Assignment 1: Laser Cut Container

MECE 4606 Digital Manufacturing

Nico Aldana (na2851)

Silvester Nava (sn2818)

Submitted: Feb 13, 2023 2:20 pm

Introductory Code

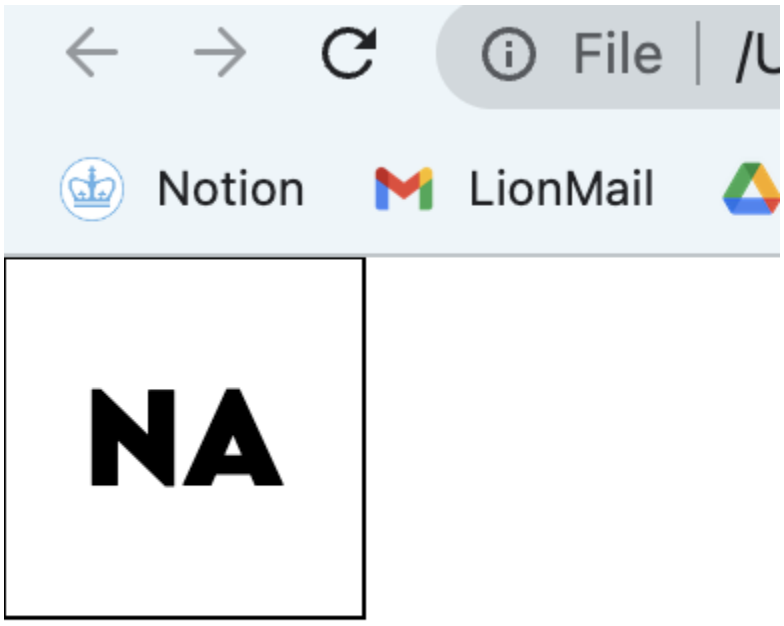
```
test.py U   square.py U   square.svg U   laser.py M
venv > laser-cut-container > square.py > ...
1  side = float(input("Please enter a side length for the square: "))
2  content = input("Please enter your initials: ")
3
4  output = '<?xml version="1.0" encoding="utf-8"?><svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
5
6  f = open('square.svg', 'w')
7  f.write(output)
8
9
```

```
<text x="%f" y="%f" dominant-baseline="middle" text-anchor="middle" class="st1 st2">%s</text></svg>' % (side, side, side/2, side/2, content)
```

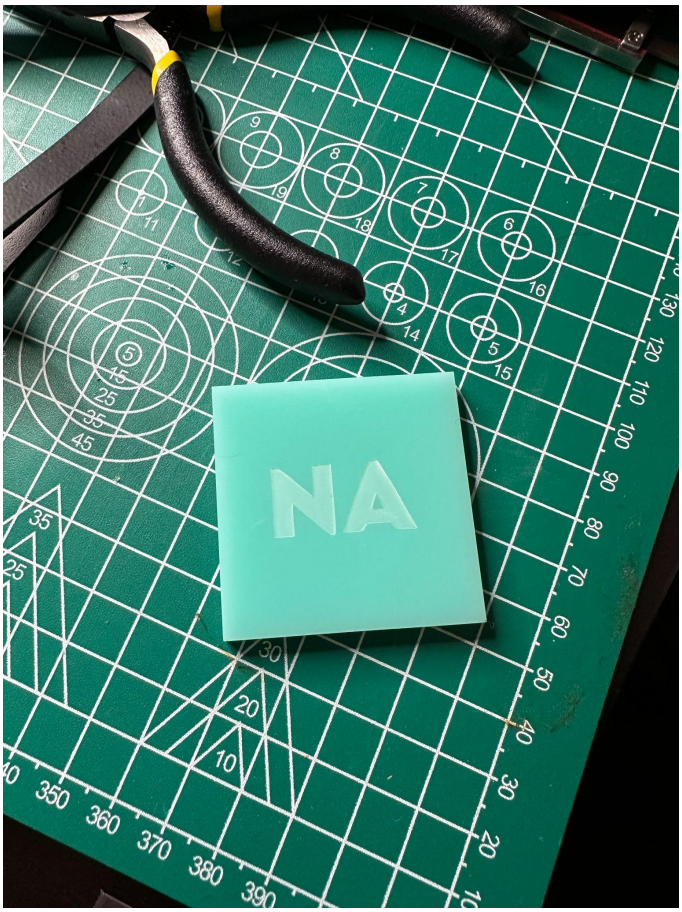
Input data is spliced into the output string.

