Preparing Graphics for Design & Prototyping II

OA Fakinlede

oafak@unilag.edu.ng lms.s2pafrica.com

Previous Work & Strategy

- Our study on Modeling & Simulation raises several issues that we address using the following simulations capabilities:
  1. Linear & Nonlinear Simulation for stress and failure analysis including elastic, plastic and other nonlinear material models.
  2. Shape Optimization
  3. Event Simulation for motion and dynamic analysis
  4. Computational Fluid Dynamics for fluid-solid interactions, optimization and turbomachinery design. Energy systems and many others
- Virtual Laboratory.
  - The analyses we can do essentially constitute a virtual laboratory for testing designs to prototyping

Last Week

- A Quick Introduction to the Fusion 360 API:
  - Two languages are presently supported: Python and C++
    - For both, Autodesk supports the Visual Studio Code Multilanguage, Multi Platform Integrated Development Environment, IDE
  - It is important to note:
    - Other IDEs, some even arguably better than VSC IDE are not directly supported.
    - The latest Python Interpreter, version 3.8.3 is not yet supported. We have good experience, in Fusion 360 API, with version 3.7.6 and lower. Do not be eaget to go for the very latest.
    - Also do not use the latest Python Extensions for Visual Studio Code. Support appears assured up till the 2019.9.34911 released 10 months ago.
  - Good News is that Visual Studio Code can manage everything and allow you to choose what to use for any specific project.

Install Python and Visual Studio Code

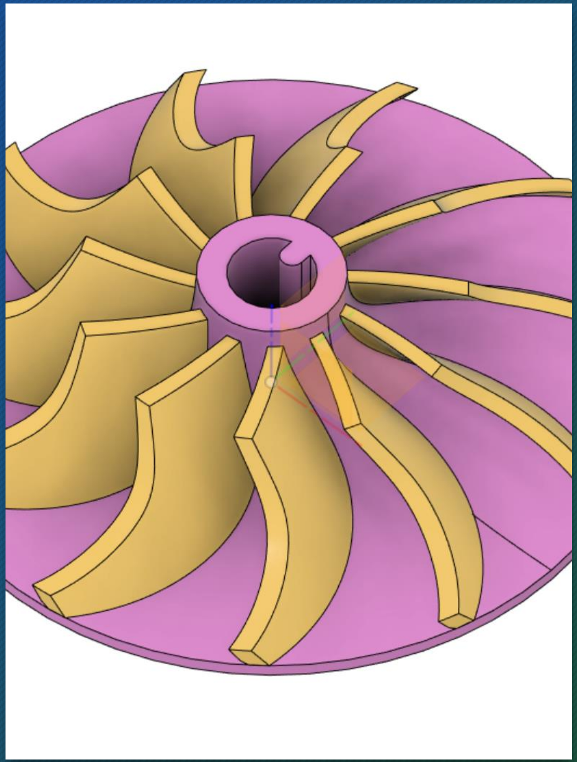
- We will respond to any questions you may have in starting off in programming the API.
  - The hardest part is to begin on a firm footing. Once you are able to run the simple codes we are discussing in class, view and play with the examples in the Fusion installation that you have, then you are off and running.
  - Note that you will need to learn Python seriously. Practice makes perfect. Just keep at it. Everyone struggles and only perseverance will win.
  - The bigger issue is the humongous API library itself. It will take a longer time to master it. But you will be productive all the way. Again, perseverance is the watchword.

