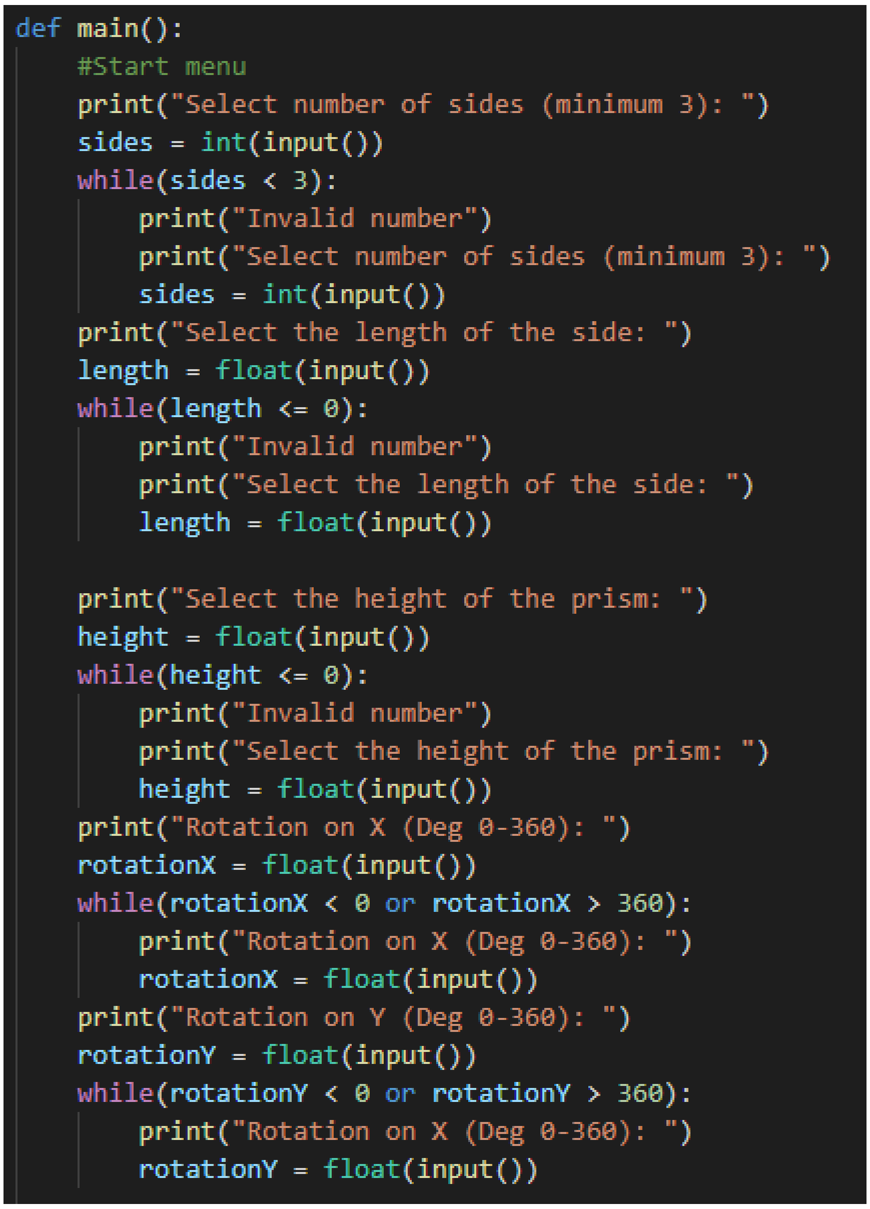
Adjustable STL Creator for creating prisms without a maximum number of facets

How it works:

The executable opens a new console and the program asks you the parameters of the prism.
These parameters are the number of sides, the length of the side and the height of the prism. Then, ask if you want to rotate the prism in the X and the Y and the degrees.
This interface accepts decimal numbers in the range of the positive numbers.