Generating an SVG

```
(base) nicoaldana@Nicos-MacBook-Air laser-cut-container % python3 square.py
Please enter a side length for the square: 80
Please enter your initials: NA
(base) nicoaldana@Nicos-MacBook-Air laser-cut-container %
```

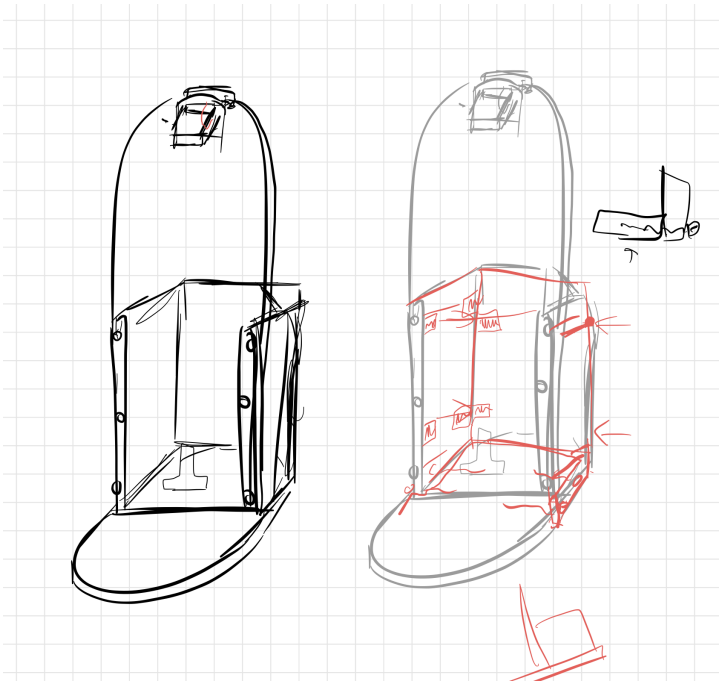


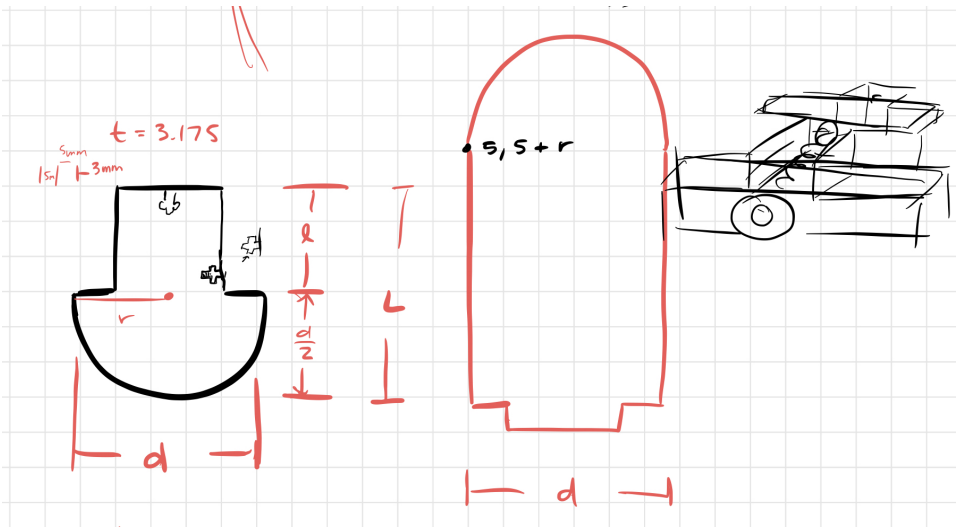
Photos of cut pieces:



Design

The desk organizer is intended to be multi-functional. One function is to hold pencils and other objects in a standard box, and the other function is to have a tall hook where a user could hang something, such as a hat or pair of headphones. Some preliminary concept sketches are shown below.





Software

All code was written in Python. The general approach was to have user-inputted variables inserted into strings that would be compiled into a master SVG document. The program would take five inputs: an overall length, overall depth, height of the pencil-holder component, text to be etched onto the base piece, and text to be etched onto the front of the pencil-holder. All other dimensions are derived from these quantities and constants (such as the thickness of the material).

```
print("Welcome to Nico & Silvester's Desk Organizer Generator! Please enter parameters. All measurements are in millimeters.")

# User inputted variables
d = check_input(90, 105, "Please input an overall depth for the container (Recommended: 90–105 mm): ")
L = check_input(125, 150, "Please input an overall length for the container (Recommended: 125–150 mm): ")
hp = check_input(100, 140, "Please input a height for the pencil holder (Recommended: 100–140 mm): ")

baseText = input("Please enter text you'd like engraved on the base of the organizer: ") # Determine appropriate font size for text length
frontText = input("Please enter text you'd like engraved on the front of the container: ")
```

The user input is run through a function to determine if the data are valid. If invalid, the user is prompted to re-enter a value.

```
# Validates data input
def check_input(lower, upper, prompt):
    while True:
        try:
            user_input = int(input(f"{prompt}"))
            if lower <= user_input <= upper:
                return user_input
            else:
                print("The number is not within the specified range.")
        except ValueError:
            print("Invalid input. Please enter a numerical value.")
```

Then, key dimensions are derived from the inputted data.

```
# Global variables
H = 300 # Overall height
t = 3.175 # Acrylic thickness
margin = 2 # Margin between pieces
svg_elements = [] # Array for fractal generation
```