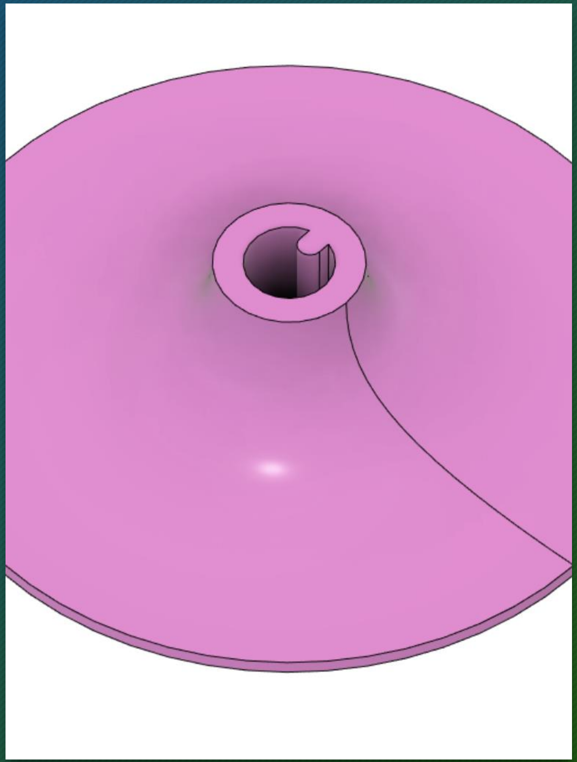
Continuing the Turbine Design

- Last week we started with modeling the Turbine Volute. I sent my code and I hope several of you looked at the simple steps involved.
- This week, the lecture continues to the model of the Impeller.
- We can do this by drawing it directly in Fusion 360. It is quite straightforward depending on how fancy a blade profile you want to start with.
- You can also do this by calling the API. In my presentation today, I will do both.

No Blade, No impeller

- If you remove all the blades on a regular impeller, what do you have left?
  - Either a circular plate with or without a hole in the center; or
  - A plate with an axi-symmetric bulge in the middle, and a hole in the center.
- Either one of these will take an average member of this class less than ten minutes to model!
- I have therefore decided to focus on where the issues are: The blades, the blades the blades!
- Without the blades, you are looking at three extrusions and a revolve with a slot.





Creating Blades in Python

You can simply extrude a simple arc profile, do a pattern replication of the same and get a decent impeller.

When you get to Computational Fluid Dynamics, you will want to find things to do to make the turbine still more efficient. Many of these things center around blade geometry.

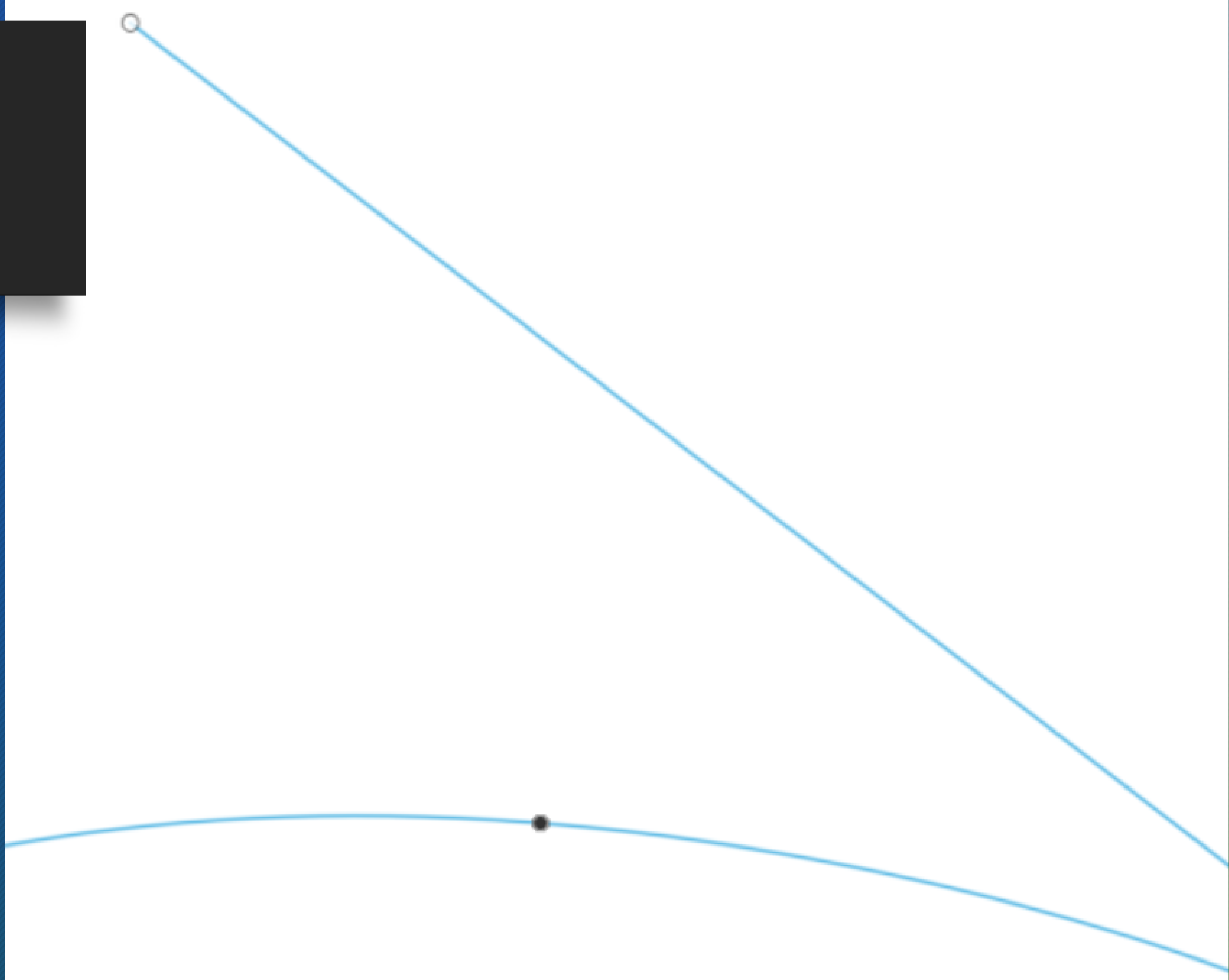
Companies in other countries are spending a lot of time fine tuning blade geometry using all sorts of mathematical profiles, curves, transformations and other automations to make blades on the fly.