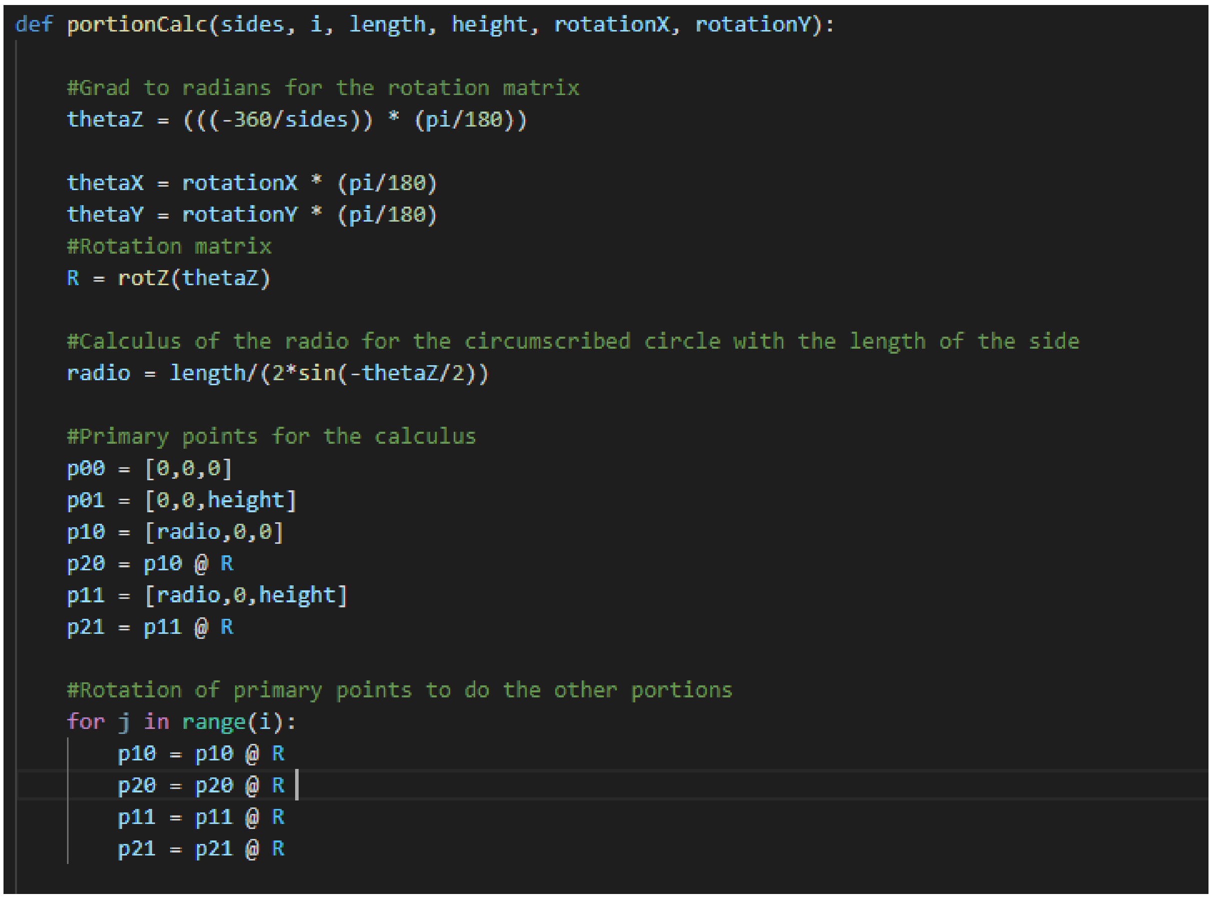


```python
def main():
    #Start menu
    print("Select number of sides (minimum 3): ")
    sides = int(input())
    while(sides < 3):
        print("Invalid number")
        print("Select number of sides (minimum 3): ")
        sides = int(input())
    print("Select the length of the side: ")
    length = float(input())
    while(length <= 0):
        print("Invalid number")
        print("Select the length of the side: ")
        length = float(input())

    print("Select the height of the prism: ")
    height = float(input())
    while(height <= 0):
        print("Invalid number")
        print("Select the height of the prism: ")
        height = float(input())
    print("Rotation on X (Deg 0-360): ")
    rotationX = float(input())
    while(rotationX < 0 or rotationX > 360):
        print("Rotation on X (Deg 0-360): ")
        rotationX = float(input())
    print("Rotation on Y (Deg 0-360): ")
    rotationY = float(input())
    while(rotationY < 0 or rotationY > 360):
        print("Rotation on X (Deg 0-360): ")
        rotationY = float(input())
```

The main program starts after selecting the parameters. The file created in the same directory of the execution of the program is being filled by the header of the STL file.