Each component of the assembly is localized to its own function. This resulted in four main pieces: the base, the spine, the sides, and the lid. Other functions are used but are not considered main physical components.

```
# Create base piece paths
def Base():
    xOffset = 5 + d + t + margin
    baseOutput = f'<path d="M{xOffset} 5 l{dp} 0 l0 {l} l{t} 0 a{r} {r} 0 0 1 {-2*r} 0 l{t} 0 l{0} {-l}" stroke="black" stroke-width="1" fill="n
    baseOutput += f'<rect x="{xOffset+dp/3}" y="{5+l}" width="{dp/3}" height="{t}" stroke="black" stroke-width="1" fill="none"/>'

    # User-specified text
    baseOutput += '<style type="text/css">.st0{fill:#FFFFFF;stroke:#000000;stroke-miterlimit:10;}.st1{font-family:\'Impact\';}.st4{font-size:'
    baseOutput += f'{baseFontSize}px;'
    baseOutput += '}</style>'
    baseOutput += f'<text x="{xOffset - t + r}" y="{l+r/2}" dominant-baseline="middle" text-anchor="middle" class="st1 st4">{baseText}</text>'

    #Top Screwhole
    baseOutput += f'<path d="M{xOffset + dp/2} 5 h -1.1 v 3 h -1.3 v 1.6 h 1.3 v 1.8 h 2.2 v -1.8 h 1.5 v -1.6 h -1.5 v -3 h -1.1" stroke="black
    #Left Screwholes
    baseOutput += f'<path d="M{xOffset} {5 + l/4} v -1.1 h 3 v -1.3 h 1.6 v 1.3 h 1.8 v 2.2 h -1.8 v 1.5 h -1.6 v -1.5 h -3 v -1.1" stroke="blac
    baseOutput += f'<path d="M{xOffset} {5 + 3*l/4} v -1.1 h 3 v -1.3 h 1.6 v 1.3 h 1.8 v 2.2 h -1.8 v 1.5 h -1.6 v -1.5 h -3 v -1.1" stroke="bl
    #Right Screwholes
    baseOutput += f'<path d="M{xOffset + dp} {5 + l/4} v -1.1 h -3 v -1.3 h -1.6 v 1.3 h -1.8 v 2.2 h 1.8 v 1.5 h 1.6 v -1.5 h 3 v -1.1" stroke=
    baseOutput += f'<path d="M{xOffset + dp} {5 + 3*l/4} v -1.1 h -3 v -1.3 h -1.6 v 1.3 h -1.8 v 2.2 h 1.8 v 1.5 h 1.6 v -1.5 h 3 v -1.1" strok

    return baseOutput
```

```
# Create spine piece paths
def Spine():
    # Derived measurements:
    rectX = 5 + (d-20)/2
    rectY = 5 + r/3
    tabX = 5 + d + margin
    tabY = 5 + H

    spineOutput = f'<path d="M5 {5+r} a{r} {r} 0 0 1 {2*r} 0 l0 {h} l{-(d-dp/3)/2} 0 l0 {t} l{-dp/3} 0 l0 {-t} l{-(d-dp/3)/2} 0 l0 {-h}" stroke=
    spineOutput += f'<rect x="{rectX}" y="{rectY}" width="20" height="{t}" stroke="black" stroke-width="1" fill="none"/>'

    # Circle Screwholes
    spineOutput += f'<circle cx="{5+2.4}" cy="{5 + H + t - hp/5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'
    spineOutput += f'<circle cx="{5+2.4}" cy="{5 + H + t - hp/5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'
    spineOutput += f'<circle cx="{2.4+d}" cy="{5 + H + t - 4*hp/5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'
    spineOutput += f'<circle cx="{2.4+d}" cy="{5 + H + t - 4*hp/5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'

    # Circle Screwholes adjusted
    spineOutput += f'<circle cx="{5+2.4}" cy="{H-0.8*hp+3.2+5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'
    spineOutput += f'<circle cx="{5+2.4}" cy="{H-0.2*hp+3.2+5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'
    spineOutput += f'<circle cx="{2.4+d}" cy="{H-0.8*hp+3.2+5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'
    spineOutput += f'<circle cx="{2.4+d}" cy="{H-0.2*hp+3.2+5}" r="1.2" stroke="black" stroke-width="1" fill="none"/>'

    # Generate tab cutout
    spineOutput += f'<path d="M{tabX} {tabY} v -30 h 5 v 5 h 40 v 20 h -40 v 5 h -5" stroke="black" stroke-width="1" fill="none" />'