Programming Blade Geometry

- It is a good idea to begin to play with the idea that such geometrical modifications can happen easily when you write a computer program to crate the blade.
- We will continue here with a small effort in that direction.

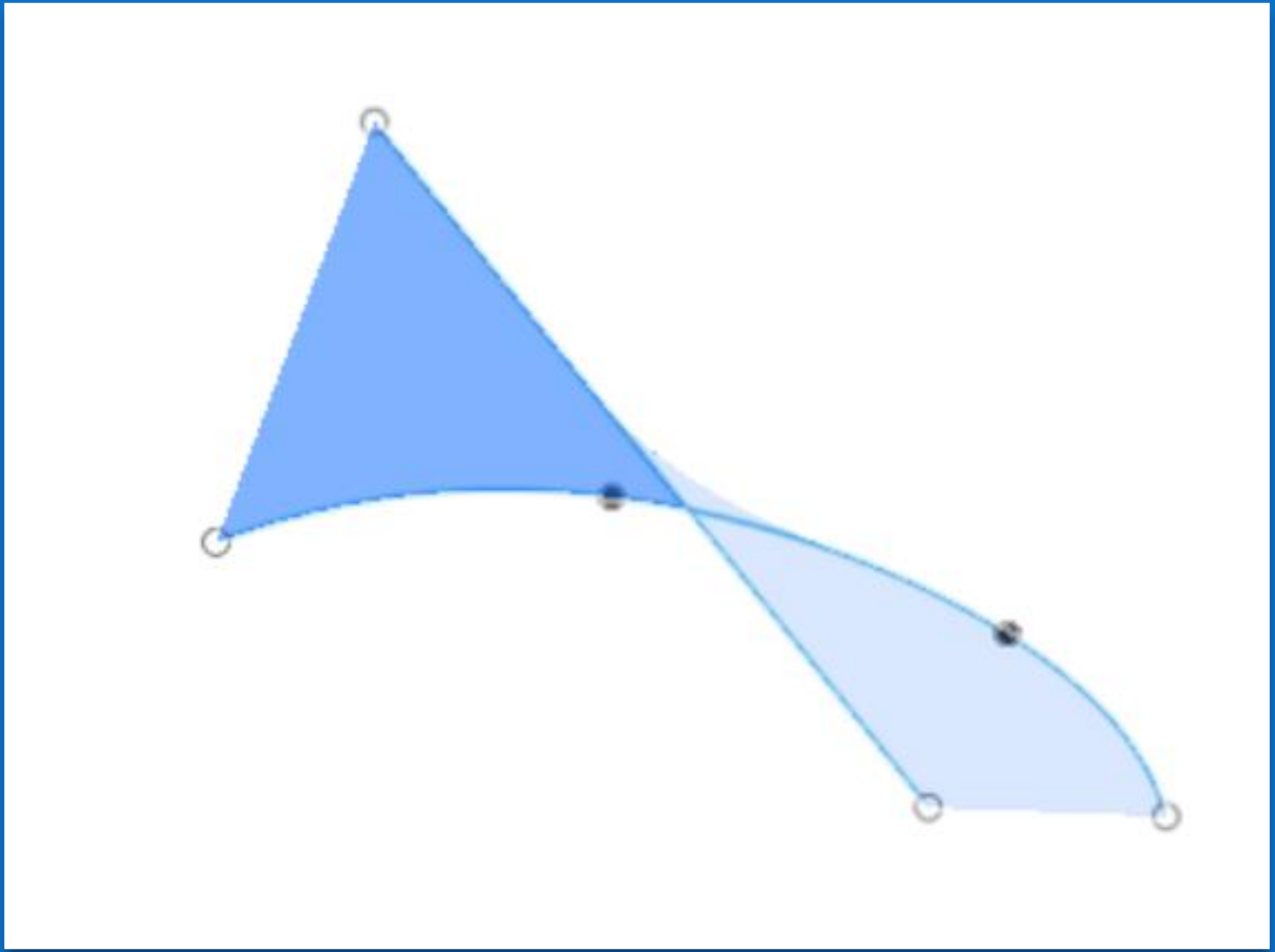
Diving In

- In the figure shown here, you can get the blade profile from a spline on the base plane and a straight line at the top part of the blade. In this drawing, I used 23 as the height (between the planes of the lines)
- The rest is easy:
  - Simply do a surface loft between the two of them.
  - Fusion 360 finds the spline rail to the edges.
