACK).move_to(explanation_position)

        # Explanation for the Generate Coordinate Grid section
        explanation11 = Tex(
            r"\textbf{Creates a grid of points} within the given radius using \textit{np.meshgrid}.\\"
            r"\textbf{xi\_range and yi\_range} are the coordinate ranges.",
            font_size=28,
            color=BLACK
        ).next_to(section_title11, DOWN)

        # Code Block for Generating Coordinate Grid
        code11 = Code(
            code="""xi_range = np.arange(-rad / 2 + rad % uedim, rad / 2 + uedim, uedim)
yi_range = np.arange(-rad / 2 + rad % uedim, rad / 2 + uedim, uedim)
XI, YI = np.meshgrid(xi_range, yi_range)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation11, DOWN).scale(0.8)

        # Transition to the Generate Coordinate Grid section
        self.play(TransformMatchingShapes(section_title10, section_title11), run_time=1)
        self.play(TransformMatchingShapes(explanation10, explanation11), TransformMatchingShapes(code10, code11), run_time=1)
        self.wait(2)
```

```
        # Subtitle for the Calculate Angular Phase section
        section_title12 = Tex(r"11 - Calculate Angular Phase:", font_size=36, color=BLACK).move_to(explanation_position)

        # Explanation for the Calculate Angular Phase section
        explanation12 = Tex(
            r"\textbf{Computes the angular phase shift} using the arctangent of \textit{YI} and \textit{XI}.",
            font_size=28,
            color=BLACK
        ).next_to(section_title12, DOWN)

        # Code Block for Calculating Angular Phase
        code12 = Code(
            code="""angular_phase = OAM_m * np.arctan2(YI, XI)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation12, DOWN)

        # Transition to the Calculate Angular Phase section
        self.play(TransformMatchingShapes(section_title11, section_title12), run_time=1)
        self.play(TransformMatchingShapes(explanation11, explanation12), TransformMatchingShapes(code11, code12), run_time=1)
        self.wait(2)
```

```
  # Subtitle for the Initialize Phase Data section
      section_title13 = Tex(r"12 - Initialize Phase Data:", font_size=36, color=BLACK).move_to(explanation_position)

      # Explanation for the Initialize Phase Data section
      explanation13 = Tex(
          r"\textbf{Creates an empty 2D array} to store the calculated phase values.",
          font_size=28,
          color=BLACK
      ).next_to(section_title13, DOWN)

      # Code Block for Initializing Phase Data
      code13 = Code(
          code="""phase_deg_2d = np.full(XI.shape, np.nan)""",
          tab_width=4,
          language="python",
          font="Monospace",
          background="window",
          insert_line_no=False
      ).next_to(explanation13, DOWN)

      # Transition to the Initialize Phase Data section
      self.play(TransformMatchingShapes(section_title12, section_title13), run_time=1)
      self.play(TransformMatchingShapes(explanation12, explanation13), TransformMatchingShapes(code12, code13), run_time=1)
      self.wait(2)
```

```
        # Part 13: Loop Through the Grid
    section_title14 = Tex(r"13 - Loop Through the Grid:", font_size=36, color=BLACK).move_to(explanation_position)
    explanation14 = Tex(
        r"\textbf{Iterates over all grid points} and checks if they fall within the radius.",
        font_size=28,
        color=BLACK
    ).next_to(section_title14, DOWN)
    code14 = Code(
        code="""for ix, xi in enumerate(xi_range):
for iy, yi in enumerate(yi_range):
if np.sqrt(xi ** 2 + yi ** 2) < rad / 2:
..."""",
        tab_width=4,
        language="python",
        font="Monospace",
        background="window",
        insert_line_no=False
    ).next_to(explanation14, DOWN)

    self.play(TransformMatchingShapes(section_title13, section_title14), run_time=1)
    self.play(TransformMatchingShapes(explanation13, explanation14), TransformMatchingShapes(code13, code14), run_time=1)
    self.wait(2)

    # Part 14: Calculate the Height and Distance
    section_title15 = Tex(r"14 - Calculate the Height and Distance:", font_size=36, color=BLACK).move_to(explanation_position)
    explanation15 = Tex(
```