    # Generate fractal pattern
    spineOutput += sierpinski_triangle(6, 5 + r, -d-r)
```

```
# Generate side piece paths
def Sides():
    yOffset = 5 + H + t + margin

    # Rectangles
    sideOutput = f'<rect x="5" y="{5 + yOffset}" width="{l}" height="{hp}" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<rect x="{5 + l}" y="{5 + yOffset}" width="{l}" height="{hp}" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<rect x="{5 + 2*l}" y="{5 + yOffset}" width="{d}" height="{hp}" stroke="black" stroke-width="1" fill="none" />' # Front pane

    # Tab cutouts
    sideOutput += f'<path d="M{5 + l/3} {5 + yOffset} l0 {t} l{l/3} 0 l0 {-t}" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<path d="M{5 + 4*l/3} {5 + yOffset} l0 {t} l{l/3} 0 l0 {-t}" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<path d="M{5 + 2*l + (d-w/3)/2} {5 + yOffset} l0 {t} l{w/3} 0 l0 {-t}" stroke="black" stroke-width="1" fill="none" />'

    sideOutput += '<style type="text/css">.st0{fill:#FFFFFF;stroke:#000000;stroke-miterlimit:10;}.st1{font-family:\'Impact\';}.st2{'
    sideOutput += f'font-size:{frontFontSize}px;'
    sideOutput += '}.st3{font-size:8px;}</style>'
    sideOutput += f'<text x="{5 + 2*l + d/2}" y="{hp/4 + yOffset}" dominant-baseline="middle" text-anchor="middle" class="st1 st2">{frontText}</

    # Side piece bottom screw holes
    sideOutput += f'<circle cx="{5 + l/4}" cy="{yOffset + hp + 2.4}" r="1.2" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<circle cx="{5 + 3*l/4}" cy="{yOffset + hp + 2.4}" r="1.2" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<circle cx="{5 + l + l/4}" cy="{yOffset + hp+2.4}" r="1.2" stroke="black" stroke-width="1" fill="none" />'
    sideOutput += f'<circle cx="{5 + l + 3*l/4}" cy="{yOffset + hp+2.4}" r="1.2" stroke="black" stroke-width="1" fill="none" />'

    # Facing right screwholes
    sideOutput += f'<path d="M5 {yOffset + 5+ hp/5} v -1.1 h 3 v -1.3 h 1.6 v 1.3 h 1.8 v 2.2 h -1.8 v 1.5 h -1.6 v -1.5 h -3 v -1.1" stroke="bl
    sideOutput += f'<path d="M5 {yOffset + 5 + 4*hp/5} v -1.1 h 3 v -1.3 h 1.6 v 1.3 h 1.8 v 2.2 h -1.8 v 1.5 h -1.6 v -1.5 h -3 v -1.1" stroke=
    sideOutput += f'<path d="M{5 + l} {yOffset + 5 + hp/5} v -1.1 h 3 v -1.3 h 1.6 v 1.3 h 1.8 v 2.2 h -1.8 v 1.5 h -1.6 v -1.5 h -3 v -1.1" str
    sideOutput += f'<path d="M{5 + l} {yOffset + 5 + 4*hp/5} v -1.1 h 3 v -1.3 h 1.6 v 1.3 h 1.8 v 2.2 h -1.8 v 1.5 h -1.6 v -1.5 h -3 v -1.1" s
```

```
def Lid():
    # Calculate offset here
    xOffset = 5 + d + t + margin
    yOffset = 5 + L + margin
    w = d - 2*t
    lidOutput = f'<path d="M{xOffset} {yOffset} l{w} 0 l0 {l/3} l{t} 0 l0 {l/3} l{-t} 0 l0 {l/3} l{-w/3} 0 l0 {t} l{-w/3} 0 l0 {-t} l{-w/3} 0
    lidOutput += f'<circle cx="{xOffset + w/4}" cy="{yOffset + l/4}" r="{w/9}" stroke="black" stroke-width="1" fill="none"/>'
    lidOutput += f'<circle cx="{xOffset + w/2}" cy="{yOffset + l/4}" r="{w/9}" stroke="black" stroke-width="1" fill="none"/>'
    lidOutput += f'<circle cx="{xOffset + 3*w/4}" cy="{yOffset + l/4}" r="{w/9}" stroke="black" stroke-width="1" fill="none"/>'
    lidOutput += f'<rect x="{xOffset + w/4 - w/9}" y="{yOffset + l/2}" width="{w/2 + 2*w/9}" height="{l/3}" rx="3" ry="3" stroke="black" stro
    return lidOutput
```