  - Here, I thickened the ensuing surface
- Three or four easy steps in Fusion 360 Modeling



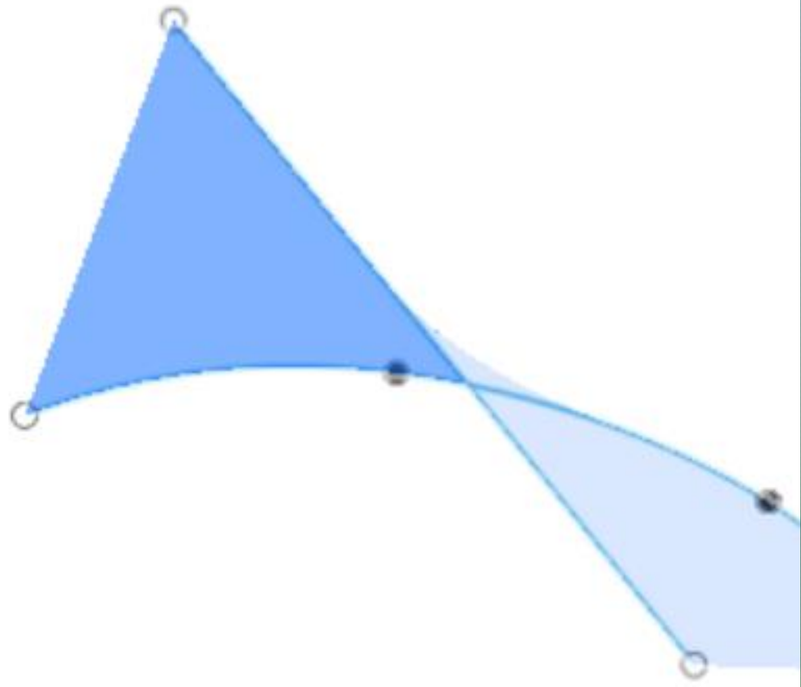
Diving In

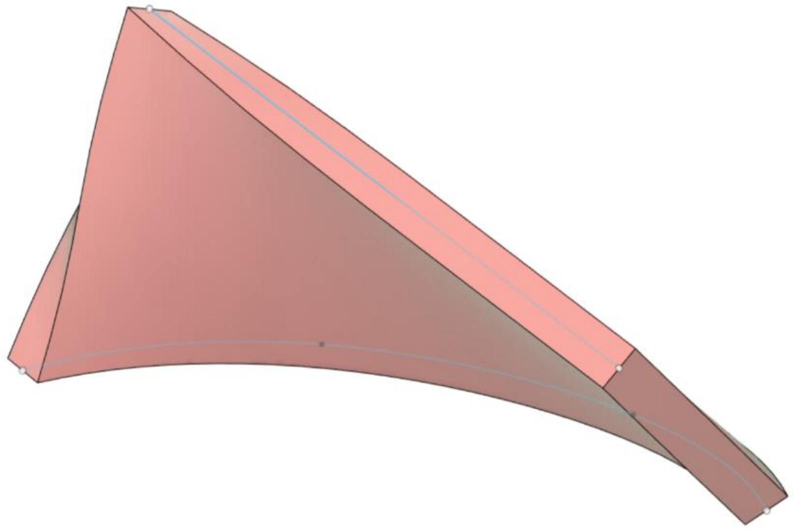
- In the figure shown here, you can get the blade profile from a spline on the base plane and a straight line at the top part of the blade. In this drawing, I used 23 as the height (between the planes of the lines)
- The rest is easy:
    - Simply do a surface loft between the two of them.
    - Fusion 360 finds the spline rail to the edges.
    - Here, I thickened the ensuing surface
- Three or four easy steps in Fusion 360 Modeling