```
        r"\textbf{Computes the height } \textit{zi} \textbf{and the distance } \textit{R} \textbf{from the feed coordinates.}",
        font_size=28,
        color=BLACK
    ).next_to(section_title15, DOWN)
    code15 = Code(
        code="""zi = sign_z *
(h - (xi ** 2 + yi ** 2) / (4 * f))
R = np.sqrt((x_cor - xi) ** 2 +
(y_cor - yi) ** 2 + (z_cor - zi) ** 2)""",
        tab_width=4,
        language="python",
        font="Monospace",
        background="window",
        insert_line_no=False
    ).next_to(explanation15, DOWN)

    self.play(TransformMatchingShapes(section_title14, section_title15), run_time=1)
    self.play(TransformMatchingShapes(explanation14, explanation15), TransformMatchingShapes(code14, code15), run_time=1)
    self.wait(2)

    # Part 15: Calculate Phase Shift
    section_title16 = Tex(r"15 - Calculate Phase Shift:", font_size=36, color=BLACK).move_to(explanation_position)
    explanation16 = Tex(
        r"\textbf{Calculates the phase shift} for each grid point.",
        font_size=28,
        color=BLACK
    ).next_to(section_title16, DOWN)
```

```
        code16 = Code(
            code="""phase = k * (R - zi * np.cos(np.deg2rad(the_dir))
- np.sin(np.deg2rad(the_dir)) *
(xi * np.cos(np.deg2rad(phi_dir))
+ yi * np.sin(np.deg2rad(phi_dir))))""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation16, DOWN)

        self.play(TransformMatchingShapes(section_title15, section_title16), run_time=1)
        self.play(TransformMatchingShapes(explanation15, explanation16),
TransformMatchingShapes(code15, code16), run_time=1)
        self.wait(2)

        # Part 16: Calculate Modified Phase in Degrees
        section_title17 = Tex(r"16 - Calculate Modified Phase in Degrees:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation17 = Tex(
            r"\textbf{Converts the calculated phase into degrees} and adds a OAM phase.",
            font_size=28,
            color=BLACK
        ).next_to(section_title17, DOWN)
        code17 = Code(
            code="""m_phase = np.mod(angular_phase[iy, ix] + phase, 2 * np.pi)
m_phase_deg = m_phase * 180 / np.pi + pha_zer""",
            tab_width=4,
```

```
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation17, DOWN)

        self.play(TransformMatchingShapes(section_title16, section_title17), run_time=1)
        self.play(TransformMatchingShapes(explanation16, explanation17),
TransformMatchingShapes(code16, code17), run_time=1)
        self.wait(2)

        # Part 17: Store Phase in 2D Array
        section_title18 = Tex(r"17 - Store Phase in 2D Array:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation18 = Tex(
            r"\textbf{Stores the calculated phase} in the 2D array.",
            font_size=28,
            color=BLACK
        ).next_to(section_title18, DOWN)
        code18 = Code(
            code="""phase_deg_2d[iy, ix] = m_phase_deg""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation18, DOWN)

        self.play(TransformMatchingShapes(section_title17, section_title18), run_time=1)
```

```
        self.play(TransformMatchingShapes(explanation17, explanation18), TransformMatchingShapes(code17, code18), run_time=1)
        self.wait(2)

        # Part 18: Determine Patch Size
        section_title19 = Tex(r"18 - Determine Patch Size:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation19 = Tex(
            r"\textbf{Determines the size of the patch} based on the calculated phase.",
            font_size=28,
            color=BLACK
        ).next_to(section_title19, DOWN)
        code19 = Code(
            code="""patch_size = calculate_patch_size(m_phase_deg, phases, sizes)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation19, DOWN)

        self.play(TransformMatchingShapes(section_title18, section_title19), run_time=1)
        self.play(TransformMatchingShapes(explanation18, explanation19), TransformMatchingShapes(code18, code19), run_time=1)
        self.wait(2)

        # Part 19: Append Patch Data
        section_title20 = Tex(r"19 - Append Patch Data:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation20 = Tex(
```

```
            r"\textbf{Appends the calculated data} to the patch\_data list.",
            font_size=28,
            color=BLACK
        ).next_to(section_title20, DOWN)
        code20 = Code(
            code="""patch_data.append({
"xi": xi,
"yi": yi,
"zi": zi,
"patch_size": patch_size,
"m_phase_deg": m_phase_deg
})""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation20, DOWN)

        self.play(TransformMatchingShapes(section_title19, section_title20), run_time=1)
        self.play(TransformMatchingShapes(explanation19, explanation20),
TransformMatchingShapes(code19, code20), run_time=1)
        self.wait(2)