The output of each function is a string that is repeatedly concatenated. At the very end of the program, each function is called and the strings are compiled into one master SVG.

```
# Compile all
def compile():
    final = starter
    final += Spine()
    final += Base()
    final += Sides()
    final += Lid()
    final += "</svg>"
    f = open('final.svg','w')
    f.write(final)

compile()
```

For the fractal pattern, a recursive function is used. A randomly generated integer determines the depth of the pattern, which is then created and centered on the spine piece. The fractal stays within the confines of the spine dimensions.

```python
# Generate Sierpinski triangle fractal
def sierpinski_triangle(n, x, y):
    svg_elements = []
    def recursive_triangle(points, depth):
        if depth == 0:
            x1, y1 = points[0]
            x2, y2 = points[1]
            x3, y3 = points[2]
            svg_elements.append(f'<polygon points="{x1},{-y1} {x2},{-y2} {x3},{-y3}" stroke-width="1" fill="black"/>')
        else:
            x1, y1 = points[0]
            x2, y2 = points[1]
            x3, y3 = points[2]
            x12 = (x1 + x2) / 2
            y12 = (y1 + y2) / 2
            x23 = (x2 + x3) / 2
            y23 = (y2 + y3) / 2
            x31 = (x3 + x1) / 2
            y31 = (y3 + y1) / 2
            recursive_triangle([(x1, y1), (x12, y12), (x31, y31)], depth - 1)
            recursive_triangle([(x12, y12), (x2, y2), (x23, y23)], depth - 1)
            recursive_triangle([(x31, y31), (x23, y23), (x3, y3)], depth - 1)

    size = 2**n
    x1 = x
    y1 = y
    x2 = x + size
    y2 = y
    x3 = x + size / 2
    y3 = y + (3**0.5) / 2 * size
```

```python
    size = 2**n
    x1 = x
    y1 = y
    x2 = x + size
    y2 = y
    x3 = x + size / 2
    y3 = y + (3**0.5) / 2 * size
    x_offset = -size / 2
    y_offset = -y3 / 2
    recursive_triangle([(x1 + x_offset, y1 + y_offset), (x2 + x_offset, y2 + y_offset), (x3 + x_offset, y3 + y_offset)], n)

    return '\n'.join(svg_elements)
```

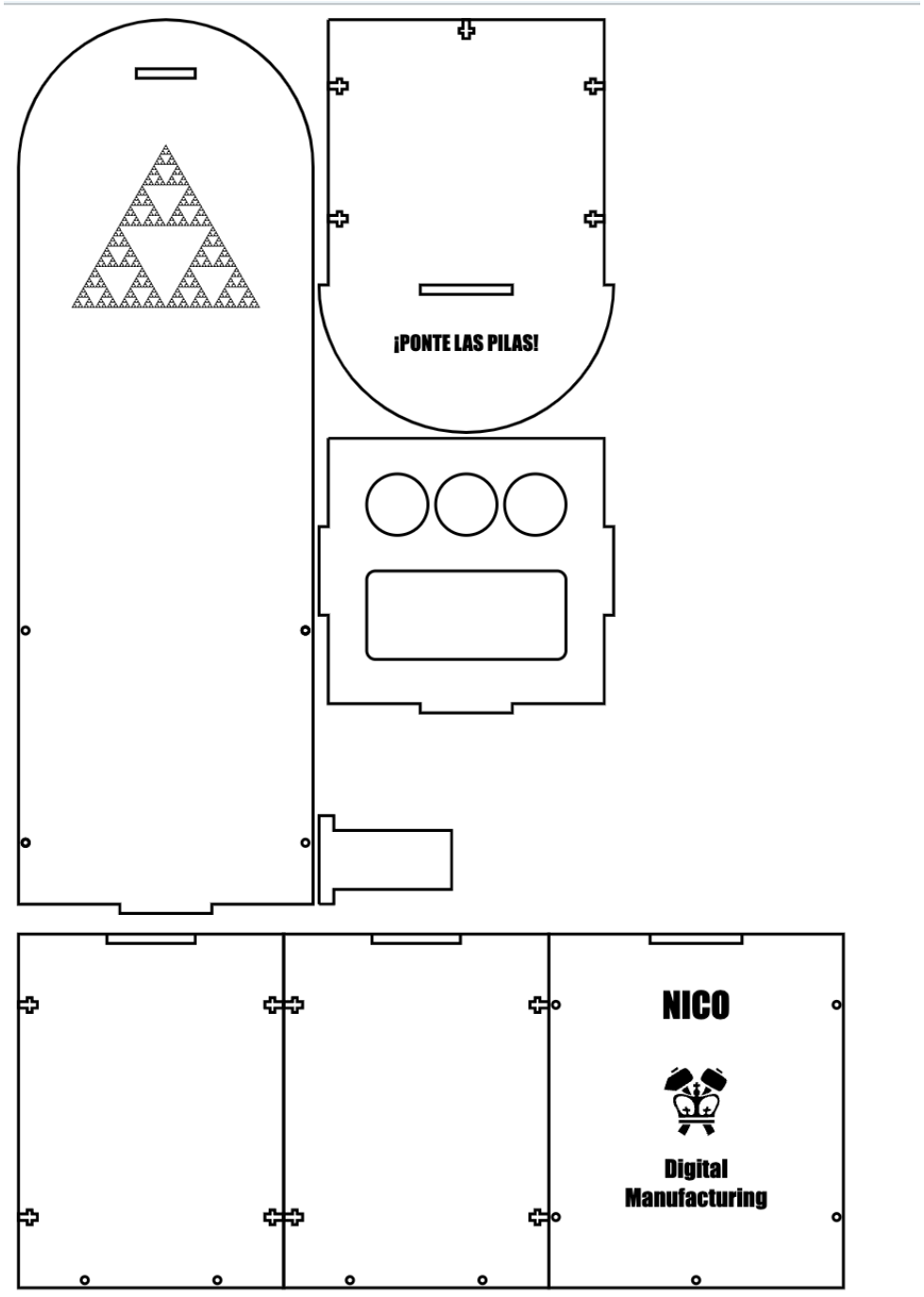
The Columbia SEAS logo is localized to its own function due to the complexity of its SVG path.

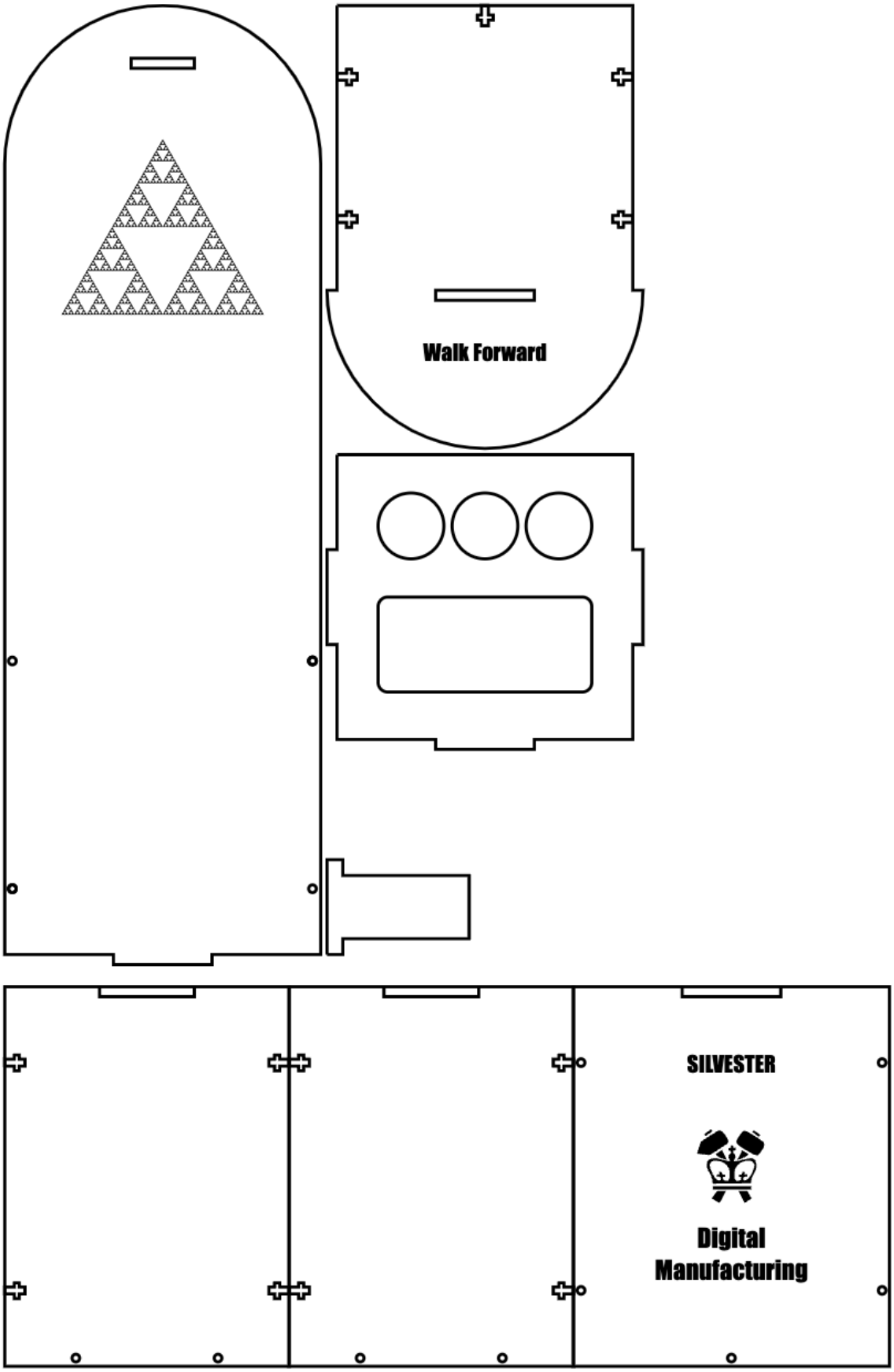
```python
# Generate CU SEAS logo as svg path
def seas_logo():
    output = '<g transform="translate(0.000000,870.000000) scale(0.100000,-0.100000)">'
    output += '<path d="M139.2,8666.2c-23.5-16-42.7-29.7-42.4-30.3c0.2-0.8,4-6.5,8.5-12.7l8.1-11.4l42.3,28.6,c23.2,15.8,42.7,29,43.2,29.6c0.7,
    output += '<path d="M715.4,8673.5c-3.9-6.2-7.2-11.9-7.3-12.8c-0.1-1.1,80-57.5,81.7-57.5c0.6,0,15.9,22.7,15.5,23,c-0.2,0.2-17.7,12.6-38.8,
    output += '<path d="M286.9,8682c-6.4-1.1-17.2-5.3-27.1-10.4c-15.8-8.4-67.7-40.6-169.4-105.4l-35.1-22.5L29.5,8466,c-15-45.4-25.1-77.8-24.5-
    output += '<path d="M599.9,8676.3c-8.5-1.5-12.2-3.9-23.1-14.6c-24-24.1-45.7-54.7-63.1-89.3c-8.2-16.1-10.3-25.4-9.4-40.7,c0.7-13.1,1.8-14.7
    output += '<path d="M432.7,8476.9v-14.4l-15.1-0.3l-15.2-0.3l-0.3-17.4l-0.2-17.4l15.1-0.3l15.2-0.3l0.3-15.3l0.3-15.3l-5.6-3.7 c-7.3-4.7-16.
    output += '<path d="M299,8421.4c-13.4-9-31.9-21.4-41.2-27.5l-16.8-11.1l2.4-4.5c3.7-6.8,15.1-18,25.4-24.8 c22.3-14.8,59.3-28.9,99.7-37.6l11
    output += '<path d="M567.6,8422.5c-3.9-16.1-9-29.7-18.3-48.8c-7.8-16-21.9-38.8-32.3-52.1c-4.5-5.8-5.1-7.1-3.5-7.3 c3.1-0.5,35.5,6.9,51,11.
    output += '<path d="M204.9,7956.5v-21h242.2h242.2v21v21H447.1H204.9V7956.5z"/>'
    output += '<path d="M256.8,7913.8c-1.5-1.9-16.8-22.7-34.1-46.1l-31.4-42.6l47.4-30.5c26.1-16.8,47.7-30.6,48-30.9 c1.5-0.5,14.1,21,30.1,51.9
    output += '<path d="M527.5,7916.7c0-0.3,1.2-2.4,2.6-4.6c4.9-7.8,31.2-59.7,52.8-103.7l21.9-44.9l45.7,29.8 c25.2,16.3,45.9,29.8,46.1,30c0.1,
    output += '</g>'
    return output
```

An example console interaction is shown below:

```
(laser-cut-container) (base) nicoaldana@Nicos-MacBook-Air laser-cut-container % python master.py
Welcome to Nico & Silvester's Desk Organizer Generator! Please enter parameters. All measurements are in millimeters.
Please input an overall depth for the container (Recommended: 80-100 mm): 100
Please input an overall length for the container (Recommended: 120-150 mm): 140
Please input a height for the pencil holder (Recommended: 100-140 mm): 120
Please enter text you'd like engraved on the base of the organizer: ¡PONTE LAS PILAS!
Please enter text you'd like engraved on the front of the container: NICO
(laser-cut-container) (base) nicoaldana@Nicos-MacBook-Air laser-cut-container %
```

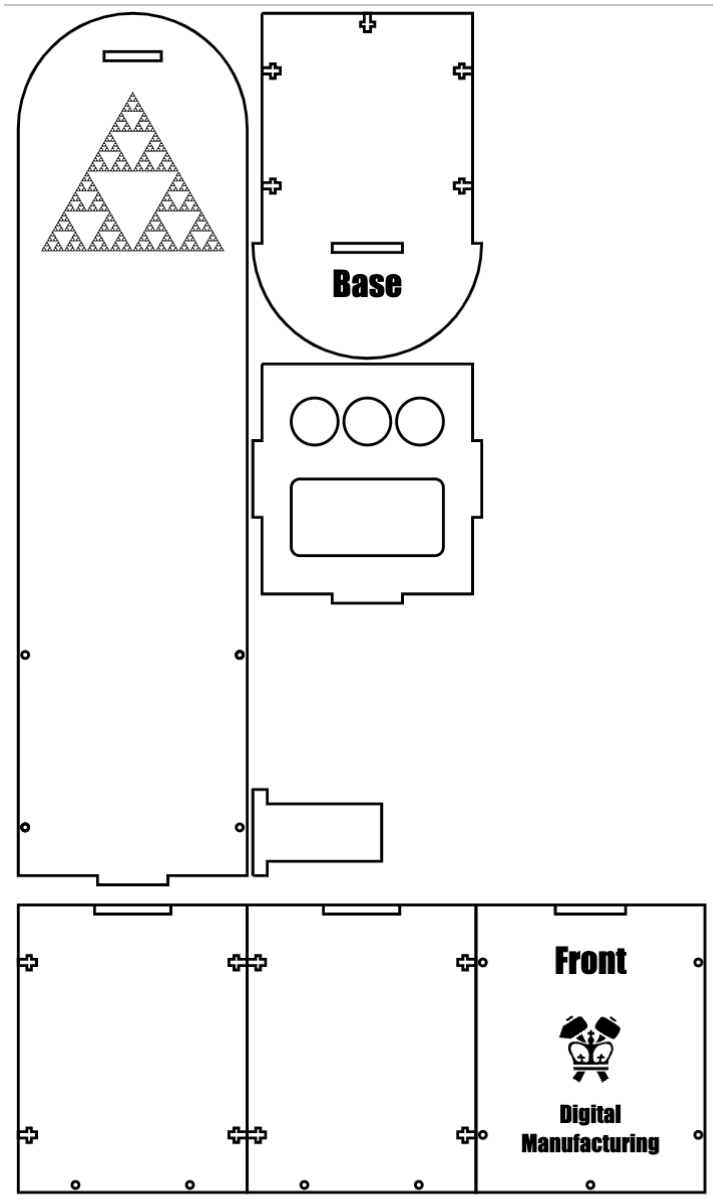
And here is the generated SVG:



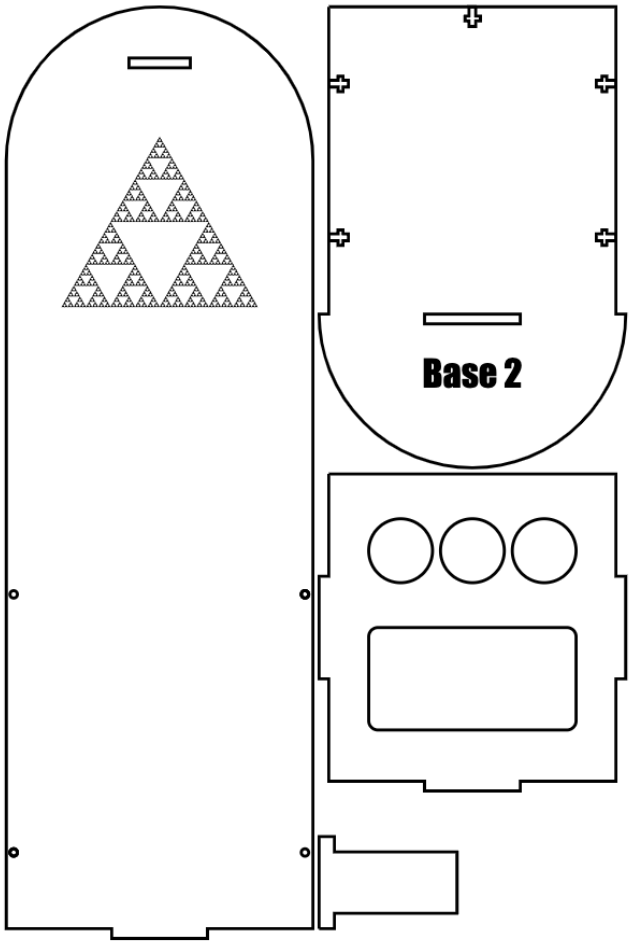

Walk Forward
SILVESTER
Digital
Manufacturing

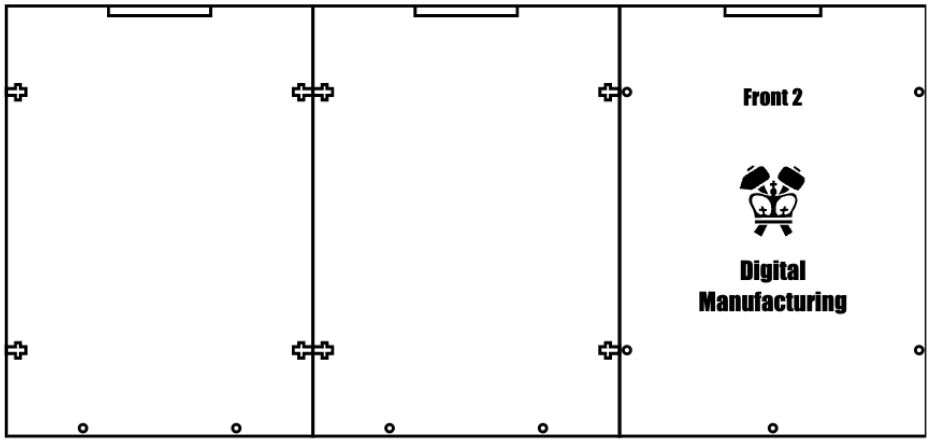
Here are two examples with different parameters: (Minimum and maximum dimensions)

```
(laser-cut-container) (base) nicoaldana@Nicos-MacBook-Air laser-cut-container % python master.py
Welcome to Nico & Silvester's Desk Organizer Generator! Please enter parameters. All measurements are in millimeters.
Please input an overall depth for the container (Recommended: 80-100 mm): 80
Please input an overall length for the container (Recommended: 120-150 mm): 10
The number is not within the specified range.
Please input an overall length for the container (Recommended: 120-150 mm): 120
Please input a height for the pencil holder (Recommended: 100-140 mm): 100
Please enter text you'd like engraved on the base of the organizer: Base
Please enter text you'd like engraved on the front of the container: Front
(laser-cut-container) (base) nicoaldana@Nicos-MacBook-Air laser-cut-container %
```



```
Welcome to Nico & Silvester's Desk Organizer Generator! Please enter parameters. All measurements are in millimeters.
Please input an overall depth for the container (Recommended: 80-100 mm): 100
Please input an overall length for the container (Recommended: 120-150 mm): 150
Please input a height for the pencil holder (Recommended: 100-140 mm): 140
Please enter text you'd like engraved on the base of the organizer: Base 2
Please enter text you'd like engraved on the front of the container: Front 2
```





Physical Results


SILVESTER
Digital
Manufacturing
NICO
Digital
Manufacturing


SONY


SONY