Diving In

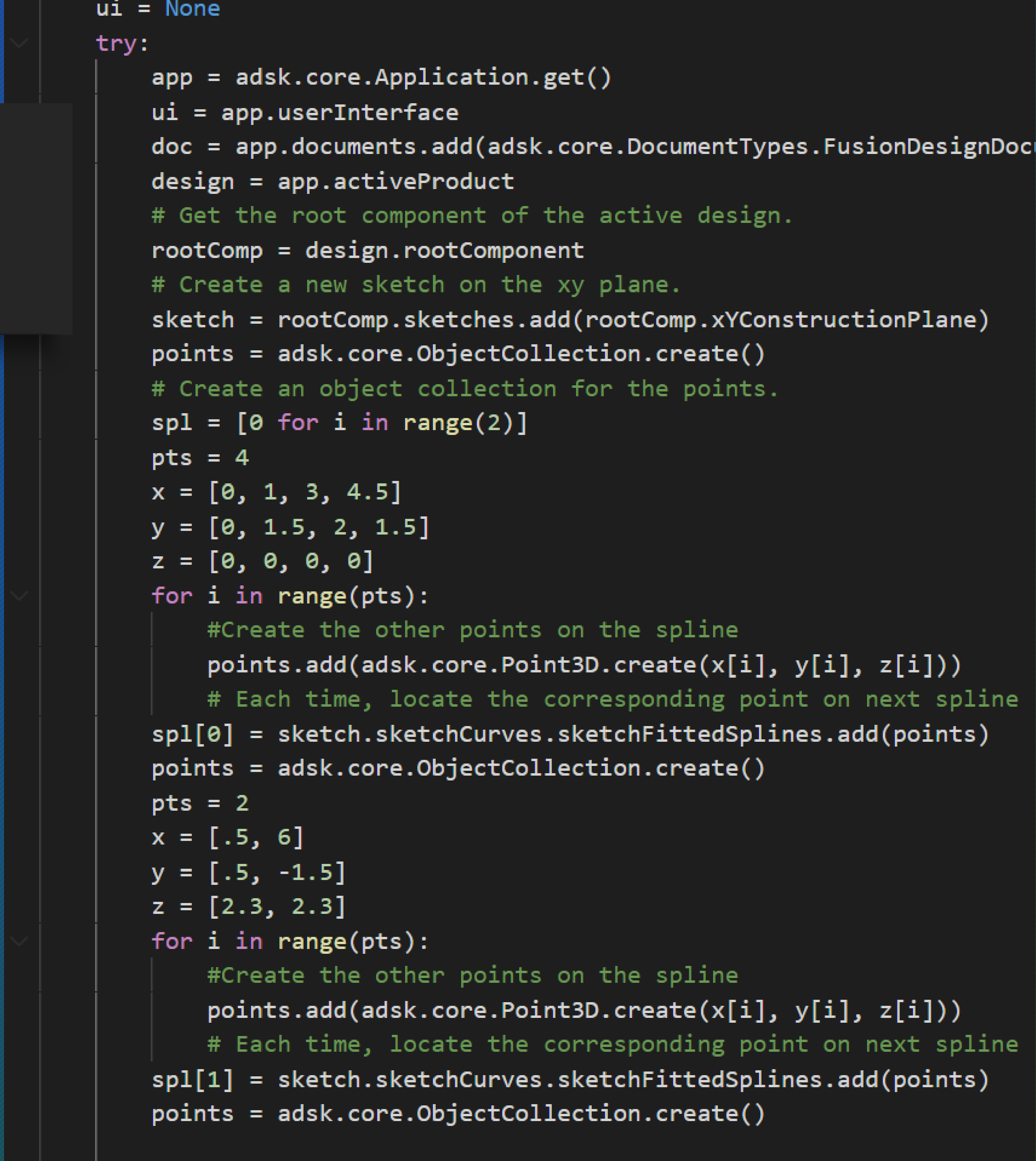
- In the figure shown here, you can get the blade profile from a spline on the base plane and a straight line at the top part of the blade. In this drawing, I used 23 as the height (between the planes of the lines)
- The rest is easy:
  - Simply do a surface loft between the two of them.
  - Fusion 360 finds the spline rail to the edges.
  - Here, I thickened the ensuing surface
- Three or four easy steps in Fusion 360 Modeling