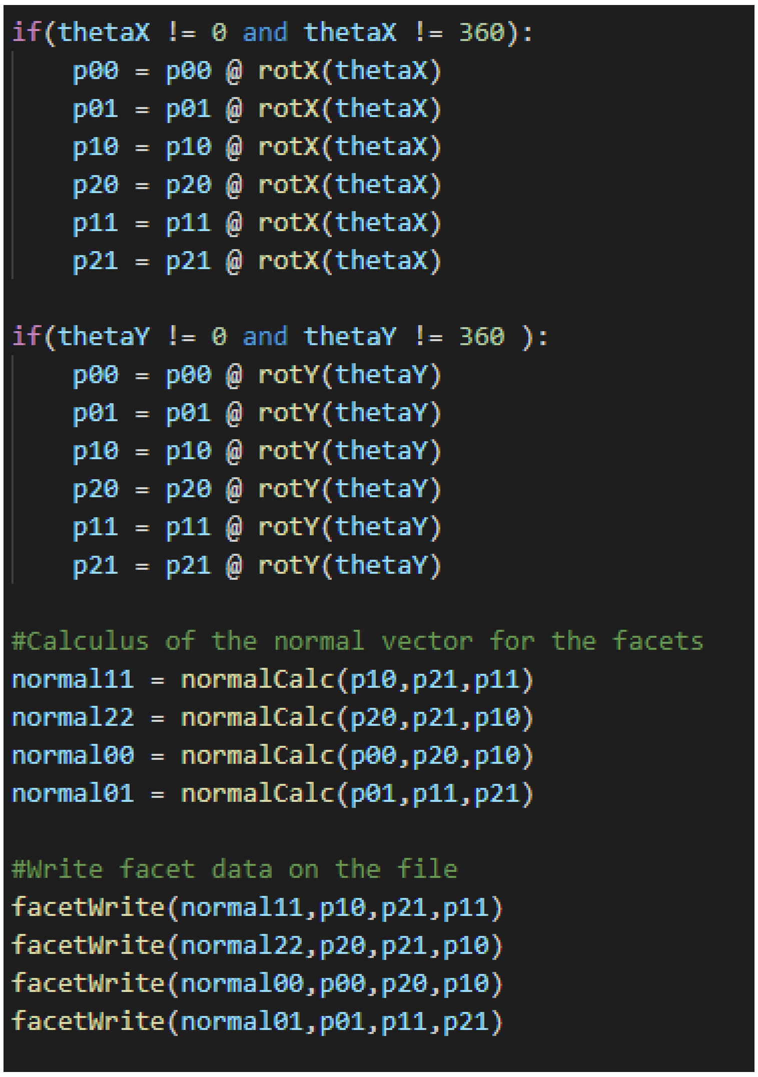
Then, a for loop with the same iterations of the sides of the polygon calls the function that does the calculations for creating the initial points and the normal vectors of the facets created.

A for loop is necessary for calculating all the portions of the prism. As an example, a cube consists of a 4 iterations with portions of 90deg, each portion rotated by the matrix 90deg.



```python
def portionCalc(sides, i, length, height, rotationX, rotationY):

    #Grad to radians for the rotation matrix
    thetaZ = (((-360/sides)) * (pi/180))

    thetaX = rotationX * (pi/180)
    thetaY = rotationY * (pi/180)
    #Rotation matrix
    R = rotZ(thetaZ)

    #Calculus of the radio for the circumscribed circle with the length of the side
    radio = length/(2*sin(-thetaZ/2))

    #Primary points for the calculus
    p00 = [0,0,0]
    p01 = [0,0,height]
    p10 = [radio,0,0]
    p20 = p10 @ R
    p11 = [radio,0,height]
    p21 = p11 @ R

    #Rotation of primary points to do the other portions
    for j in range(i):
        p10 = p10 @ R
        p20 = p20 @ R
        p11 = p11 @ R
        p21 = p21 @ R
```

In the same function, the rotation on the X and Y axis is made, and after that, the normal vectors are calculated and every facet created is created on the file calling the function “facetWrite()”.



```
if(thetaX != 0 and thetaX != 360):
    p00 = p00 @ rotX(thetaX)
    p01 = p01 @ rotX(thetaX)
    p10 = p10 @ rotX(thetaX)
    p20 = p20 @ rotX(thetaX)
    p11 = p11 @ rotX(thetaX)
    p21 = p21 @ rotX(thetaX)

if(thetaY != 0 and thetaY != 360 ):
    p00 = p00 @ rotY(thetaY)
    p01 = p01 @ rotY(thetaY)
    p10 = p10 @ rotY(thetaY)
    p20 = p20 @ rotY(thetaY)
    p11 = p11 @ rotY(thetaY)
    p21 = p21 @ rotY(thetaY)

#Calculus of the normal vector for the facets
normal11 = normalCalc(p10,p21,p11)
normal22 = normalCalc(p20,p21,p10)
normal00 = normalCalc(p00,p20,p10)
normal01 = normalCalc(p01,p11,p21)

#Write facet data on the file
facetWrite(normal11,p10,p21,p11)
facetWrite(normal22,p20,p21,p10)
facetWrite(normal00,p00,p20,p10)
facetWrite(normal01,p01,p11,p21)
```

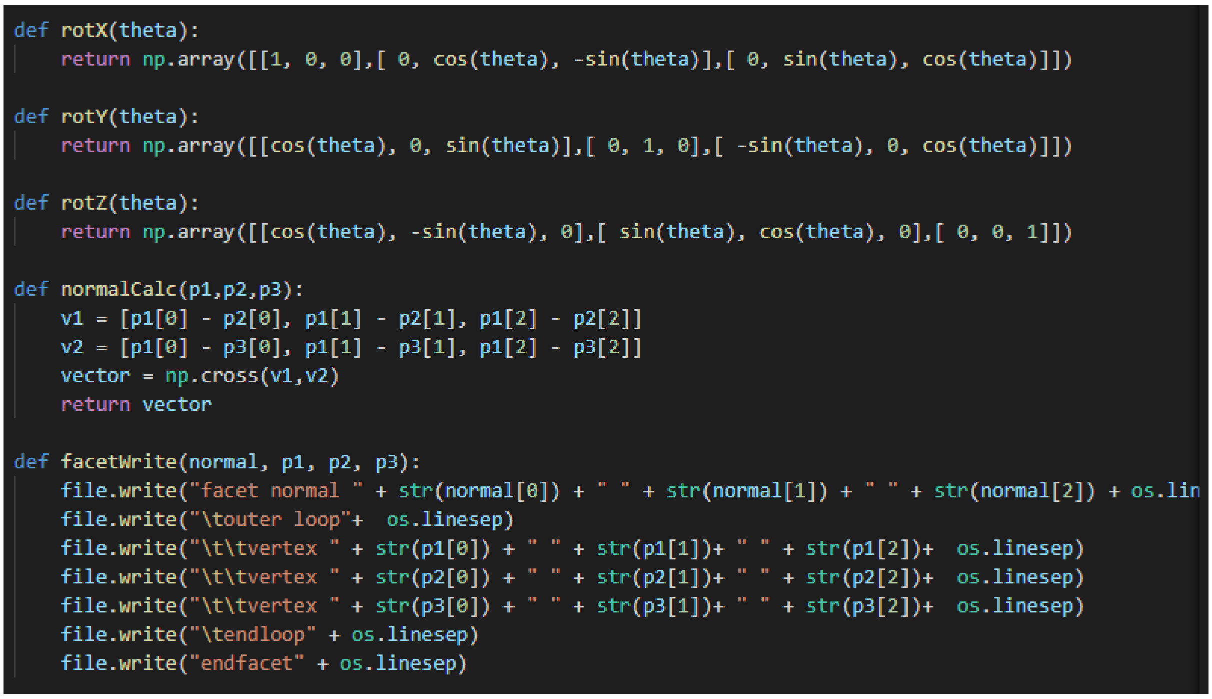
Functions used:

The functions that I created are rotX(), rotY(), rotZ(), normalCalc() and facetWrite() as they are needed for a clean code.

The 3 functions “rot” are the rotation matrix for each axis. They return the matrix that later can be multiplicated in each point.

The function normalCalc() consists in the calculus of the normal vector of 3 points.

And the most important function for writing the file is facetWrite, that writes the information of the normal vector and 3 points with the structure of an STL file.



```
def rotX(theta):
    return np.array([[1, 0, 0],[ 0, cos(theta), -sin(theta)],[ 0, sin(theta), cos(theta)]])

def rotY(theta):
    return np.array([[cos(theta), 0, sin(theta)],[ 0, 1, 0],[ -sin(theta), 0, cos(theta)]])

def rotZ(theta):
    return np.array([[cos(theta), -sin(theta), 0],[ sin(theta), cos(theta), 0],[ 0, 0, 1]])

def normalCalc(p1,p2,p3):
    v1 = [p1[0] - p2[0], p1[1] - p2[1], p1[2] - p2[2]]
    v2 = [p1[0] - p3[0], p1[1] - p3[1], p1[2] - p3[2]]
    vector = np.cross(v1,v2)
    return vector

def facetWrite(normal, p1, p2, p3):
    file.write("facet normal " + str(normal[0]) + " " + str(normal[1]) + " " + str(normal[2]) + os.lin
    file.write("\touter loop"+  os.linesep)
    file.write("\t\tvertex " + str(p1[0]) + " " + str(p1[1])+ " " + str(p1[2])+  os.linesep)
    file.write("\t\tvertex " + str(p2[0]) + " " + str(p2[1])+ " " + str(p2[2])+  os.linesep)
    file.write("\t\tvertex " + str(p3[0]) + " " + str(p3[1])+ " " + str(p3[2])+  os.linesep)
    file.write("\tendloop" + os.linesep)
    file.write("endfacet" + os.linesep)
```

Additional information:

This program is created for creating prisms, but if you put an elevated number of sides, essentially what you are creating is a cylinder.

If you execute the program multiple times, the file polygon.stl only saves the last prism that you created, it didn't merge.