        # Part 20: Save Phase Data to CSV
        section_title21 = Tex(r"20 - Save Phase Data to CSV:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation21 = Tex(
            r"\textbf{Saves the 2D phase data} as a CSV file.",
```

```
            font_size=28,
            color=BLACK
        ).next_to(section_title21, DOWN)
        code21 = Code(
            code="""pd.DataFrame(phase_deg_2d)
.to_csv(phase_data_file,
header=False, index=False)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation21, DOWN)

        self.play(TransformMatchingShapes(section_title20, section_title21), run_time=1)
        self.play(TransformMatchingShapes(explanation20, explanation21),
TransformMatchingShapes(code20, code21), run_time=1)
        self.wait(2)

        # Part 21: Save Patch Data to JSON
        section_title22 = Tex(r"21 - Save Patch Data to JSON:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation22 = Tex(
            r"\textbf{Saves the patch data} as a JSON file.",
            font_size=28,
            color=BLACK
        ).next_to(section_title22, DOWN)
        code22 = Code(
            code="""pd.DataFrame(patch_data).to_json(output_file, orient="records")""",
```

```
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation22, DOWN)

        self.play(TransformMatchingShapes(section_title21, section_title22), run_time=1)
        self.play(TransformMatchingShapes(explanation21, explanation22),
TransformMatchingShapes(code21, code22), run_time=1)
        self.wait(2)

        # Part 22: Execute the Main Function
        section_title23 = Tex(r"22 - Execute the Main Function:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation23 = Tex(
            r"\textbf{Calls the pre\_calculate\_data function} if the script is run directly.",
            font_size=28,
            color=BLACK
        ).next_to(section_title23, DOWN)
        code23 = Code(
            code="""if __name__ == "__main__":
pre_calculate_data("patch_data.json", "phase_data.csv")""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation23, DOWN)
```

```
        self.play(TransformMatchingShapes(section_title22, section_title23), run_time=1)
        self.play(TransformMatchingShapes(explanation22, explanation23),
TransformMatchingShapes(code22, code23), run_time=1)
        self.wait(2)



# Continuing within the same file (python_code_generate_hfss.py)
from manim import *

class GenerateHFSSScript(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Main Title for the slide
        main_title = Tex(r"generate\_hfss\_script.py Code Explanation", font_size=48, color=BLACK)
        main_title.to_edge(UP)

        # Position for text and code elements
        explanation_position = UP * 2

        # Part 1: Importing Packages
        section_title1 = Tex(r"1 - Importing Packages:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation1 = Tex(
            r"\textbf{ScriptEnv}: HFSS package for interacting with Ansys Electronics Desktop.\\"
            r"\textbf{json}: Reads the pre-calculated patch data stored in a JSON file.",
```

```
            font_size=28,
            color=BLACK
        ).next_to(section_title1, DOWN)
        code1 = Code(
            code="""import ScriptEnv
import json""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation1, DOWN)

        self.play(Write(main_title), run_time=1)
        self.play(Write(section_title1), run_time=1)
        self.play(Write(explanation1), run_time=1)
        self.play(Write(code1), run_time=1)
        self.wait(2)

        # Part 2: Defining the Main Function
        section_title2 = Tex(r"2 - Defining the Main Function (main):", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation2 = Tex(
            r"\textbf{Purpose}: Serves as the entry point of the script, encapsulating all logic.",
            font_size=28,
            color=BLACK
        ).next_to(section_title2, DOWN)
        code2 = Code(
            code="""def main():
```

..."""",

tab_width=4,

language="python",

font="Monospace",

background="window",

insert_line_no=False

).next_to(explanation2, DOWN)

self.play(TransformMatchingShapes(section_title1, section_title2), run_time=1)

self.play(TransformMatchingShapes(explanation1, explanation2), TransformMatchingShapes(code1, code2), run_time=1)

self.wait(2)

Part 3: Initializing HFSS Environment

section_title3 = Tex(r"3 - Initializing HFSS Environment:", font_size=36, color=BLACK).move_to(explanation_position)

explanation3 = Tex(

r"\textbf{ScriptEnv.Initialize}: Initializes HFSS environment.\\"

r"\textbf{oDesktop.RestoreWindow}: Restores the main HFSS window.",

font_size=28,

color=BLACK

).next_to(section_title3, DOWN)

code3 = Code(

code="""ScriptEnv.Initialize("Ansoft.ElectronicsDesktop")
oDesktop.RestoreWindow()""",

tab_width=4,

language="python",

font="Monospace",

background="window",

```
            insert_line_no=False
        ).next_to(explanation3, DOWN)

        self.play(TransformMatchingShapes(section_title2, section_title3), run_time=1)
        self.play(TransformMatchingShapes(explanation2, explanation3), TransformMatchingShapes(code2, code3), run_time=1)
        self.wait(2)

        # Part 4: Setting Up Active Project and Design
        section_title4 = Tex(r"4 - Setting Up Active Project and Design:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation4 = Tex(
            r"\textbf{oProject}: Sets the active project.\\"
            r"\textbf{oDesign}: Sets the active design.\\"
            r"\textbf{oEditor}: Sets the active editor.",
            font_size=28,
            color=BLACK
        ).next_to(section_title4, DOWN)
        code4 = Code(
            code="""oProject = oDesktop.SetActiveProject("Project1")
oDesign = oProject.SetActiveDesign("HFSSDesign1")
oEditor = oDesign.SetActiveEditor("3D Modeler")""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation4, DOWN)
```

```
        self.play(TransformMatchingShapes(section_title3, section_title4), run_time=1)
        self.play(TransformMatchingShapes(explanation3, explanation4), TransformMatchingShapes(code3, code4), run_time=1)
        self.wait(2)

        # Part 5: Loading Pre-Calculated Patch Data
        section_title5 = Tex(r"5 - Loading Pre-Calculated Patch Data:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation5 = Tex(
            r"\textbf{Purpose}: Loads the pre-calculated patch data from a JSON file.",
            font_size=28,
            color=BLACK
        ).next_to(section_title5, DOWN)
        code5 = Code(
            code="""with open("C:\\Users\\...\\patch_data.json", "r") as f:
patch_data = json.load(f)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation5, DOWN)

        self.play(TransformMatchingShapes(section_title4, section_title5), run_time=1)
        self.play(TransformMatchingShapes(explanation4, explanation5), TransformMatchingShapes(code4, code5), run_time=1)
        self.wait(2)

        # Part 6: User-Defined Dimensions
```

```
        section_title6 = Tex(r"6 - User-Defined Dimensions:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation6 = Tex(
            r"\textbf{Purpose}: Sets dimensions of each layer and other parameters such as cell spacing.",
            font_size=28,
            color=BLACK
        ).next_to(section_title6, DOWN)
        code6 = Code(
            code="""ground_height = 0.1
dielectric_height = 1.0
top_patch_height = 0.1
dielectric_material = "Rogers RO3203 (tm)"
uedim = 5""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation6, DOWN)

        self.play(TransformMatchingShapes(section_title5, section_title6), run_time=1)
        self.play(TransformMatchingShapes(explanation5, explanation6), TransformMatchingShapes(code5, code6), run_time=1)
        self.wait(2)

        # Part 7: Loop Through the Pre-Calculated Patch Data
        section_title7 = Tex(r"7 - Loop Through the Pre-Calculated Patch Data:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation7 = Tex(
```

```
        r"\textbf{Purpose}: Loops through each entry in the pre-calculated patch data and extracts values.",
        font_size=28,
        color=BLACK
    ).next_to(section_title7, DOWN)
    code7 = Code(
        code="""for i, patch in enumerate(patch_data):
xi = patch["xi"]
yi = patch["yi"]
zi = patch["zi"]
patch_size = patch["patch_size"]""",
        tab_width=4,
        language="python",
        font="Monospace",
        background="window",
        insert_line_no=False
    ).next_to(explanation7, DOWN)

    self.play(TransformMatchingShapes(section_title6, section_title7), run_time=1)
    self.play(TransformMatchingShapes(explanation6, explanation7), TransformMatchingShapes(code6, code7), run_time=1)
    self.wait(2)

    # Part 8: Centering the Top Patches
    section_title8 = Tex(r"8 - Centering the Top Patches:", font_size=36, color=BLACK).move_to(explanation_position)
    explanation8 = Tex(
        r"\textbf{Purpose}: Shifts the center of the top patches by half the cell spacing.",
        font_size=28,
        color=BLACK
```

```
        ).next_to(section_title8, DOWN)
        code8 = Code(
            code="""xi_center = xi + uedim / 2
yi_center = yi + uedim / 2""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation8, DOWN)

        self.play(TransformMatchingShapes(section_title7, section_title8), run_time=1)
        self.play(TransformMatchingShapes(explanation7, explanation8), TransformMatchingShapes(code7, code8), run_time=1)
        self.wait(2)

        # Part 9: Creating the Ground Layer
        section_title9 = Tex(r"9 - Creating the Ground Layer:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation9 = Tex(
            r"\textbf{Purpose}: Creates a rectangular box representing the ground layer.",
            font_size=28,
            color=BLACK
        ).next_to(section_title9, DOWN)
        code9 = Code(
            code="""oEditor.CreateBox(
    [
"NAME:BoxParameters",
"XPosition:=", "{}mm".format(xi),
```

```
"YPosition:=", "{}mm".format(yi),
"ZPosition:=", "{}mm".format(zi),
"XSize:=", "{}mm".format(uedim),
"YSize:=", "{}mm".format(uedim),
"ZSize:=", "{}mm".format(ground_height)
    ],
    [
"NAME:Attributes",
"Name:=", "Ground_{}".format(i),
"MaterialValue:=", "\"pec\"",
"SolveInside:=", False
    ]
)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        )
        code9.scale(0.7)
        code9.to_edge(DOWN)
        self.play(TransformMatchingShapes(section_title8, section_title9), run_time=1)
        self.play(TransformMatchingShapes(explanation8, explanation9), TransformMatchingShapes(code8, code9), run_time=1)
        self.wait(2)

        # Part 10: Creating the Dielectric (PCB) Layer
        section_title10 = Tex(r"10 - Creating the Dielectric (PCB) Layer:", font_size=36, color=BLACK).move_to(explanation_position)
```

```
        explanation10 = Tex(
            r"\textbf{Purpose}: Creates a rectangular box for the dielectric (PCB) layer on top of the ground layer.",
            font_size=28,
            color=BLACK
        ).next_to(section_title10, DOWN)
        code10 = Code(
            code="""oEditor.CreateBox(
    [
"NAME:BoxParameters",
"XPosition:=", "{}mm".format(xi),
"YPosition:=", "{}mm".format(yi),
"ZPosition:=", "{}mm".format(zi + ground_height),
"XSize:=", "{}mm".format(uedim),
"YSize:=", "{}mm".format(uedim),
"ZSize:=", "{}mm".format(dielectric_height)
    ],
    [
"NAME:Attributes",
"Name:=", "Dielectric_{}".format(i),
"MaterialValue:=", "\"{}\"".format(dielectric_material),
"SolveInside:=", True,
"Color:=", "(128 128 128)"
    ]
)""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
```

```
            insert_line_no=False
        )
        code10.scale(0.6)
        code10.to_edge(DOWN)
        self.play(TransformMatchingShapes(section_title9, section_title10), run_time=1)
        self.play(TransformMatchingShapes(explanation9, explanation10),
TransformMatchingShapes(code9, code10), run_time=1)
        self.wait(2)


        # Part 11: Creating the Top Metallic Patch
        section_title11 = Tex(r"11 - Creating the Top Metallic Patch:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation11 = Tex(
            r"\textbf{Purpose}: Creates a rectangular box for the top metallic patch centered within the cell.",
            font_size=28,
            color=BLACK
        ).next_to(section_title11, DOWN)
        code11 = Code(
            code="""oEditor.CreateBox(
    [
"NAME:BoxParameters",
"XPosition:=", "{}mm".format(xi_center - patch_size / 2),
"YPosition:=", "{}mm".format(yi_center - patch_size / 2),
 "ZPosition:=", "{}mm".format(zi + ground_height + dielectric_height),
 "XSize:=", "{}mm".format(patch_size),
"YSize:=", "{}mm".format(patch_size),
 "ZSize:=", "{}mm".format(top_patch_height)
    ],
    [
```

```
"NAME:Attributes",
"Name:=", "TopPatch_{}".format(i),
 "MaterialValue:=", "\"pec\"",
"SolveInside:=", False
   ]
)""",
          tab_width=4,
          language="python",
          font="Monospace",
          background="window",
          insert_line_no=False
       )
       code11.scale(0.65)
       code11.to_edge(DOWN)
       self.play(TransformMatchingShapes(section_title10, section_title11), run_time=1)
       self.play(TransformMatchingShapes(explanation10, explanation11),
TransformMatchingShapes(code10, code11), run_time=1)
       self.wait(2)


       # Part 12: Execute the Main Function
       section_title12 = Tex(r"12 - Execute the Main Function:", font_size=36,
color=BLACK).move_to(explanation_position)
       explanation12 = Tex(
          r"\textbf{Purpose}: Ensures the main function is called only when the script is executed directly.",
          font_size=28,
          color=BLACK
       ).next_to(section_title12, DOWN)
       code12 = Code(
          code="""if __name__ == "__main__":
```

```
main()""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation12, DOWN)

        self.play(TransformMatchingShapes(section_title11, section_title12), run_time=1)
        self.play(TransformMatchingShapes(explanation11, explanation12),
TransformMatchingShapes(code11, code12), run_time=1)
        self.wait(2)

# Continuing within the same file (plotter_code_explanation.py)
from manim import *

class PlotterCodeExplanation(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Main Title for the slide
        main_title = Tex(r"Plotter.py Code Explanation", font_size=48, color=BLACK)
        main_title.to_edge(UP)

        # Position for text and code elements
        explanation_position = UP * 1.5

        # Part 1: Importing Packages
```

```
        section_title1 = Tex(r"1 - Importing Packages:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation1 = Tex(
            r"\textbf{numpy (np)}: Numerical operations and efficient array manipulations.\\"
            r"\textbf{pandas (pd)}: Reads and handles tabular data from CSV files.\\"
            r"\textbf{matplotlib.pyplot (plt)}: Creates visualizations like plotting 2D phase distributions.",
            font_size=28,
            color=BLACK
        ).next_to(section_title1, DOWN)
        code1 = Code(
            code="""import numpy as np
import pandas as pd
import matplotlib.pyplot as plt""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation1, DOWN)

        self.play(Write(main_title), run_time=1)
        self.play(Write(section_title1), run_time=1)
        self.play(FadeIn(explanation1), run_time=1)
        self.play(FadeIn(code1), run_time=1)
        self.wait(2)

        # Part 2: Defining plot_phase_data Function
        section_title2 = Tex(r"2 - Defining plot\_phase\_data Function:", font_size=36,
color=BLACK).move_to(explanation_position)
```

```
        explanation2 = Tex(
            r"\textbf{Purpose}: Defines a function to read 2D phase data and create a plot saved as an image.\\"
            r"\textbf{Parameters}: \textit{phase\_data\_file} and \textit{output\_image\_file}.",
            font_size=28,
            color=BLACK
        ).next_to(section_title2, DOWN)
        code2 = Code(
            code="""def plot_phase_data(phase_data_file="phase_data.csv", output_image_file="phase_plot.png"):
..."""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation2, DOWN)

        self.play(TransformMatchingShapes(section_title1, section_title2), run_time=1)
        self.play(TransformMatchingShapes(explanation1, explanation2), TransformMatchingShapes(code1, code2), run_time=1)
        self.wait(2)

        # Part 3: Loading the 2D Phase Data
        section_title3 = Tex(r"3 - Loading the 2D Phase Data:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation3 = Tex(
            r"\textbf{Purpose}: Reads the phase data from a CSV file using pandas.",
            font_size=28,
            color=BLACK
```

```
        ).next_to(section_title3, DOWN)
        code3 = Code(
            code="""phase_deg_2d = pd.read_csv(phase_data_file,
header=None).to_numpy()""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation3, DOWN)


        self.play(TransformMatchingShapes(section_title2, section_title3), run_time=1)
        self.play(TransformMatchingShapes(explanation2, explanation3), TransformMatchingShapes(code2,
code3), run_time=1)
        self.wait(2)


        # Part 4: Plotting the Phase Data
        section_title4 = Tex(r"4 - Plotting the Phase Data:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation4 = Tex(
            r"\textbf{Purpose}: Plots the 2D array using matplotlib. Uses 'jet' colormap and sets axis labels.",
            font_size=28,
            color=BLACK
        ).next_to(section_title4, DOWN)
        code4 = Code(
            code="""plt.figure(figsize=(8, 8))
plt.imshow(phase_deg_2d, cmap='jet', origin='lower')
plt.colorbar(label="Phase (degrees)")
plt.title("Phase Distribution")
```

```
plt.xlabel("X (mm)")
plt.ylabel("Y (mm)")""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation4, DOWN)

        self.play(TransformMatchingShapes(section_title3, section_title4), run_time=1)
        self.play(TransformMatchingShapes(explanation3, explanation4), TransformMatchingShapes(code3, code4), run_time=1)
        self.wait(2)

        # Part 5: Saving the Plot as an Image
        section_title5 = Tex(r"5 - Saving the Plot as an Image:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation5 = Tex(
            r"\textbf{Purpose}: Saves the plotted image to the specified output file.",
            font_size=28,
            color=BLACK
        ).next_to(section_title5, DOWN)
        code5 = Code(
            code="""plt.savefig(output_image_file)
plt.close()""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
```

```
            insert_line_no=False
        ).next_to(explanation5, DOWN)

        self.play(TransformMatchingShapes(section_title4, section_title5), run_time=1)
        self.play(TransformMatchingShapes(explanation4, explanation5), TransformMatchingShapes(code4, code5), run_time=1)
        self.wait(2)

        # Part 6: Calling the Main Function
        section_title6 = Tex(r"6 - Calling the Main Function:", font_size=36, color=BLACK).move_to(explanation_position)
        explanation6 = Tex(
            r"\textbf{Purpose}: Executes the \textit{plot\_phase\_data} function when the script is run directly.",
            font_size=28,
            color=BLACK
        ).next_to(section_title6, DOWN)
        code6 = Code(
            code="""if __name__ == "__main__":
plot_phase_data("phase_data.csv", "phase_plot.png")""",
            tab_width=4,
            language="python",
            font="Monospace",
            background="window",
            insert_line_no=False
        ).next_to(explanation6, DOWN)

        self.play(TransformMatchingShapes(section_title5, section_title6), run_time=1)
        self.play(TransformMatchingShapes(explanation5, explanation6), TransformMatchingShapes(code5, code6), run_time=1)
```

```
self.wait(2)
```

```
# File: testttttttttttttttt
from manim import *

class PythonCodeAutomatedHFSS3(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Main Title for the slide
        main_title = Text(r"Jason_data_creator.py Code Explanation", font_size=48, color=BLACK)
        main_title.to_edge(UP)

        # Position for text and code
        explanation_position = UP * 2
        code_position = UP* 0.5



        # Part 11: Creating the Top Metallic Patch
        section_title11 = Tex(r"11 - Creating the Top Metallic Patch:", font_size=36,
color=BLACK).move_to(explanation_position)
        explanation11 = Tex(
```

```
        r"\textbf{Purpose}: Creates a rectangular box for the top metallic patch centered within the cell.",
        font_size=28,
        color=BLACK
    ).next_to(section_title11, DOWN)
    code11 = Code(
        code="""oEditor.CreateBox(
  [
"NAME:BoxParameters",
"XPosition:=", "{}mm".format(xi_center - patch_size / 2),
"YPosition:=", "{}mm".format(yi_center - patch_size / 2),
 "ZPosition:=", "{}mm".format(zi + ground_height + dielectric_height),
 "XSize:=", "{}mm".format(patch_size),
"YSize:=", "{}mm".format(patch_size),
 "ZSize:=", "{}mm".format(top_patch_height)
  ],
  [
"NAME:Attributes",
"Name:=", "TopPatch_{}".format(i),
 "MaterialValue:=", "\"pec\"",
"SolveInside:=", False
  ]
)""",
        tab_width=4,
        language="python",
        font="Monospace",
        background="window",
        insert_line_no=False
    )
    code11.scale(0.65)
```

```
        code11.to_edge(DOWN)
        # Transition to the Example Values section
        self.add (main_title)
        self.add (section_title11)
        self.add(explanation11)
        self.add(code11)



# Continuing within the same file (thank_you_slide.py)
from manim import *

class ThankYouSlide(Scene):
    def construct(self):
        # Set background color to white
        self.camera.background_color = WHITE

        # Main Title for the slide
        main_title = Tex(r"Thank You for Watching!", font_size=72, color=BLACK)
        main_title.move_to(ORIGIN)

        # Animations for the final slide
        self.play(Write(main_title), run_time=2)
        self.wait(4)
```