Python Blade

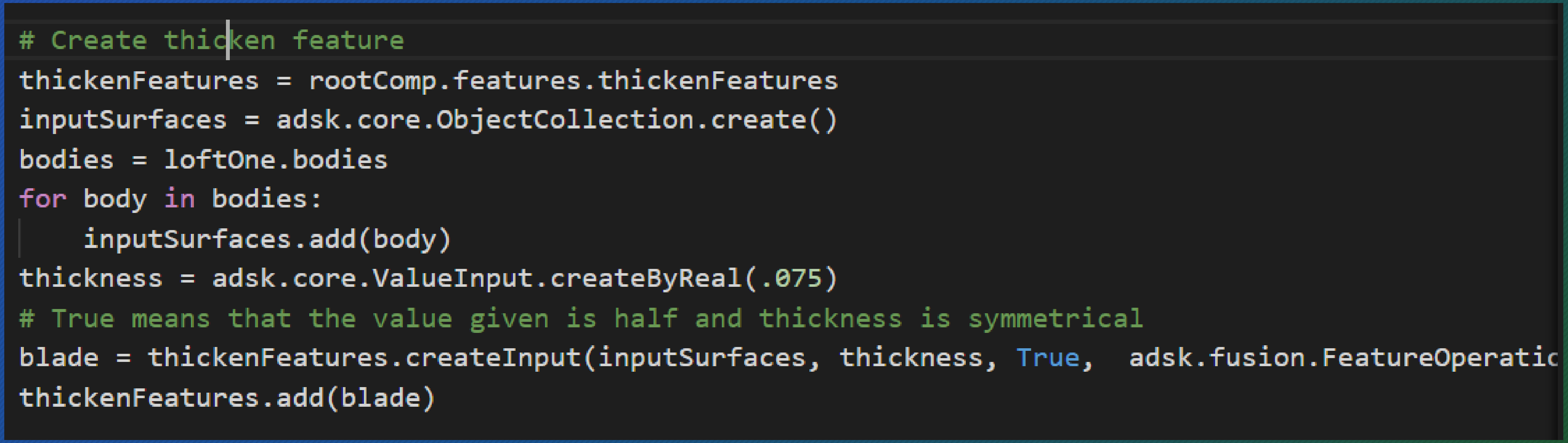
- The code here creates the spline and the straight line. As usual, 30% of the code is written by Fusion 360 for you. You only need to worry about the logic and some housekeeping.
- You easily recognize the data for the x, y and z coordinates of the four spline points, and the two points of the upper line.
- In this place, I used the spline to create the straight line from two points. Later in the same code, I used another Fusion construct to do the same. Its all up to what you prefer.
- One important difference to note here:
    - Whereas, you create lines, splines, etc., on sketch planes. These are not needed when you are programming. You can always get any line at any time on any plane or direction by supplying its coordinates.



```
ui = None
try:
    app = adsk.core.Application.get()
    ui = app.userInterface
    doc = app.documents.add(adsk.core.DocumentTypes.FusionDesignDoc
    design = app.activeProduct
    # Get the root component of the active design.
    rootComp = design.rootComponent
    # Create a new sketch on the xy plane.
    sketch = rootComp.sketches.add(rootComp.xYConstructionPlane)
    points = adsk.core.ObjectCollection.create()
    # Create an object collection for the points.
    spl = [0 for i in range(2)]
    pts = 4
    x = [0, 1, 3, 4.5]
    y = [0, 1.5, 2, 1.5]
    z = [0, 0, 0, 0]
    for i in range(pts):
        #Create the other points on the spline
        points.add(adsk.core.Point3D.create(x[i], y[i], z[i]))
        # Each time, locate the corresponding point on next spline
    spl[0] = sketch.sketchCurves.sketchFittedSplines.add(points)
    points = adsk.core.ObjectCollection.create()
    pts = 2
    x = [.5, 6]
    y = [.5, -1.5]
    z = [2.3, 2.3]
    for i in range(pts):
        #Create the other points on the spline
        points.add(adsk.core.Point3D.create(x[i], y[i], z[i]))
        # Each time, locate the corresponding point on next spline
    spl[1] = sketch.sketchCurves.sketchFittedSplines.add(points)
    points = adsk.core.ObjectCollection.create()
```

Thicken: Housekeeping

- Now you can thicken the surface created by lofting the curve to the line by the following code:

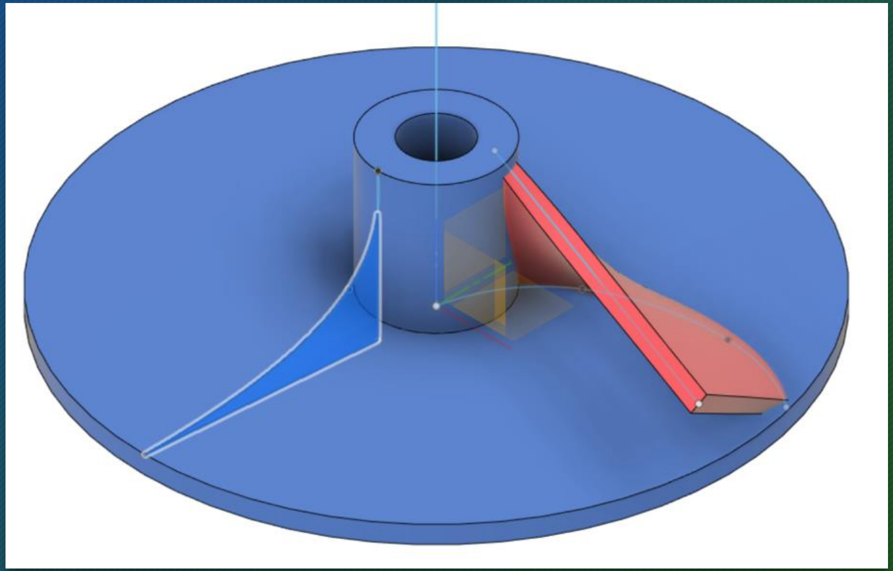


```
# Create thicken feature
thickenFeatures = rootComp.features.thickenFeatures
inputSurfaces = adsk.core.ObjectCollection.create()
bodies = loftOne.bodies
for body in bodies:
    inputSurfaces.add(body)
thickness = adsk.core.ValueInput.createByReal(.075)
# True means that the value given is half and thickness is symmetrical
blade = thickenFeatures.createInput(inputSurfaces, thickness, True,  adsk.fusion.FeatureOperatic
thickenFeatures.add(blade)
```

Extruding and revolving your way

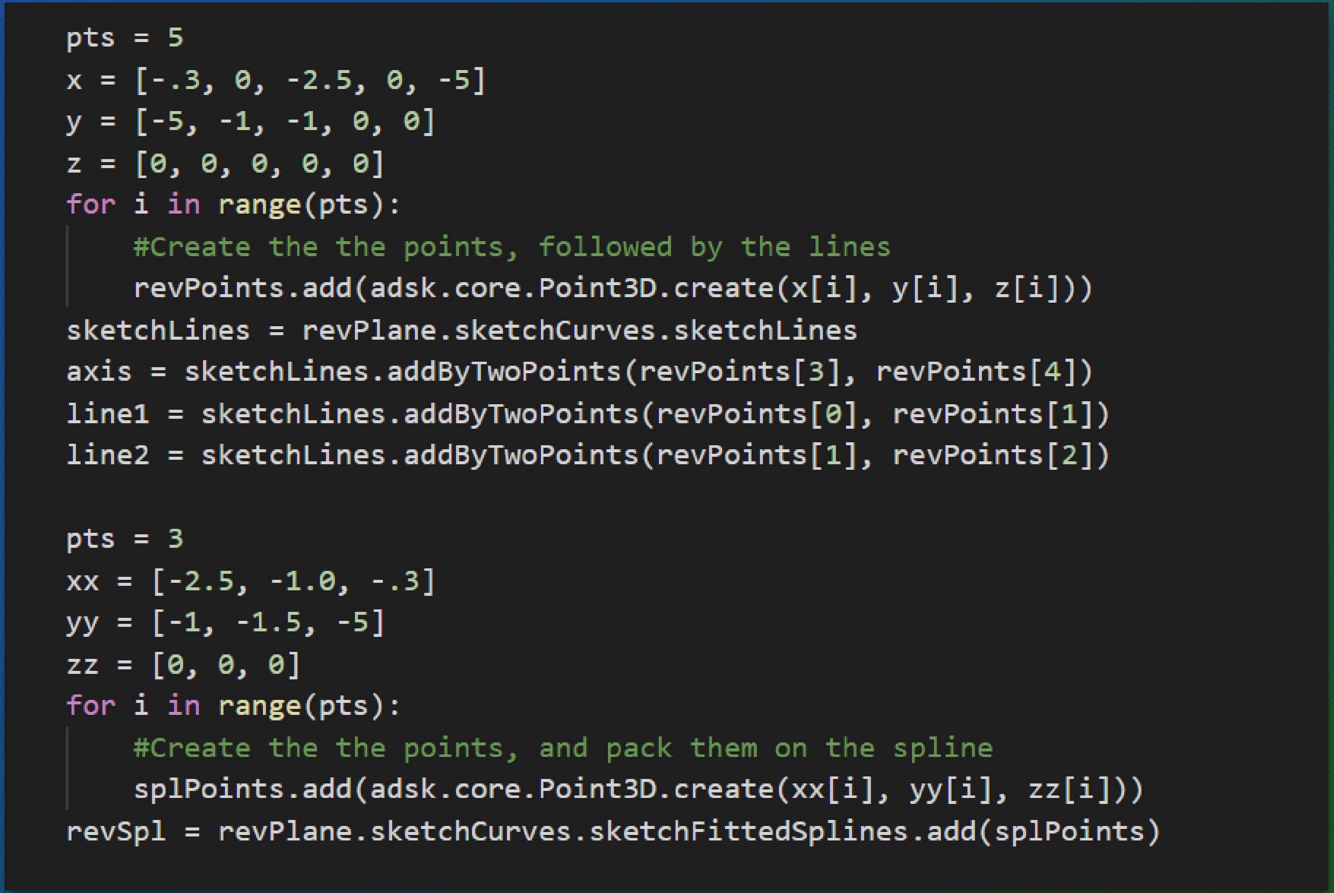
- The code so far gets you from scratch to the picture on top.
- How do you get the picture below?
  - Simple answer, you extrude your way from top to bottom!

  1. Create the surface spline that you can rotate and get the bulging shape of the impeller plate. Can be done programmatically, by it is more tedious.
  2. Create a cutting extrusion to cut the blade any way you like before you do a circular pattern replication.
  3. The revolve profile created here was done in Python. You will be surprised that it to me several hours to do: There is a bewildering issue in Fusion programming that can cause new users a lot of confusion. This happens once you are using coordinates wrt created planes. Fusion does not necessarily behave as you expect. It took me sometime to understand why.



Revolve Profile code

- For those that are keen, here is my code for the face of the impeller.
- I was exhausted by the time I completed it, I therefore continued the rest of the impeller for this lecture manually. Will go back later.
- Take a look at the coordinates that generated the curves here! They are completely different from the global coordinates of the locations. For more explanations, see:
- Brian Ekins is the creator of the Fusion 360 API and you can see some comments in reply to my questions here



```python
pts = 5
x = [-.3, 0, -2.5, 0, -5]
y = [-5, -1, -1, 0, 0]
z = [0, 0, 0, 0, 0]
for i in range(pts):
    #Create the the points, followed by the lines
    revPoints.add(adsk.core.Point3D.create(x[i], y[i], z[i]))
sketchLines = revPlane.sketchCurves.sketchLines
axis = sketchLines.addByTwoPoints(revPoints[3], revPoints[4])
line1 = sketchLines.addByTwoPoints(revPoints[0], revPoints[1])
line2 = sketchLines.addByTwoPoints(revPoints[1], revPoints[2])

pts = 3
xx = [-2.5, -1.0, -.3]
yy = [-1, -1.5, -5]
zz = [0, 0, 0]
for i in range(pts):
    #Create the the points, and pack them on the spline
    splPoints.add(adsk.core.Point3D.create(xx[i], yy[i], zz[i]))
revSpl = revPlane.sketchCurves.sketchFittedSplines.add(splPoints)
```

Cutting Revolve

- A cutting revolve on the blade followed by a pattern replication produces the result shown here: