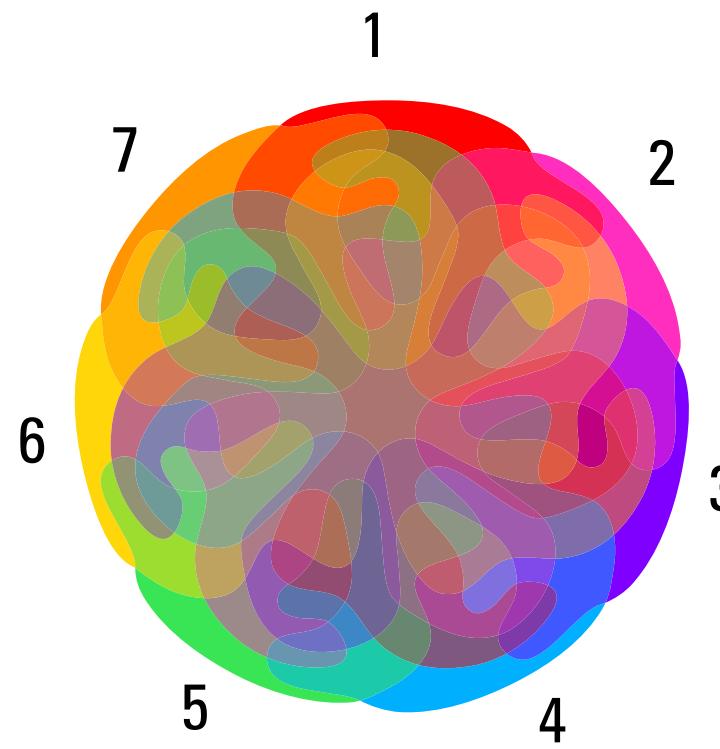


August 22, 2023

Automating logo revisions with ChatGPT

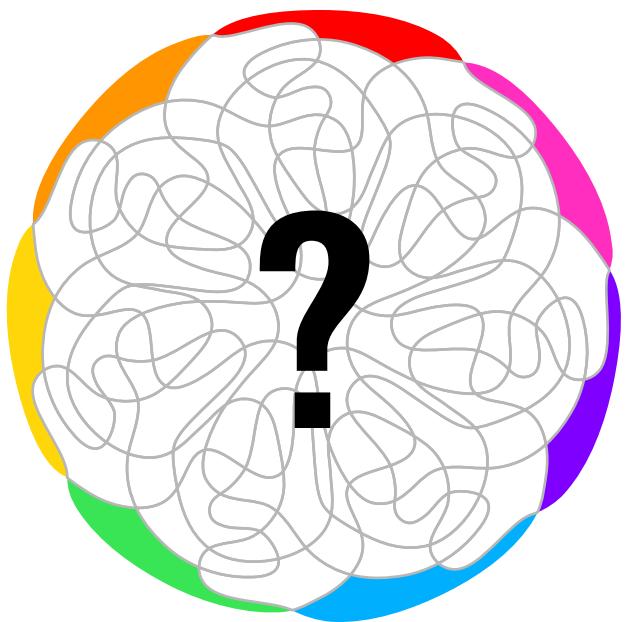
The DrumWave logo is a Venn diagram of 7 sets.

Each of the 7 sets is assigned a base color, and the remaining 120 regions are colored an average of the overlapping sets.



The problem:

When tweaking the 7 base colors,
120 averages have to be calculated
in order to produce a new version of the logo.



This could be done by hand —
but it is time consuming and there is a high probability of error.



This could be done with a program — but I don't know how to code, and the office developer is busy.

Search

```
hexcode_replace.py
Users > jacobsheiner > Downloads > dw_logo > hexcode_replace.py ...
1  import re
2  from PIL import Image
3
4  def extract_all_hexcodes(svg_content):
5      # Regular expression pattern to find hexcodes in fill and stroke attributes
6      pattern = r'(#(?:[A-Fa-f0-9]{6}|[A-Fa-f0-9]{3}))'
7      all_hexcodes = re.findall(pattern, svg_content)
8      return all_hexcodes
9
10 def calculate_average_color(combination, base_colors):
11     total_r, total_g, total_b = 0, 0, 0
12     num_sets = sum(combination)
13
14     for i, present in enumerate(combination):
15         if present:
16             color = base_colors[i]
17             r, g, b = int(color[1:3], 16), int(color[3:5], 16), int(color[5:7], 16)
18             total_r += r
19             total_g += g
20             total_b += b
21
22     average_r = total_r // num_sets
23     average_g = total_g // num_sets
24     average_b = total_b // num_sets
25
26     average_color = f'#{average_r:02X}{average_g:02X}{average_b:02X}'
27     return average_color
28
29 def replace_all_hexcodes(svg_content, new_hexcodes):
30     # Regular expression pattern to find hexcodes in fill and stroke attributes
31     pattern = r'(#(?:[A-Fa-f0-9]{6}|[A-Fa-f0-9]{3}))'
32     def replace_func(match):
33         return new_hexcodes.pop(0)
34
35     # Replace hexcodes using re.sub and the custom replace_func
36     new_svg_content = re.sub(pattern, replace_func, svg_content)
37     return new_svg_content
38
39 def main():
40     input_svg_file = "dw_logo_slides.svg" # Replace with your input SVG file path
```

□ □ □ □

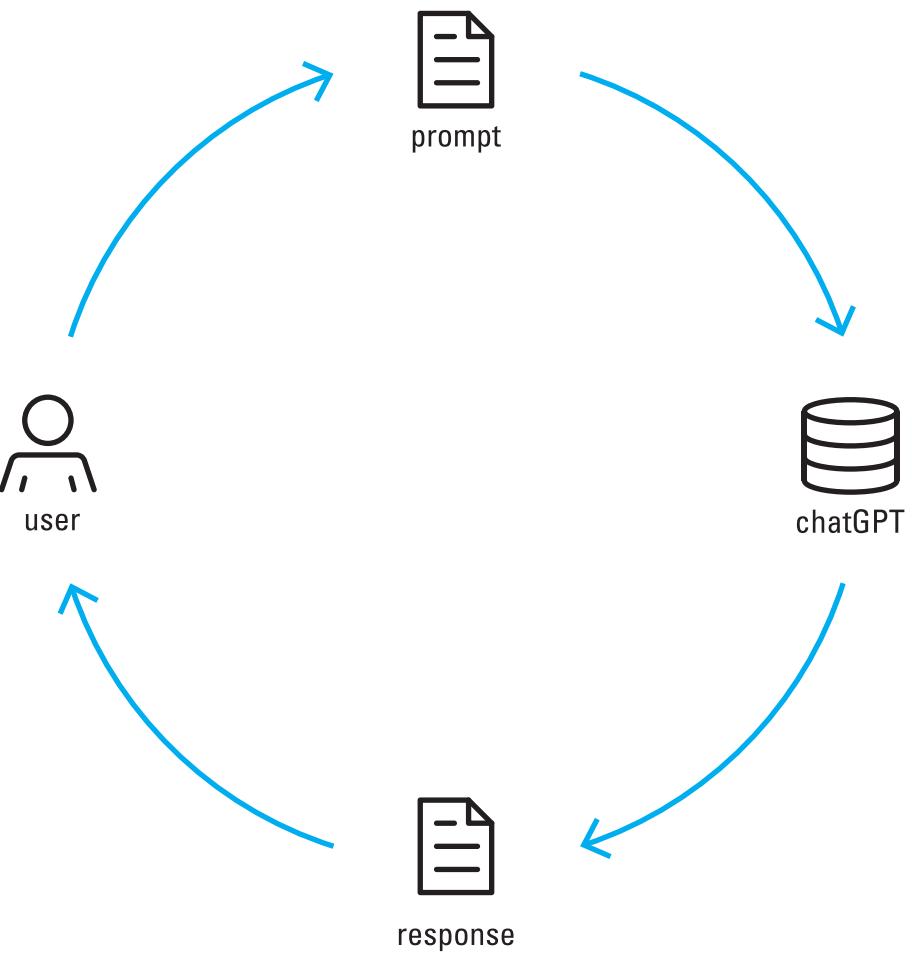
▷ ▽ □ ...

```
hexcode_replace.py
Users > jacobsheiner > Downloads > dw_logo > hexcode_replace.py ...
1  import re
2  from PIL import Image
3
4  def extract_all_hexcodes(svg_content):
5      # Regular expression pattern to find hexcodes in fill and stroke attributes
6      pattern = r'(#(?:[A-Fa-f0-9]{6}|[A-Fa-f0-9]{3}))'
7      all_hexcodes = re.findall(pattern, svg_content)
8      return all_hexcodes
9
10 def calculate_average_color(combination, base_colors):
11     total_r, total_g, total_b = 0, 0, 0
12     num_sets = sum(combination)
13
14     for i, present in enumerate(combination):
15         if present:
16             color = base_colors[i]
17             r, g, b = int(color[1:3], 16), int(color[3:5], 16), int(color[5:7], 16)
18             total_r += r
19             total_g += g
20             total_b += b
21
22     average_r = total_r // num_sets
23     average_g = total_g // num_sets
24     average_b = total_b // num_sets
25
26     average_color = f'#{average_r:02X}{average_g:02X}{average_b:02X}'
27     return average_color
28
29 def replace_all_hexcodes(svg_content, new_hexcodes):
30     # Regular expression pattern to find hexcodes in fill and stroke attributes
31     pattern = r'(#(?:[A-Fa-f0-9]{6}|[A-Fa-f0-9]{3}))'
32     def replace_func(match):
33         return new_hexcodes.pop(0)
34
35     # Replace hexcodes using re.sub and the custom replace_func
36     new_svg_content = re.sub(pattern, replace_func, svg_content)
37     return new_svg_content
38
39 def main():
40     input_svg_file = "dw_logo_slides.svg" # Replace with your input SVG file path
```

Can we leverage AI to write a program for us?

Using ChatGPT is like having a conversation.

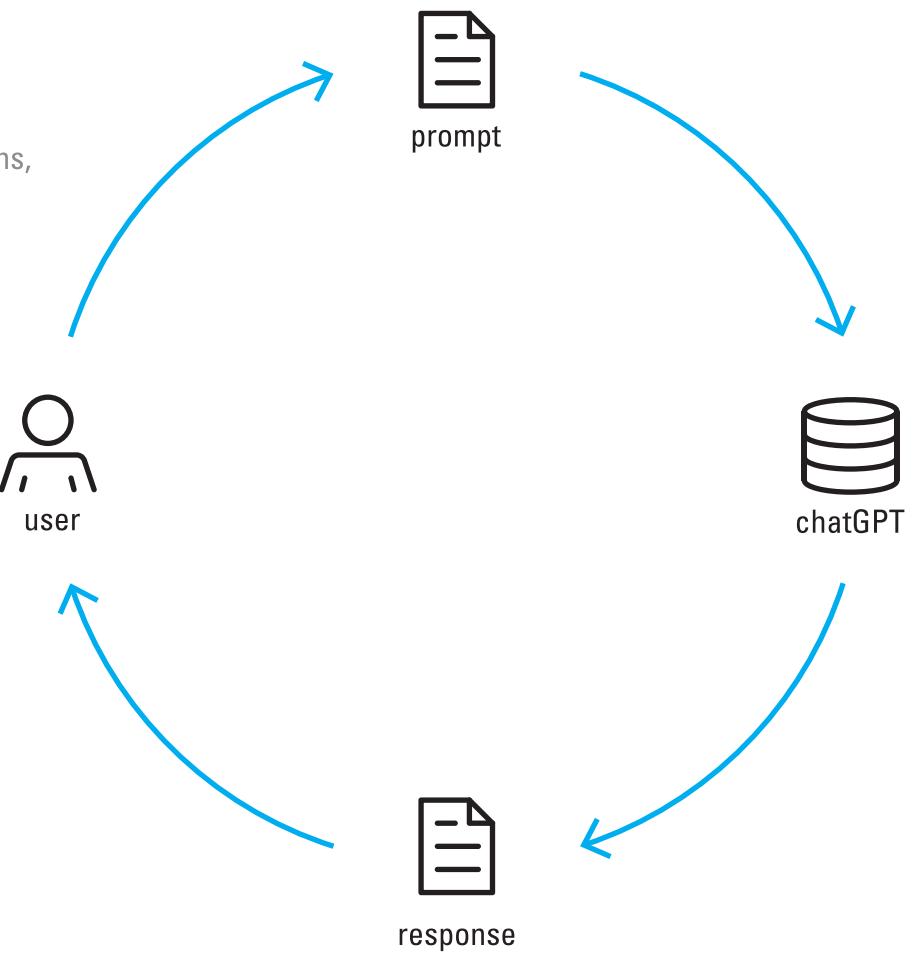
Each thread is a cycle of prompts from the user,
and responses from the model.



First I ask it how to solve our problem.

Prompt:

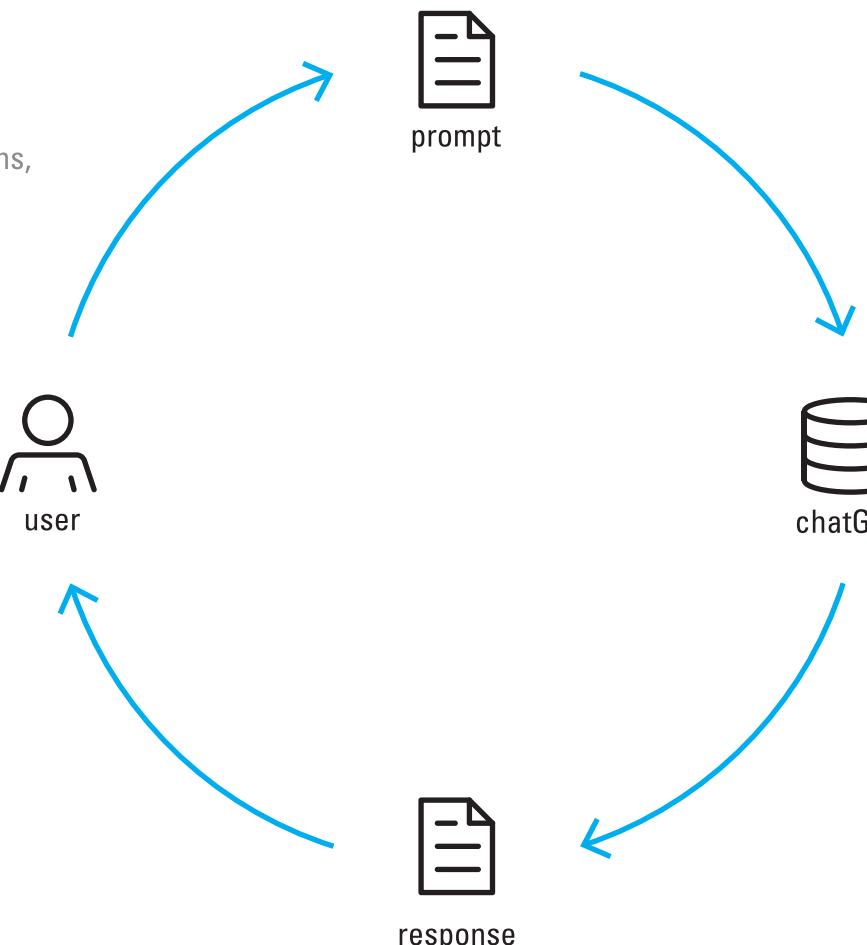
I am making a picture of a seven set venn diagram.
Every section overlaps every other section in all possible combinations,
resulting in 128 individual sections.
If each of the seven sets is a specific color (hex value),
and every section is the average color of the sets that overlap in it,
how can I calculate the colors for all 128 sections?



GPT responds.

Prompt:

I am making a picture of a seven set venn diagram.
Every section overlaps every other section in all possible combinations,
resulting in 128 individual sections.
If each of the seven sets is a specific color (hex value),
and every section is the average color of the sets that overlap in it,
how can I calculate the colors for all 128 sections?



Response

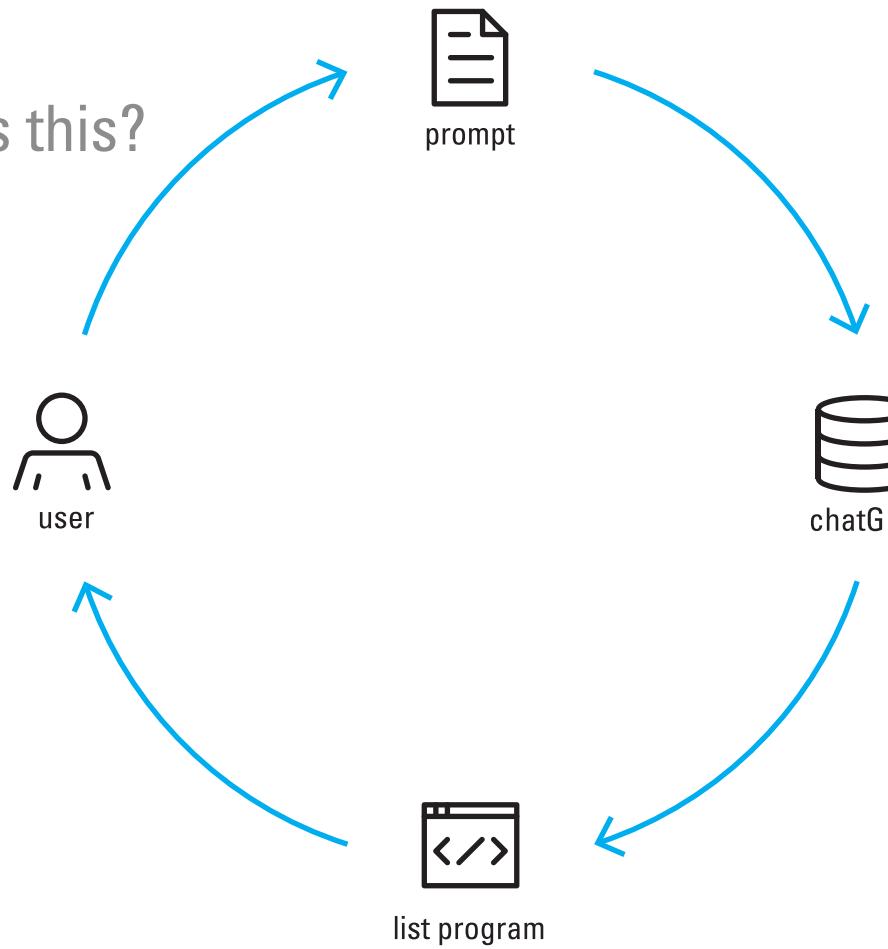
To calculate the average color for each of the 128 sections in a seven-set Venn diagram, where each section represents the overlapping sets, you can follow these steps:

1. Assign a specific color (hex value) to each of the seven sets in your diagram. Let's denote the colors as C1, C2, C3, C4, C5, C6, and C7.
2. Enumerate all the possible combinations of overlapping sets. In this case, since there are seven sets, there will be $2^7 = 128$ combinations.
3. For each combination, determine which sets overlap. Represent the presence or absence of a particular set in the combination using binary digits, where 1 represents the presence of the set and 0 represents its absence. For example, if the combination is {C1, C3, C4, C7}, it can be represented as 1100101, where the first digit corresponds to C1, the second to C2, and so on.

**This confirms that the model understands what we are asking for.
The next step is to ask it for a program that solves it for us.**

Prompt:

can you write a program that does this?



Response:

```
from itertools import combinations

# Hex color values for each set
colors = {
    'C1': '#FF0000',
    'C2': '#00FF00',
    'C3': '#0000FF',
    'C4': '#FFFF00',
    'C5': '#FF00FF',
    'C6': '#00FFFFFF',
    'C7': '#FFFFFF'
}

# Function to calculate average color
def calculate_average_color(combination):
    total_r, total_g, total_b = 0, 0, 0
    num_sets = sum(combination)

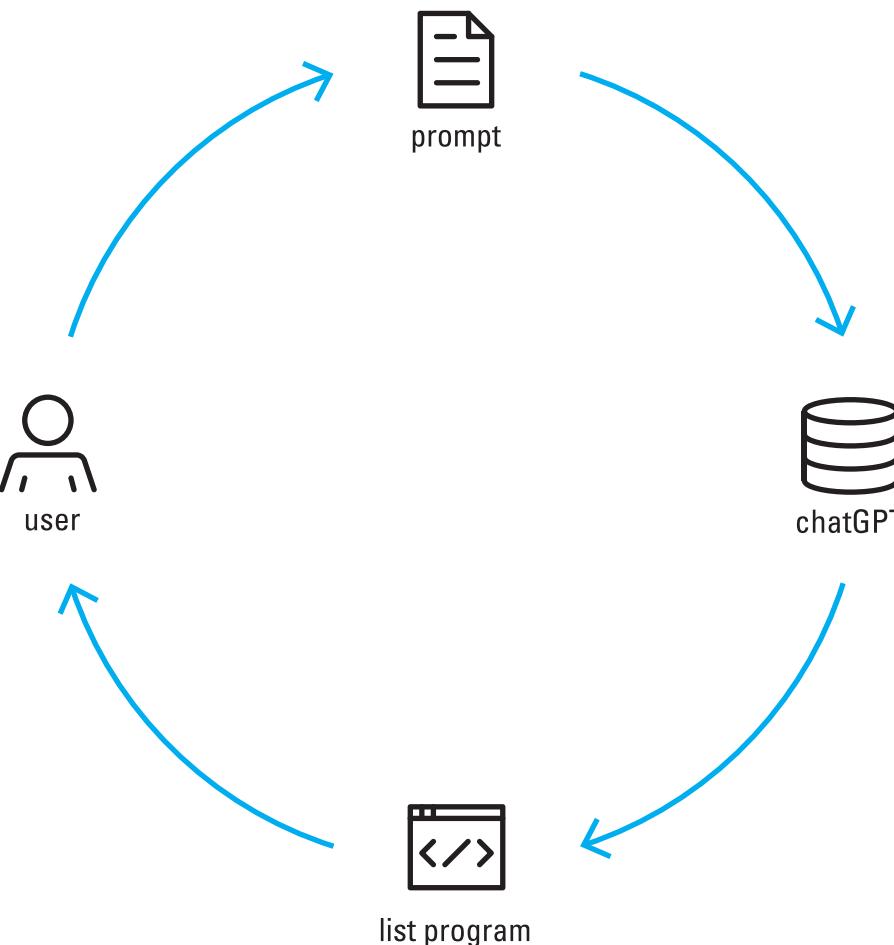
    for i, present in enumerate(combination):
        if present:
            color = colors[f'C{i+1}']
            r, g, b = int(color[1:3], 16), int(color[3:5], 16), int(color[5:7], 16)
            total_r += r
            total_g += g
            total_b += b

    average_r = total_r // num_sets
    average_g = total_g // num_sets
    average_b = total_b // num_sets
```

Now we ask how to run the program.

Prompt:

how would I run this?

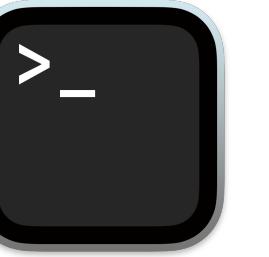


Response:

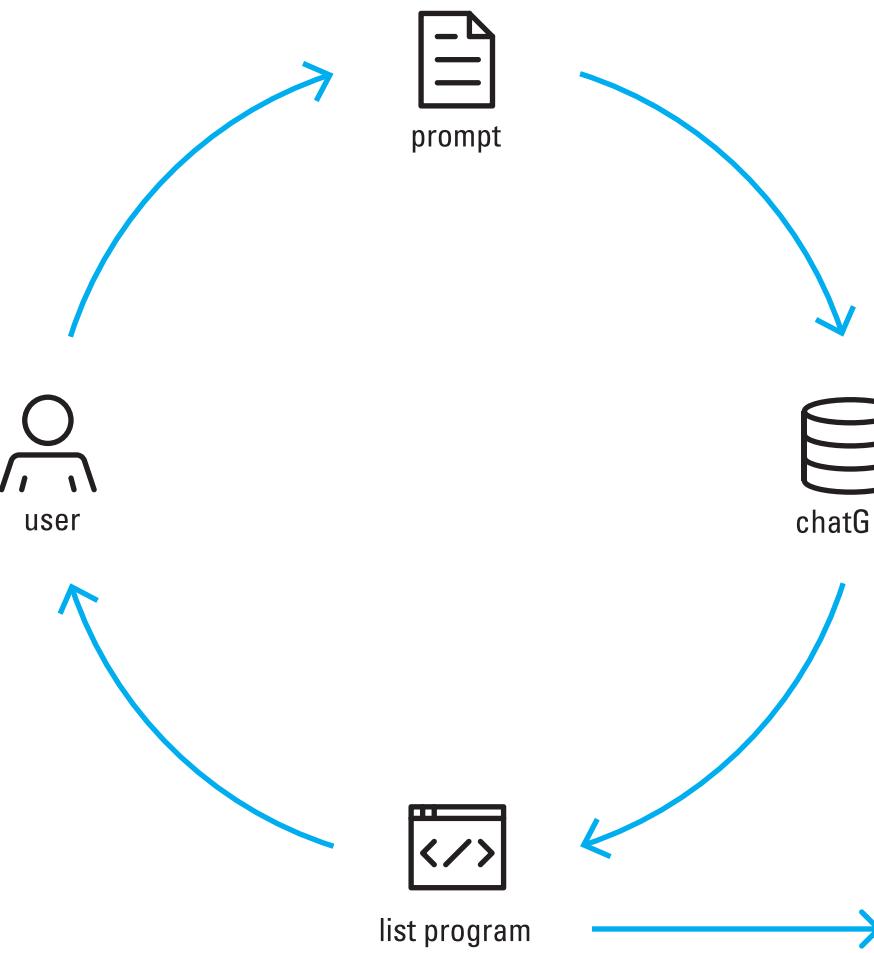
To run the Python program, you can follow these steps:

1. Install Python: Make sure you have Python installed on your computer. You can download the latest version of Python from the official Python website (<https://www.python.org/downloads/>) and follow the installation instructions for your operating system.
 2. Set up a text editor or integrated development environment (IDE): Choose a text editor or IDE to write and run your Python code. Popular options include Visual Studio Code, PyCharm, Sublime Text, or the built-in IDLE editor that comes with the Python installation.
 3. Create a new Python file: Open your preferred text editor or IDE and create a new file with a .py extension. For example, you can name it `venn_diagram.py`.
 4. Copy the program code: Copy the entire code provided in the previous response and paste it into the newly created Python file.
 5. Save the file: Save the Python file.
 6. Run the program: Open your command prompt or terminal, navigate to the directory where you saved the Python file (`venn_diagram.py`), and execute the following command:
`python venn_diagram.py`
 7. bashCopy code
 8. `python venn_diagram.py`
 9. This command runs the Python program and initiates the calculation of the average colors for each section. The results will be displayed in the command prompt or terminal.
- Ensure that you have Python correctly installed and that the Python executable is added to your system's PATH environment variable. This allows you to run Python programs from any directory without specifying the full path to the Python interpreter.
- Note: The program assumes you have Python 3.x installed. If you have both Python 2.x and 3.x installed, you may need to replace `python` with `python3` in the command mentioned above.

To run the program,
I downloaded python,
copied the code intoTextEdit,
saved it to a new folder as a python file (.py),
navigated to the folder within terminal
and ran the program using the command:
`python3 programName.py`



We now have a program that outputs a list of all 127 colors,
given the 7 base colors.

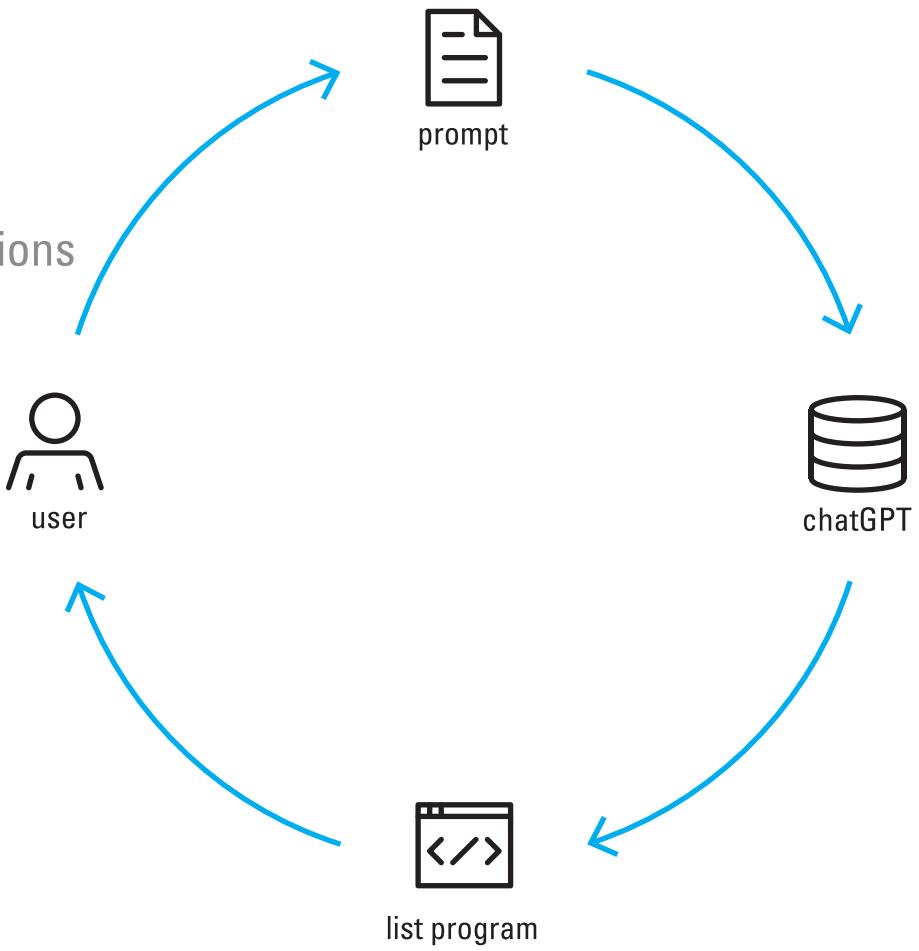


```
Combination: (1,), Average Color: #FF0000
Combination: (2,), Average Color: #FFAA00
Combination: (3,), Average Color: #FFFF00
Combination: (4,), Average Color: #00FF00
Combination: (5,), Average Color: #00B1FF
Combination: (6,), Average Color: #8000FF
Combination: (7,), Average Color: #FF00CC
Combination: (1, 2), Average Color: #FF5500
Combination: (1, 3), Average Color: #FF7F00
Combination: (1, 4), Average Color: #7F7F00
Combination: (1, 5), Average Color: #7F587F
Combination: (1, 6), Average Color: #BF007F
Combination: (1, 7), Average Color: #FF0066
Combination: (2, 3), Average Color: #FFD400
Combination: (2, 4), Average Color: #7FD400
Combination: (2, 5), Average Color: #7FAD7F
Combination: (2, 6), Average Color: #BF557F
Combination: (2, 7), Average Color: #FF5566
Combination: (3, 4), Average Color: #7FFF00
Combination: (3, 5), Average Color: #7FD87F
Combination: (3, 6), Average Color: #BF7F7F
Combination: (3, 7), Average Color: #FF7F66
Combination: (4, 5), Average Color: #00D87F
Combination: (4, 6), Average Color: #407F7F
Combination: (4, 7), Average Color: #7F7F66
Combination: (5, 6), Average Color: #4058FF
Combination: (5, 7), Average Color: #7F58E5
Combination: (6, 7), Average Color: #BF00E5
Combination: (1, 2, 3), Average Color: #FF8D00
Combination: (1, 2, 4), Average Color: #AA8D00
Combination: (1, 2, 5), Average Color: #AA7355
Combination: (1, 2, 6), Average Color: #D43855
Combination: (1, 2, 7), Average Color: #FF3844
Combination: (1, 3, 4), Average Color: #AAAA00
Combination: (1, 3, 5), Average Color: #AA9055
Combination: (1, 3, 6), Average Color: #D45555
```

Next, we ask the model to update the program to output an .svg of the color palette, so that we can import the colors into Adobe Illustrator

Prompt:

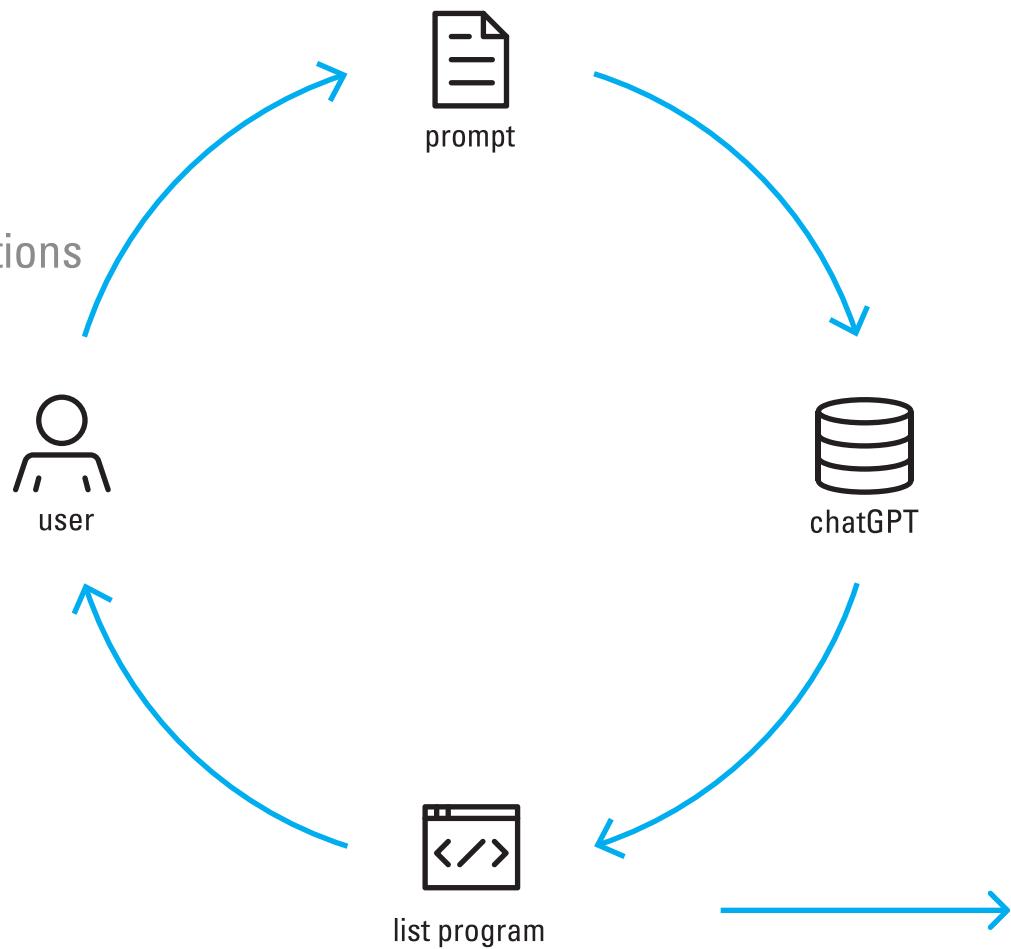
Now can you write the program, but instead of outputting a list, it outputs an .svg file of all the color combinations as squares, organized by how many colors are in the combinations?



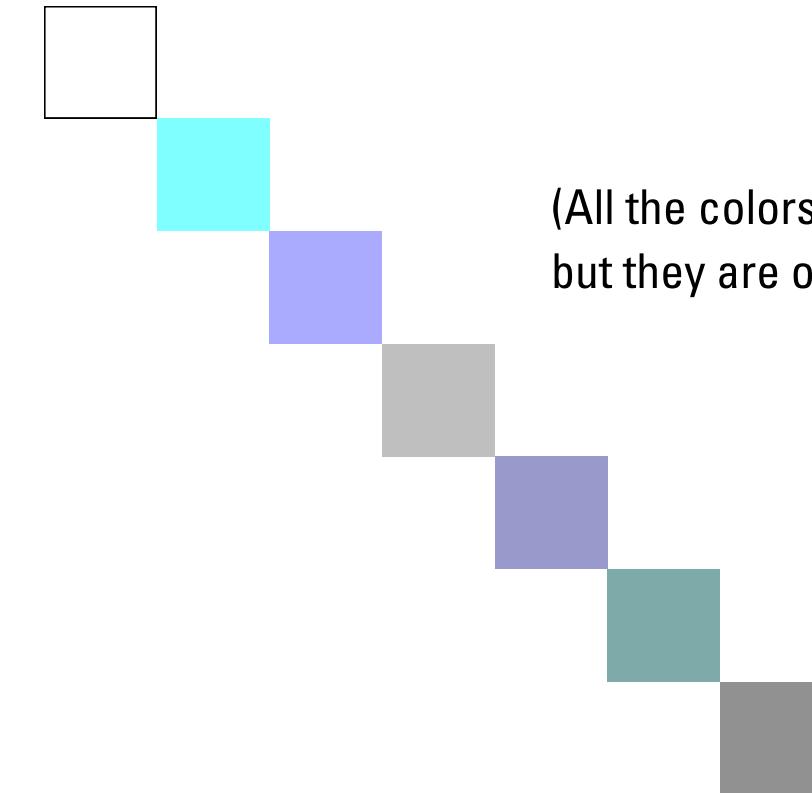
The first attempt gives us a palette where the colors are overlapping.

Prompt:

Now can you write the program,
but instead of outputting a list,
it outputs an .svg file of all the color combinations
as squares, organized by how many colors
are in the combinations?



Response output:

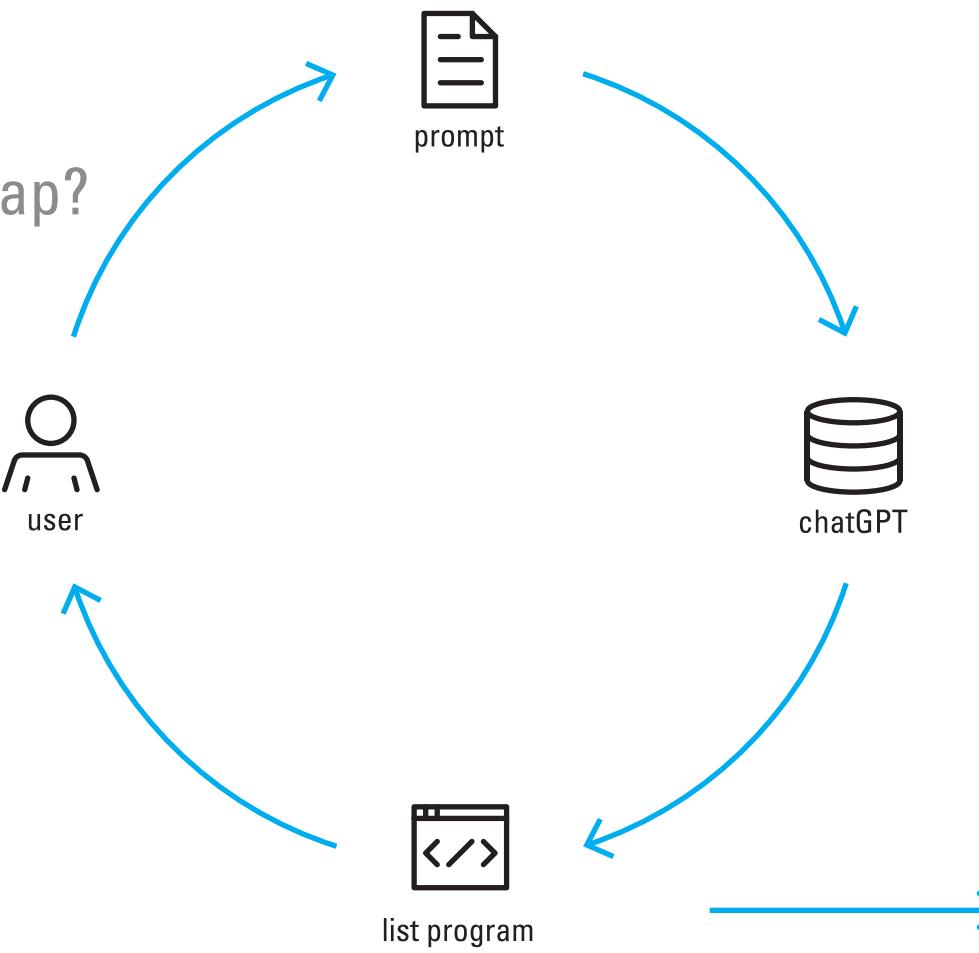


(All the colors are in this file,
but they are overlapping each other)

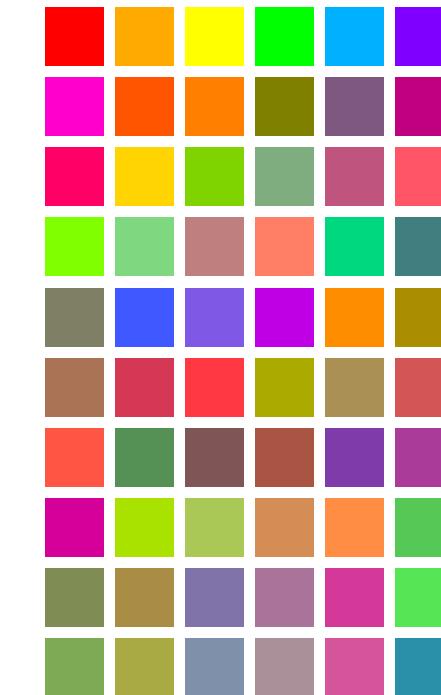
So we refine our prompt, and this time the canvas size is too small.

Prompt:

can you edit this code
so that the rectangles do not overlap?



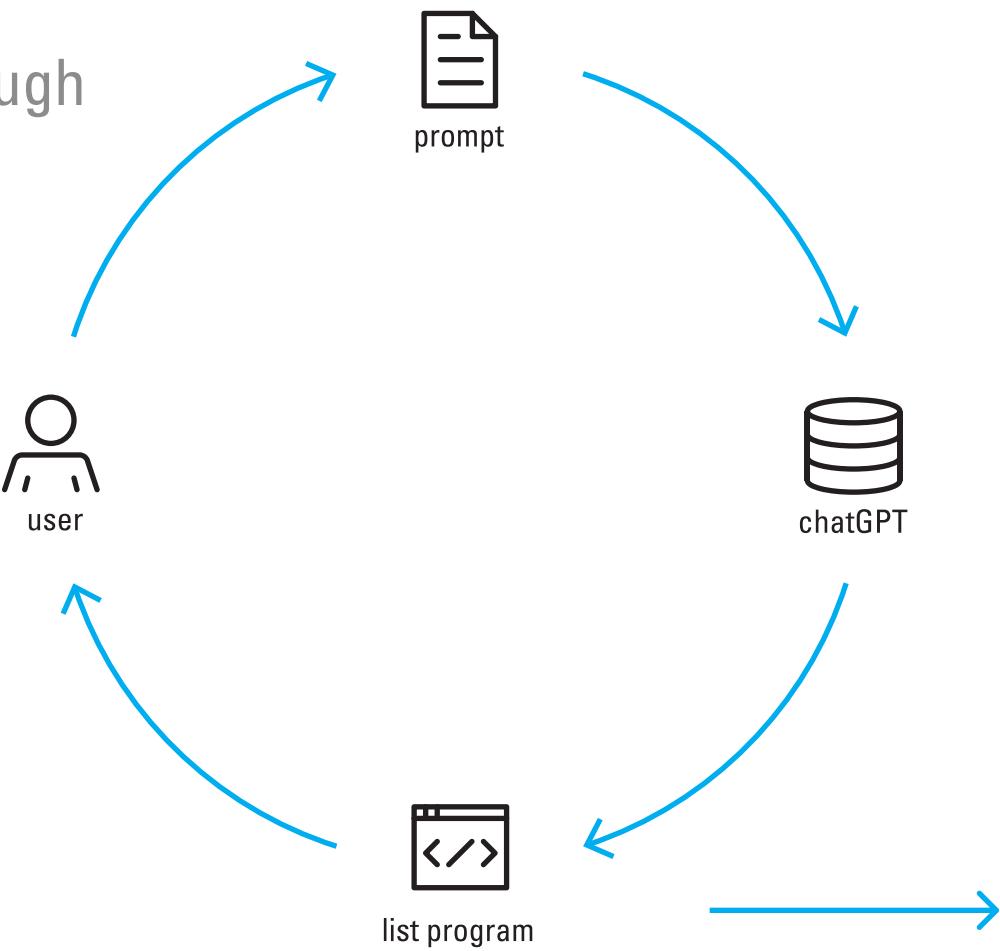
Response output:



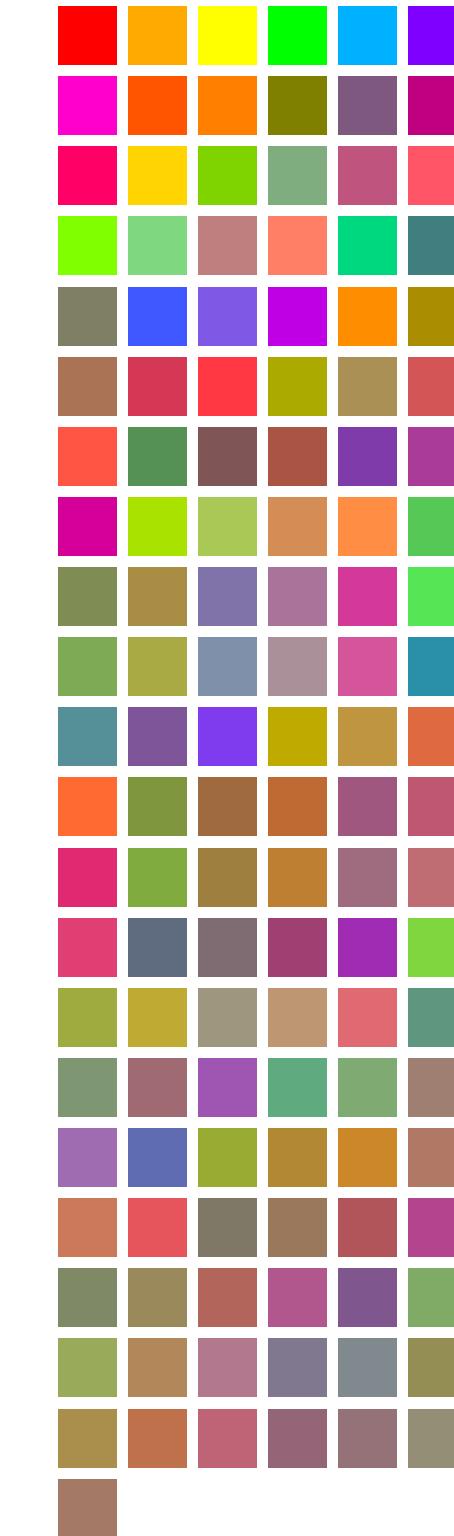
We ask for a bigger canvas size, and now we have the full palette.

Prompt:

it works but the .svg is not tall enough
to fit all the squares



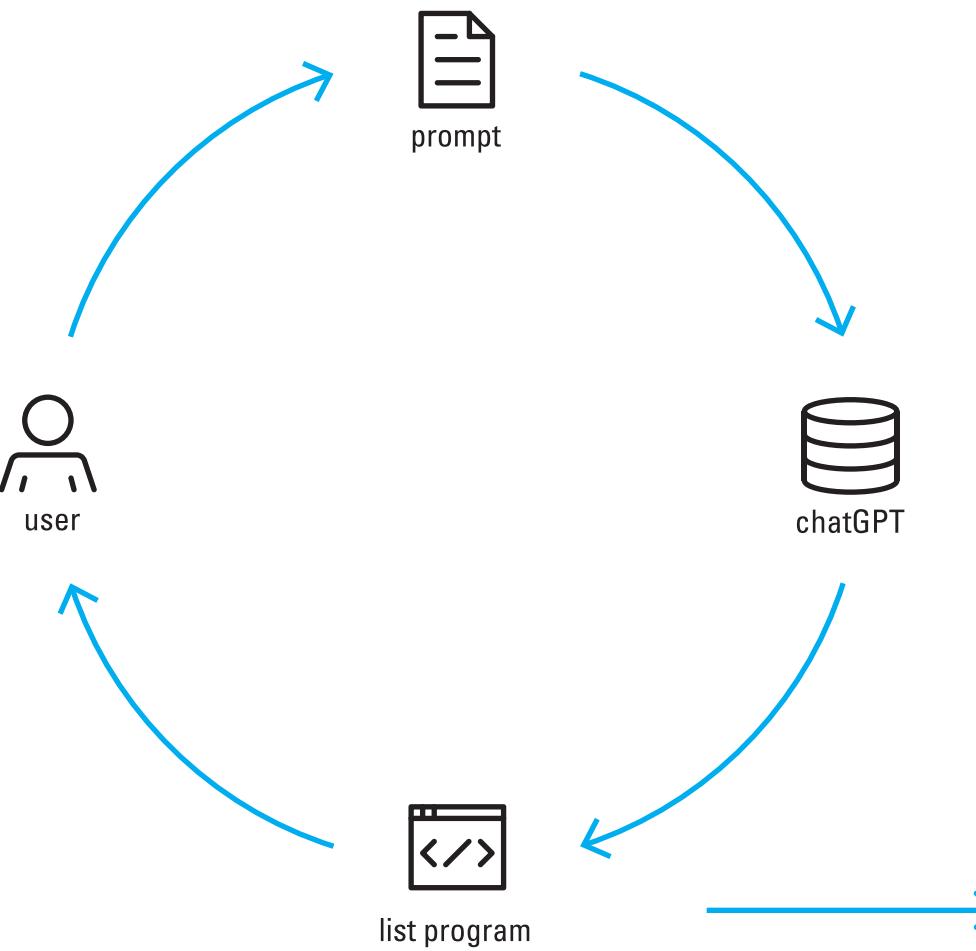
Response output:



**We ask for a labels on the palette,
so we know which base colors are in the combinations.**

Prompt:

Can you edit this code
so that every square
is labeled with the
corresponding combination?

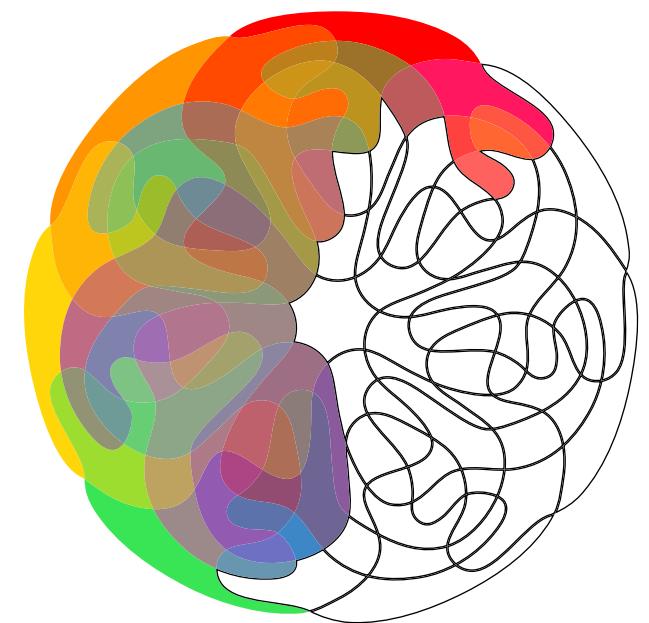


Response output:

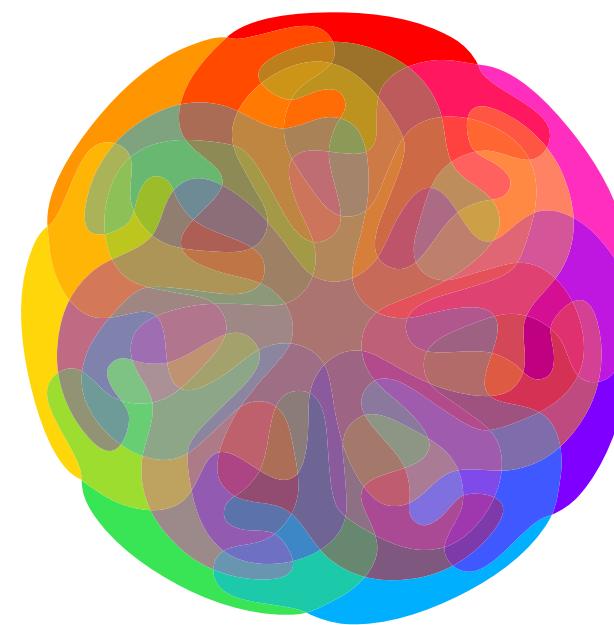
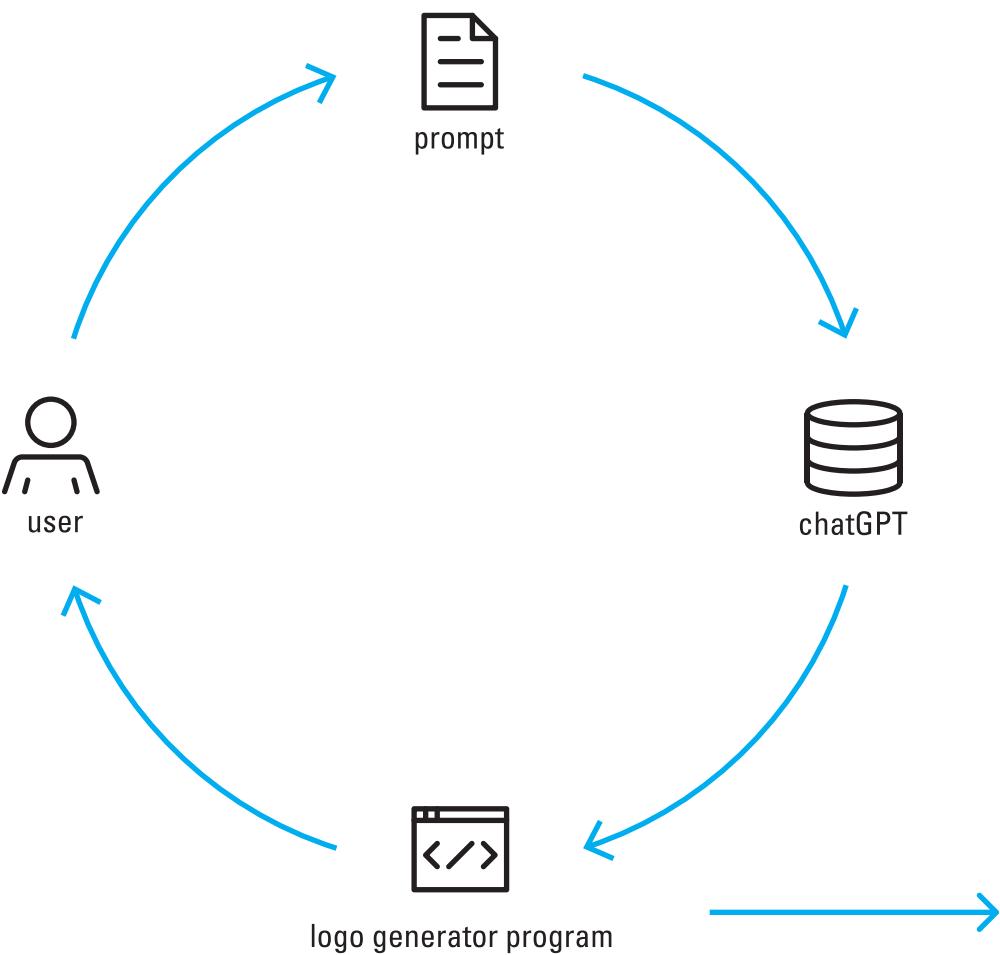
(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)
(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)
(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)
(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)
(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)
(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)
(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)
(2,1)	(3,2)	(4,3)	(5,4)	(6,5)	(7,6)
(2,2)	(3,3)	(4,4)	(5,5)	(6,6)	(7,7)
(2,3)	(3,4)	(4,5)	(5,6)	(6,7)	(7,8)
(2,4)	(3,5)	(4,6)	(5,7)	(6,8)	(7,9)
(2,5)	(3,6)	(4,7)	(5,8)	(6,9)	(7,10)
(2,6)	(3,7)	(4,8)	(5,9)	(6,10)	(7,11)
(2,7)	(3,8)	(4,9)	(5,10)	(6,11)	(7,12)
(3,1)	(4,2)	(5,3)	(6,4)	(7,5)	(8,6)
(3,2)	(4,3)	(5,4)	(6,5)	(7,6)	(8,7)
(3,3)	(4,4)	(5,5)	(6,6)	(7,7)	(8,8)
(3,4)	(4,5)	(5,6)	(6,7)	(7,8)	(8,9)
(3,5)	(4,6)	(5,7)	(6,8)	(7,9)	(8,10)
(3,6)	(4,7)	(5,8)	(6,9)	(7,10)	(8,11)
(3,7)	(4,8)	(5,9)	(6,10)	(7,11)	(8,12)
(4,1)	(5,2)	(6,3)	(7,4)	(8,5)	(9,6)
(4,2)	(5,3)	(6,4)	(7,5)	(8,6)	(9,7)
(4,3)	(5,4)	(6,5)	(7,6)	(8,7)	(9,8)
(4,4)	(5,5)	(6,6)	(7,7)	(8,8)	(9,9)
(4,5)	(5,6)	(6,7)	(7,8)	(8,9)	(9,10)
(4,6)	(5,7)	(6,8)	(7,9)	(8,10)	(9,11)
(4,7)	(5,8)	(6,9)	(7,10)	(8,11)	(9,12)
(5,1)	(6,2)	(7,3)	(8,4)	(9,5)	(10,6)
(5,2)	(6,3)	(7,4)	(8,5)	(9,6)	(10,7)
(5,3)	(6,4)	(7,5)	(8,6)	(9,7)	(10,8)
(5,4)	(6,5)	(7,6)	(8,7)	(9,8)	(10,9)
(5,5)	(6,6)	(7,7)	(8,8)	(9,9)	(10,10)
(5,6)	(6,7)	(7,8)	(8,9)	(9,10)	(10,11)
(5,7)	(6,8)	(7,9)	(8,10)	(9,11)	(10,12)
(6,1)	(7,2)	(8,3)	(9,4)	(10,5)	(11,6)
(6,2)	(7,3)	(8,4)	(9,5)	(10,6)	(11,7)
(6,3)	(7,4)	(8,5)	(9,6)	(10,7)	(11,8)
(6,4)	(7,5)	(8,6)	(9,7)	(10,8)	(11,9)
(6,5)	(7,6)	(8,7)	(9,8)	(10,9)	(11,10)
(6,6)	(7,7)	(8,8)	(9,9)	(10,10)	(11,11)
(6,7)	(7,8)	(8,9)	(9,10)	(10,11)	(11,12)
(7,1)	(8,2)	(9,3)	(10,4)	(11,5)	(12,6)
(7,2)	(8,3)	(9,4)	(10,5)	(11,6)	(12,7)
(7,3)	(8,4)	(9,5)	(10,6)	(11,7)	(12,8)
(7,4)	(8,5)	(9,6)	(10,7)	(11,8)	(12,9)
(7,5)	(8,6)	(9,7)	(10,8)	(11,9)	(12,10)
(7,6)	(8,7)	(9,8)	(10,9)	(11,10)	(12,11)
(7,7)	(8,8)	(9,9)	(10,10)	(11,11)	(12,12)
(8,1)	(9,2)	(10,3)	(11,4)	(12,5)	(13,6)
(8,2)	(9,3)	(10,4)	(11,5)	(12,6)	(13,7)
(8,3)	(9,4)	(10,5)	(11,6)	(12,7)	(13,8)
(8,4)	(9,5)	(10,6)	(11,7)	(12,8)	(13,9)
(8,5)	(9,6)	(10,7)	(11,8)	(12,9)	(13,10)
(8,6)	(9,7)	(10,8)	(11,9)	(12,10)	(13,11)
(8,7)	(9,8)	(10,9)	(11,10)	(12,11)	(13,12)
(9,1)	(10,2)	(11,3)	(12,4)	(13,5)	(14,6)
(9,2)	(10,3)	(11,4)	(12,5)	(13,6)	(14,7)
(9,3)	(10,4)	(11,5)	(12,6)	(13,7)	(14,8)
(9,4)	(10,5)	(11,6)	(12,7)	(13,8)	(14,9)
(9,5)	(10,6)	(11,7)	(12,8)	(13,9)	(14,10)
(9,6)	(10,7)	(11,8)	(12,9)	(13,10)	(14,11)
(9,7)	(10,8)	(11,9)	(12,10)	(13,11)	(14,12)
(10,1)	(11,2)	(12,3)	(13,4)	(14,5)	(15,6)
(10,2)	(11,3)	(12,4)	(13,5)	(14,6)	(15,7)
(10,3)	(11,4)	(12,5)	(13,6)	(14,7)	(15,8)
(10,4)	(11,5)	(12,6)	(13,7)	(14,8)	(15,9)
(10,5)	(11,6)	(12,7)	(13,8)	(14,9)	(15,10)
(10,6)	(11,7)	(12,8)	(13,9)	(14,10)	(15,11)
(10,7)	(11,8)	(12,9)	(13,10)	(14,11)	(15,12)
(11,1)	(12,2)	(13,3)	(14,4)	(15,5)	(16,6)
(11,2)	(12,3)	(13,4)	(14,5)	(15,6)	(16,7)
(11,3)	(12,4)	(13,5)	(14,6)	(15,7)	(16,8)
(11,4)	(12,5)	(13,6)	(14,7)	(15,8)	(16,9)
(11,5)	(12,6)	(13,7)	(14,8)	(15,9)	(16,10)
(11,6)	(12,7)	(13,8)	(14,9)	(15,10)	(16,11)
(11,7)	(12,8)	(13,9)	(14,10)	(15,11)	(16,12)
(12,1)	(13,2)	(14,3)	(15,4)	(16,5)	(17,6)
(12,2)	(13,3)	(14,4)	(15,5)	(16,6)	(17,7)
(12,3)	(13,4)	(14,5)	(15,6)	(16,7)	(17,8)
(12,4)	(13,5)	(14,6)	(15,7)	(16,8)	(17,9)
(12,5)	(13,6)	(14,7)	(15,8)	(16,9)	(17,10)
(12,6)	(13,7)	(14,8)	(15,9)	(16,10)	(17,11)
(12,7)	(13,8)	(14,9)	(15,10)	(16,11)	(17,12)
(13,1)	(14,2)	(15,3)	(16,4)	(17,5)	(18,6)
(13,2)	(14,3)	(15,4)	(16,5)	(17,6)	(18,7)
(13,3)	(14,4)	(15,5)	(16,6)	(17,7)	(18,8)
(13,4)	(14,5)	(15,6)	(16,7)	(17,8)	(18,9)
(13,5)	(14,6)	(15,7)	(16,8)	(17,9)	(18,10)
(13,6)	(14,7)	(15,8)	(16,9)	(17,10)	(18,11)
(13,7)	(14,8)	(15,9)	(16,10)	(17,11)	(18,12)
(14,1)	(15,2)	(16,3)	(17,4)	(18,5)	(19,6)
(14,2)	(15,3)	(16,4)	(17,5)	(18,6)	(19,7)
(14,3)	(15,4)	(16,5)	(17,6)	(18,7)	(19,8)
(14,4)	(15,5)	(16,6)	(17,7)	(18,8)	(19,9)
(14,5)	(15,6)	(16,7)	(17,8)	(18,9)	(19,10)
(14,6)	(15,7)	(16,8)	(17,9)	(18,10)	(19,11)
(14,7)	(15,8)	(16,9)	(17,10)	(18,11)	(19,12)
(15,1)	(16,2)	(17,3)	(18,4)	(19,5)	(20,6)
(15,2)	(16,3)	(17,4)	(18,5)	(19,6)	(20,7)
(15,3)	(16,4)	(17,5)	(18,6)	(19,7)	(20,8)
(15,4)	(16,5)	(17,6)	(18,7)	(19,8)	(20,9)
(15,5)	(16,6)	(17,7)	(18,8)	(19,9)	(20,10)
(15,6)	(16,7)	(17,8)	(18,9)	(19,10)	(20,11)
(15,7)	(16,8)	(17,9)	(18,10)	(19,11)	(20,12)
(16,1)	(17,2)	(18,3)	(19,4)	(20,5)	(21,6)
(16,2)	(17,3)	(18,4)	(19,5)	(20,6)	(21,7)
(16,3)	(17,4)	(18,5)	(19,6)	(20,7)	(21,8)
(16,4)	(17,5)	(18,6)	(19,7)	(20,8)	(21,9)
(16,5)	(17,6)	(18,7)	(19,8)	(20,9)	(21,10)
(16,6)	(17,7)	(18,8)	(19,9)	(20,10)	(21,11)
(16,7)	(17,8)	(18,9)	(19,10)	(20,11)	(21,12)
(17,1)	(18,2)	(19,3)	(20,4)	(21,5)	(22,6)
(17,2)	(18,3)	(19,4)	(20,5)	(21,6)	(22,7)
(17,3)	(18,4)	(19,5)	(20,6)	(21,7)	(22,8)
(17,4)	(18,5)	(19,6)	(20,7)	(21,8)	(22,9)
(17,5)	(18,6)	(19,7)	(20,8)	(21,9)	(22,10)
(17,6)	(18,7)	(19,8)	(20,9)	(21,10)	(22,11)
(17,7)	(18,8)	(19,9)	(20,10)	(21,11)	(22,12)
(18,1)	(19,2)	(20,3)	(21,4)	(22,5)	(23,6)
(18,2)	(19,3)	(20,4)	(21,5)	(22,6)	(23,7)
(18,3)	(19,4)	(20,5)	(21,6)	(22,7)	(23,8)
(18,4)	(19,5)	(20,6)	(21,7)	(22,8)	(23,9)
(18,5)	(19,6)	(20,7)	(21,8)	(22,9)	(23,10)
(18,6)	(19,7)	(20,8)	(21,9)	(22,10)	(23,11)
(18,7)	(19,8)	(20,9)	(21,10)	(22,11)	(23,12)
(19,1)	(20				

**Even with the ability to generate a labeled color palette,
New versions of the logo still need to be filled in manually
with the eyedropper tool.**

(1)	(2)	(3)	(4)	(5)	(6)
(7)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
(1, 7)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	(2, 7)
(3, 4)	(3, 5)	(3, 6)	(3, 7)	(4, 5)	(4, 6)
(4, 7)	(5, 6)	(5, 7)	(6, 7)	(1, 2, 3)	(1, 2, 4)
(1, 2, 5)	(1, 2, 6)	(1, 2, 7)	(1, 3, 4)	(1, 3, 5)	(1, 3, 6)
(1, 3, 7)	(1, 4, 5)	(1, 4, 6)	(1, 4, 7)	(1, 5, 6)	(1, 5, 7)
(1, 6, 7)	(2, 3, 4)	(2, 3, 5)	(2, 3, 6)	(2, 3, 7)	(2, 4, 5)
(2, 4, 6)	(2, 4, 7)	(2, 5, 6)	(2, 5, 7)	(2, 6, 7)	(3, 4, 5)
(3, 4, 6)	(3, 4, 7)	(3, 5, 6)	(3, 5, 7)	(3, 6, 7)	(4, 5, 6)
(4, 5, 7)	(4, 6, 7)	(5, 6, 7)	(1, 2, 3, 4)	(1, 2, 3, 5)	(1, 2, 3, 6)
(1, 2, 3, 7)	(1, 2, 4, 5)	(1, 2, 4, 6)	(1, 2, 4, 7)	(1, 2, 5, 6)	(1, 2, 5, 7)
(1, 2, 6, 7)	(1, 3, 4, 5)	(1, 3, 4, 6)	(1, 3, 4, 7)	(1, 3, 5, 6)	(1, 3, 5, 7)
(1, 3, 6, 7)	(1, 4, 5, 6)	(1, 4, 5, 7)	(1, 4, 6, 7)	(1, 5, 6, 7)	(2, 3, 4, 5)
(2, 3, 4, 6)	(2, 3, 4, 7)	(2, 3, 5, 6)	(2, 3, 5, 7)	(2, 3, 6, 7)	(2, 4, 5, 6)



The next step is to automate the process entirely.



**First I tried prompting chatGPT with an .svg of the unfilled logo,
and ask it to write a program to color it in.**

**The .svg is longer than ChatGPT's context window,
it cannot generate a response.**



The message you submitted was too long, please reload the conversation and submit something shorter.

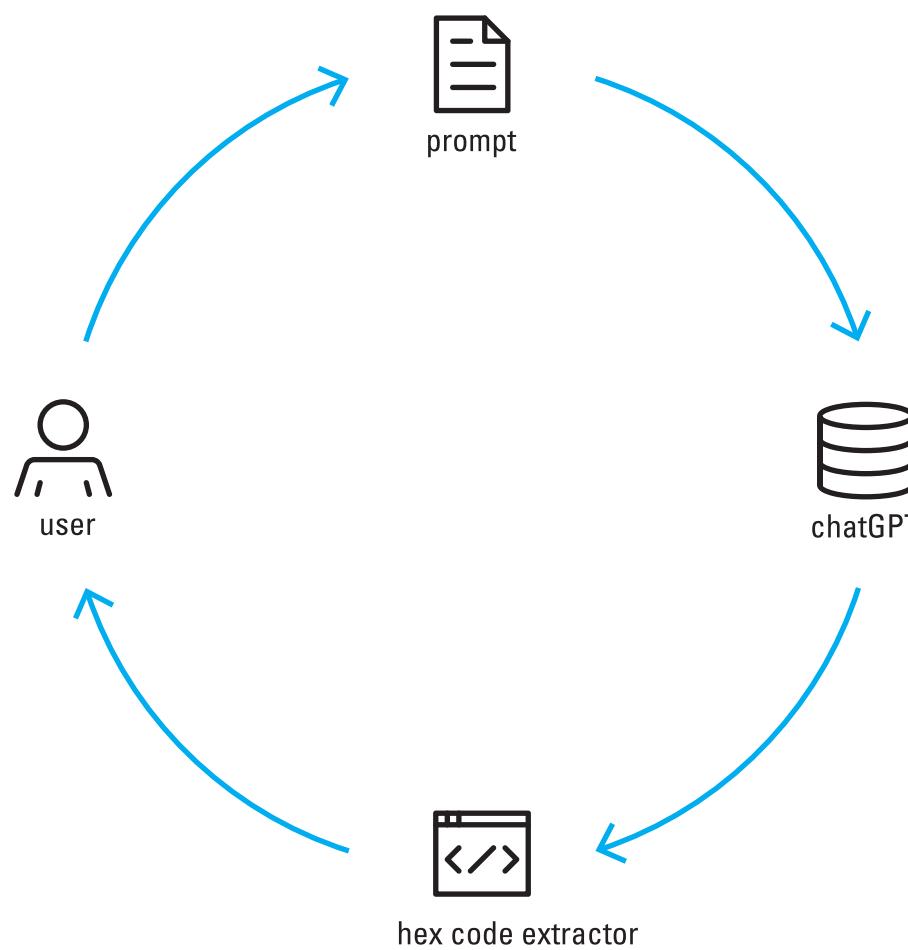
How can we prompt ChatGPT to write a program to modify an .svg that it cannot read?

The .svg of the logo contains a list of styles containing all of the hex codes in the logo.

We need a program that can identify these hex codes, and replace them.

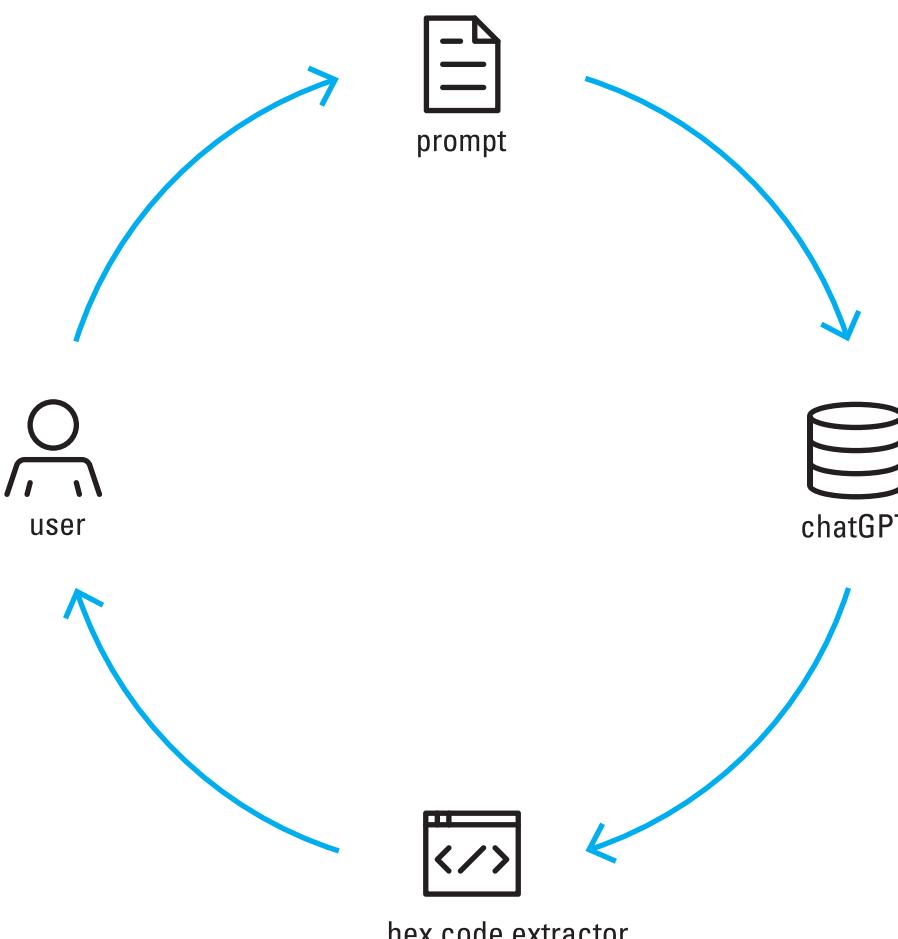
We prompt ChatGPT to create a program that scrapes all hex codes from any given .svg.

Can you write a program that takes an .svg as an input, and extracts all the hex codes for the fill colors and outputs a list of them?



It responds with a program.

Can you write a program that takes an .svg as an input, and extracts all the hex codes for the fill colors and outputs a list of them?



```
import re

def extract_fill_hexcodes(svg_content):
    # Regular expression pattern to find fill attributes with hexcodes
    pattern = r'fill="#([A-Fa-f0-9]{6})([A-Fa-f0-9]{3})"'
    fill_hexcodes = re.findall(pattern, svg_content)
    return fill_hexcodes

def main():
    input_svg_file = "input.svg" # Replace with your input SVG file path
    output_file = "fill_hexcodes.txt" # Replace with the desired output file path

    with open(input_svg_file, "r") as file:
        svg_content = file.read()

    fill_hexcodes = extract_fill_hexcodes(svg_content)

    with open(output_file, "w") as output:
        for hexcode in fill_hexcodes:
            output.write(hexcode + "\n")

    print("Fill hexcodes extracted and saved to", output_file)

if __name__ == "__main__":
    main()
```

If we pass it the DrumWave logo, it outputs a list of all 127 colors.

```
import re

def extract_fill_hexcodes(svg_content):
    # Regular expression pattern to find fill attributes with hexcodes
    pattern = r'fill="#[A-Fa-f0-9]{6}#[A-Fa-f0-9]{3}"'
    fill_hexcodes = re.findall(pattern, svg_content)
    return fill_hexcodes

def main():
    input_svg_file = "input.svg" # Replace with your input SVG file path
    output_file = "fill_hexcodes.txt" # Replace with the desired output file path

    with open(input_svg_file, "r") as file:
        svg_content = file.read()

    fill_hexcodes = extract_fill_hexcodes(svg_content)

    with open(output_file, "w") as output:
        for hexcode in fill_hexcodes:
            output.write(hexcode + "\n")

    print("Fill hexcodes extracted and saved to", output_file)

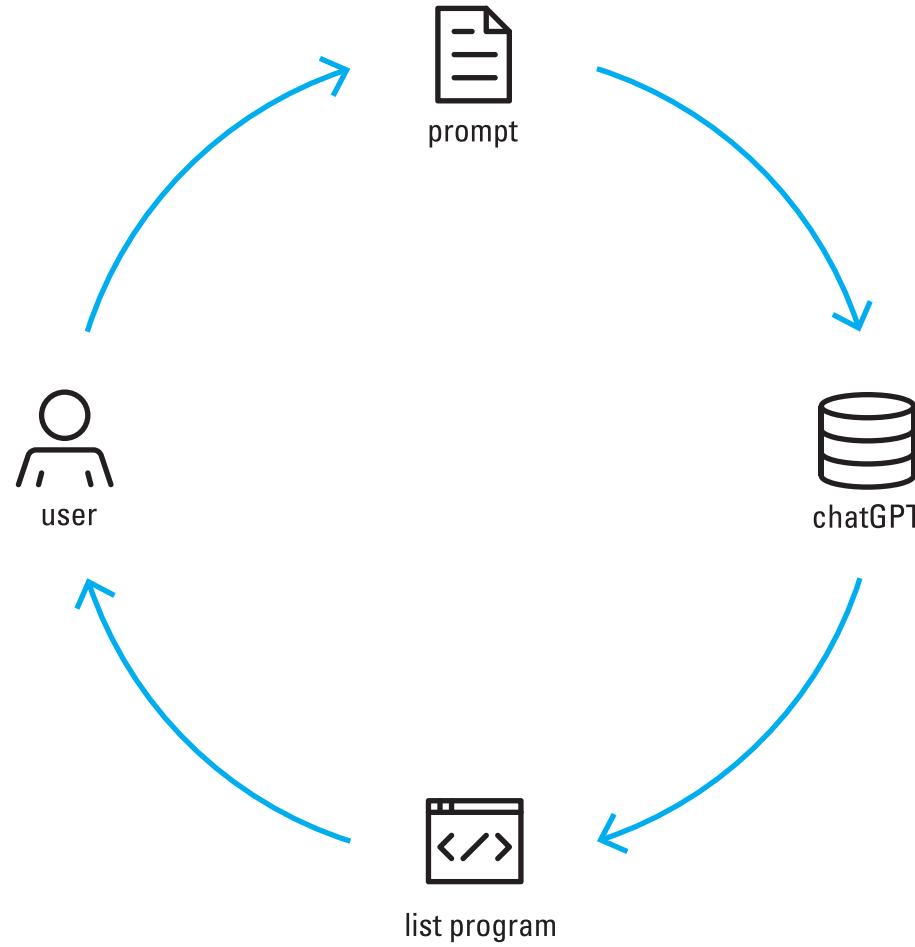
if __name__ == "__main__":
    main()
```



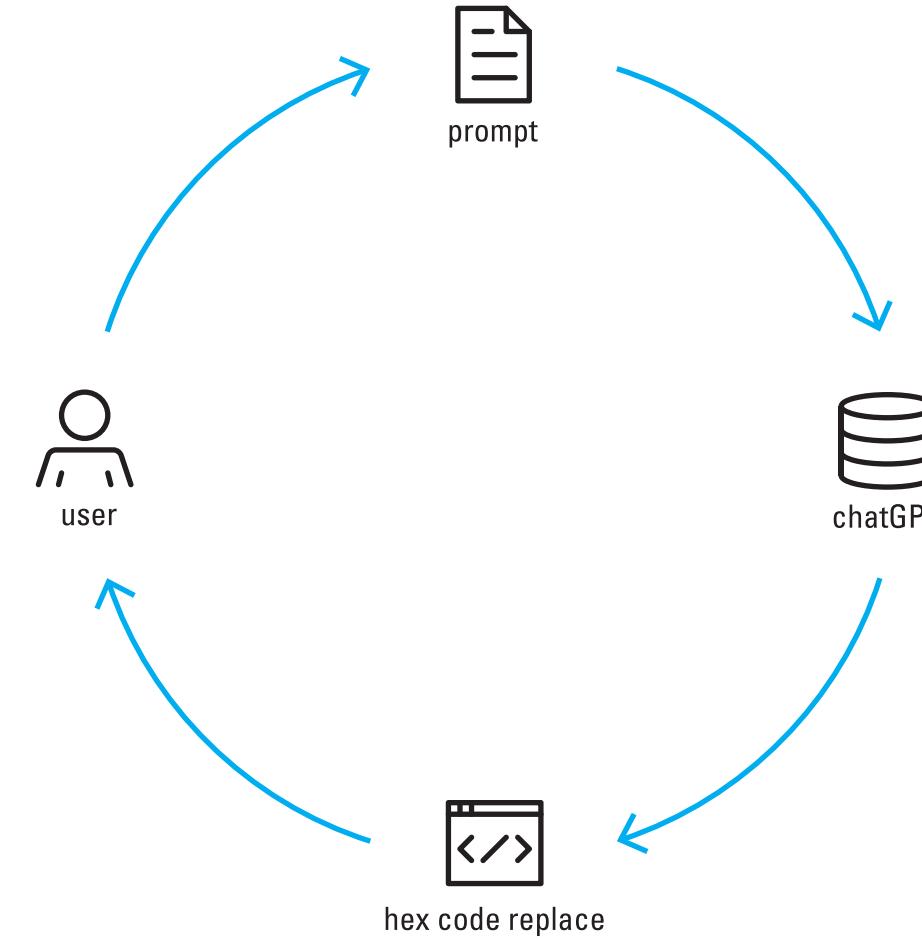
BF00E5	7FAA55	9F3F72	7F8966
FF8D00	AAAA44	9F2CB2	99895B
AA8D00	7F90AA	7FD63F	B2665B
AA7355	AA9099	9FAA3F	B2568E
D43855	D45599	BFAA33	7F568E
FF3844	2A90AA	9F967F	7FAB66
AAAA00	559099	BF9672	99AB5B
AA9055	7F5599	DF6A72	B2885B
D45555	7F3BEE	5F967F	B2788E
FF5544	BFAA00	7F9672	7F788E
559055	BF963F	9F6A72	7F898E
7F5555	DF6A3F	9F56B2	948E55
AA5544	FF6A33	5FAB7F	AA8E4C
7F3BAA	7F963F	7FAB72	BF714C
AA3B99	9F6A3F	9F7F72	BF6477
D40099	BF6A33	9F6CB2	946477
AAE200	9F567F	5F6CB2	947277
AAC855	BF5672	99AB33	948E77
D48D55	DF2A72	B28833	A47A66
FF8D44	7FAB3F	CC8828	
55C855	9F7F3F	B27866	
7F8D55	BF7F33	CC785B	
AA8D44	9F6C7F	E5555B	
7F73AA	BF6C72	7F7866	
AA7399	DF3F72	99785B	
D43899	5F6C7F	B2555B	
55E555	7F6C72	B2458E	

**We already have a program from our first thread
that outputs a new list of hex codes given 7 base colors.**

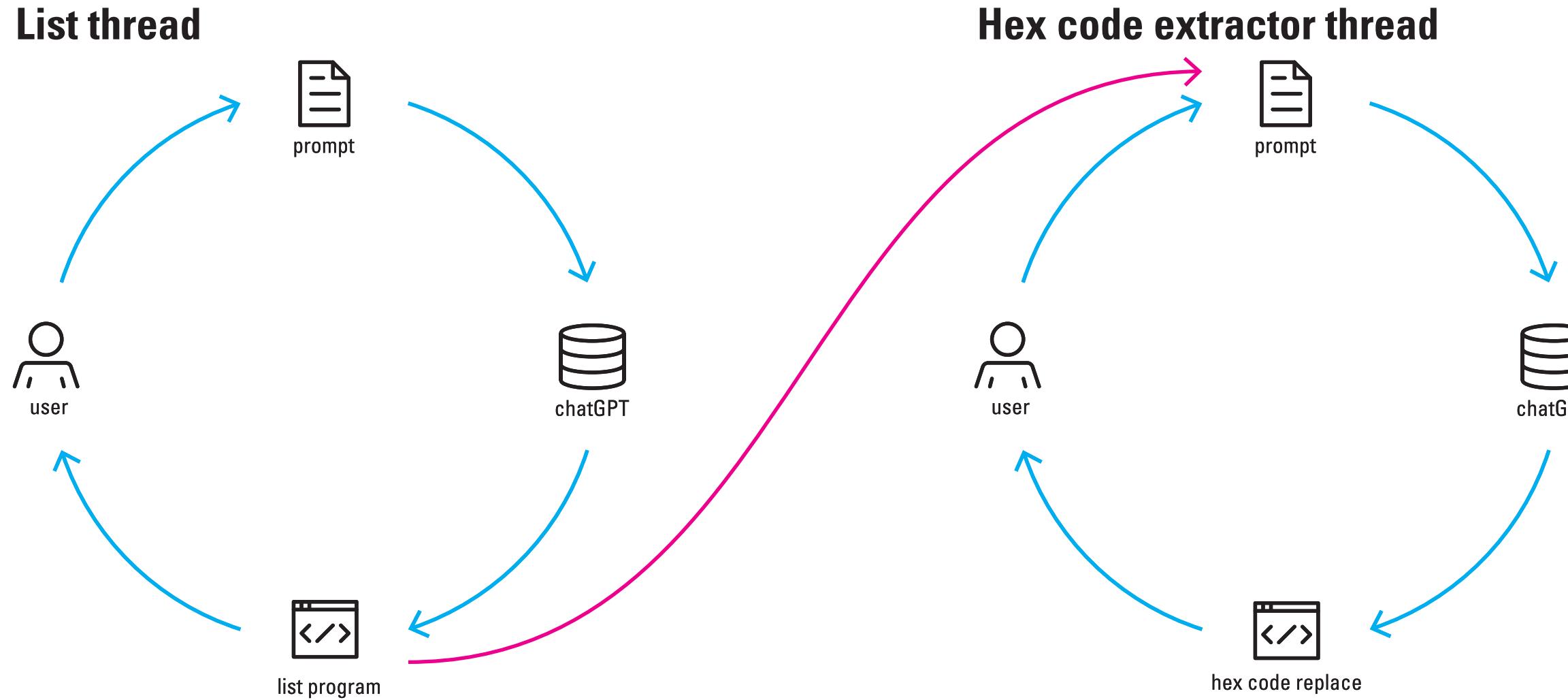
List thread



Hex code extractor thread

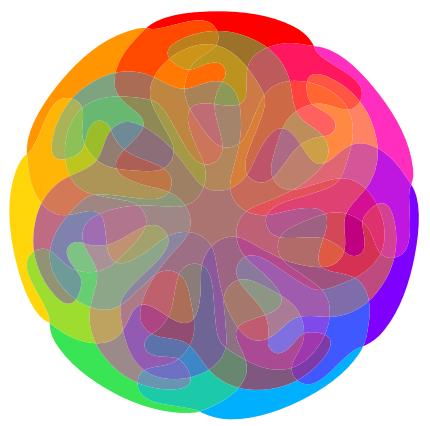


We copy the code from our first thread into a prompt in the current thread, asking Chat GPT to combine the two programs.

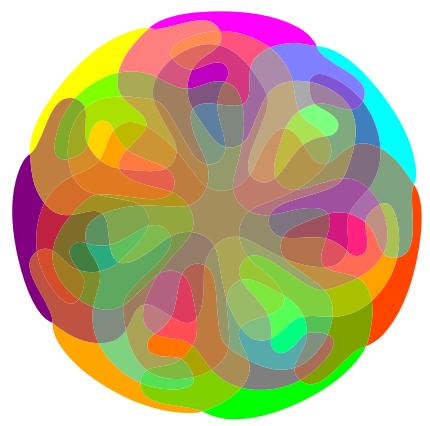


Our final program written by ChatGPT:

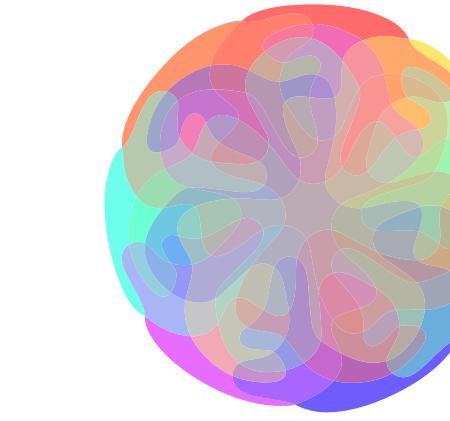
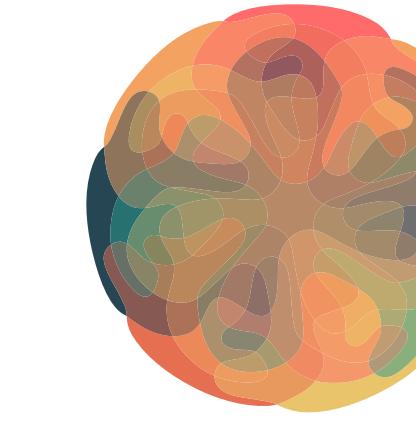
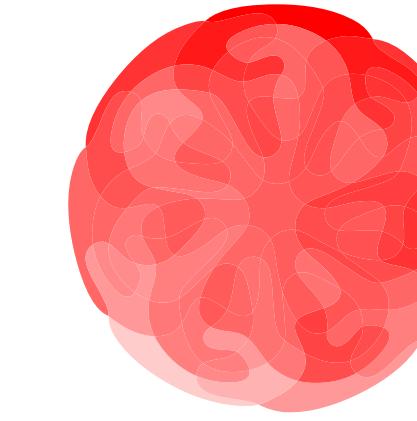
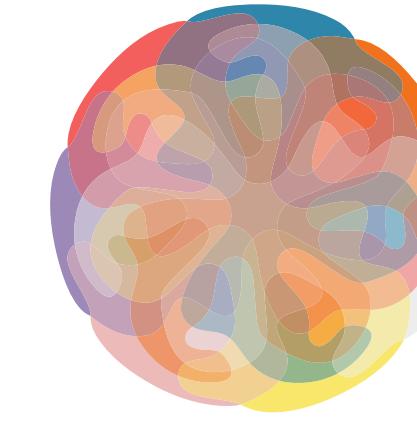
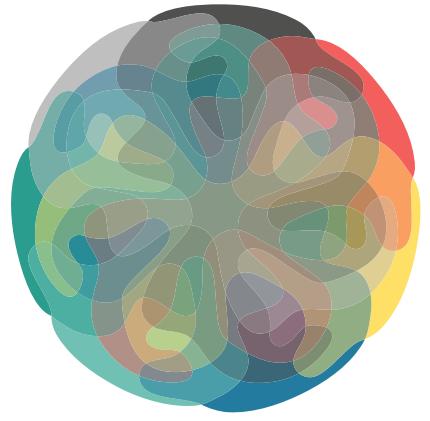
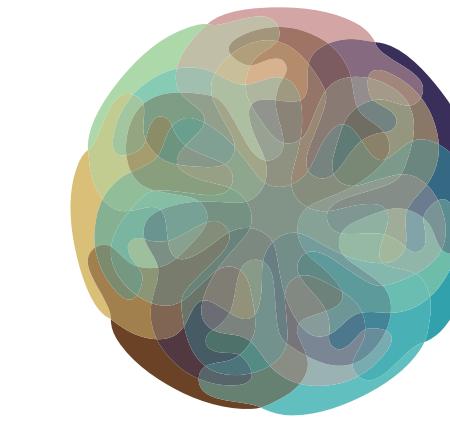
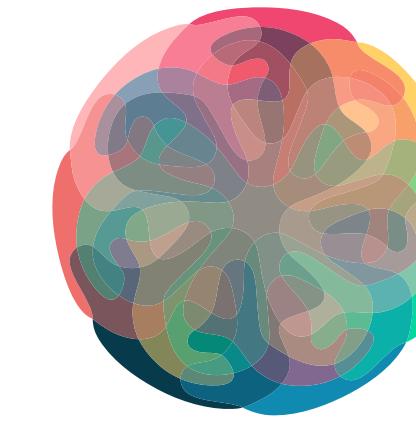
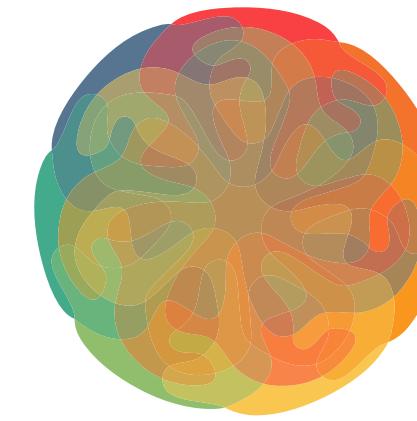
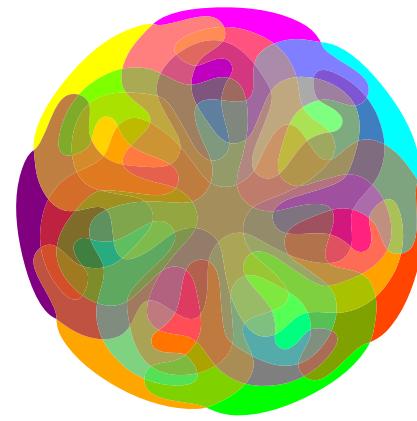
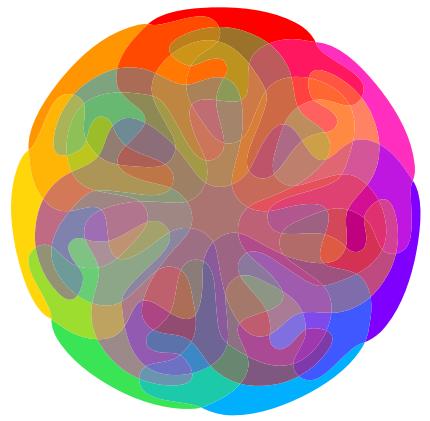
Takes an .svg and 7 base colors as input → **Generates the 127 combinations** → **Scrapes the .svg for hex codes** → **replaces the hex codes with the new list** → **output a recolored .svg**



```
'#FF00FF',  
'#00FFFF',  
'#FF4500',  
'#00FF00',  
'#FFA500',  
'#800080',  
'#FFFF00'
```



Generating new versions of the logo is now trivial.



Appendix

How to create new logos

New versions can be created with two simple steps,

Step 1:

on line 40, enter the file name of the svg you wish to modify:

```
input_svg_file = "dw_logo.svg" # Replace with your input SVG file path
```

Step 2:

on lines 43 - 49, replace the 7 base hex codes with your desired palette:

```
base_hexcodes = [  
    '#FF0000',  
    '#FF3333',  
    '#FF6666',  
    '#FF9999',  
    '#FFCCCC',  
    '#FF6666',  
    '#FF3333'  
]
```

Run the program, and the svg file will be rewritten with your desired colors.

Appendix

Prompts, responses, and code

Prompt:

i am making a picture of a seven set venn diagram. every section overlaps every other section in all possible combinations, resulting in 128 individual sections. If each of the seven sets is a specific color (hex value), and every section is the average color of the sets that overlap in it, how can I calculate the colors for all 128 sections?

Response:

To calculate the average color for each of the 128 sections in a seven-set Venn diagram, where each section represents the overlapping sets, you can follow these steps:

1. Assign a specific color (hex value) to each of the seven sets in your diagram. Let's denote the colors as C1, C2, C3, C4, C5, C6, and C7.
2. Enumerate all the possible combinations of overlapping sets. In this case, since there are seven sets, there will be $2^7 = 128$ combinations.
3. For each combination, determine which sets overlap. Represent the presence or absence of a particular set in the combination using binary digits, where 1 represents the presence of the set and 0 represents its absence. For example, if the combination is {C1, C3, C4, C7}, it can be represented as 1100101, where the first digit corresponds to C1, the second to C2, and so on.
4. Calculate the average color for each section by taking the weighted average of the colors of the overlapping sets. For each set present in the combination, multiply its color value (RGB values or hex code) by the corresponding binary digit (1 or 0). Sum up the resulting color values for all the sets present in the combination. Finally, divide the sum by the number of sets present to get the average color.
5. Repeat steps 3-4 for all 128 combinations, and you will obtain the average color for each section in your Venn diagram.

Note that the calculation of average color can be done using RGB values or hex codes, depending on the format you prefer to work with. Ensure that you correctly handle the conversion between different color formats, if necessary.

By following these steps, you should be able to determine the average color for each of the 128 sections in your seven-set Venn diagram.

Prompt:

can you write a program that does this

Response:

Certainly! Here's an example of a Python program that calculates the average color for each section in a seven-set Venn diagram:

```
pythonCopy code
from itertools import combinations # Hex color values for each set
colors = { 'C1': '#FF0000', 'C2': '#00FF00', 'C3': '#0000FF', 'C4': '#FFFF00', 'C5': '#FF00FF', 'C6': '#00FFFF', 'C7': '#FFFFFF' } # Function to calculate average color
def calculate_average_color(combination):
    total_r, total_g, total_b = 0, 0, 0
    num_sets = sum(combination)
    for i, present in enumerate(combination):
        if present:
            color = colors[f'C{i+1}']
            r, g, b = int(color[1:3], 16), int(color[3:5], 16), int(color[5:7], 16)
            total_r += r
            total_g += g
            total_b += b
    average_r = total_r // num_sets
    average_g = total_g // num_sets
    average_b = total_b // num_sets
    average_color = f'#{average_r:02X}{average_g:02X}{average_b:02X}'
    return average_color # Generate all combinations of sets
combinations_list = list(combinations(range(1, 8), 1)) + list(combinations(range(1, 8), 2)) + \
    list(combinations(range(1, 8), 3)) + list(combinations(range(1, 8), 4)) + \
    list(combinations(range(1, 8), 5)) + list(combinations(range(1, 8), 6)) + \
    list(combinations(range(1, 8), 7)) # Calculate average color for each section for combination in combinations_list:
binary_combination = [1 if i+1 in combination else 0 for i in range(7)]
average_color = calculate_average_color(binary_combination)
print(f"Combination: {combination}, Average Color: {average_color}")
```

(The code was written with good style, indents and line breaks, it has been compressed to fit on the page)

In this program, the colors dictionary holds the hex color values for each set (C1 to C7). The calculate_average_color function takes a combination (represented as a binary list) and calculates the average color based on the presence of each set. The program then generates all possible combinations using the combinations function from the itertools module. Finally, it iterates over each combination, converts it to a binary representation, and calls the calculate_average_color function to obtain the average color for that section. The results are then printed for each combination.

Feel free to customize the program by adding or modifying the hex color values in the colors dictionary to match your desired colors for each set.

Prompt:

can you write a program that takes an svg as an input, and extracts all the hexcodes for the fill colors and outputs a list of them?

Response:

Sure! Below is a Python program that takes an SVG file as input, extracts all the hexcodes for the fill colors, and outputs a list of them.

```
pythonCopy code
import re def extract_fill_hexcodes(svg_content): # Regular expression pattern to find fill attributes with hexcodes pattern = r'fill="#([A-Fa-f0-9]{6})|[A-Fa-f0-9]{3})"' fill_hexcodes = re.findall(pattern, svg_content) return fill_hexcodes def main(): input_svg_file = "input.svg" # Replace with your input SVG file path output_file = "fill_hexcodes.txt" # Replace with the desired output file path with open(input_svg_file, "r") as file: svg_content = file.read() fill_hexcodes = extract_fill_hexcodes(svg_content) with open(output_file, "w") as output: for hexcode in fill_hexcodes: output.write(hexcode + "\n") print("Fill hexcodes extracted and saved to", output_file) if __name__ == "__main__": main()
```

To use this program, make sure you have Python installed on your machine, and replace "input.svg" with the path to your SVG file. After running the program, it will create a new file named "fill_hexcodes.txt" in the same directory, containing all the extracted fill color hexcodes, one per line.

Prompt:

(re-entering the list program as a prompt)

```
from itertools import combinations

# Hex color values for each set
colors = {
    'C1': '#FF0000',
    'C2': '#FFAA00',
    'C3': '#FFFF00',
    'C4': '#00FF00',
    'C5': '#00B1FF',
    'C6': '#8000FF',
    'C7': '#FF00CC'
}

# Function to calculate average color
def calculate_average_color(combination):
    total_r, total_g, total_b = 0, 0, 0
    num_sets = sum(combination)

    for i, present in enumerate(combination):
        if present:
            color = colors[f'C{i+1}']
            r, g, b = int(color[1:3], 16), int(color[3:5], 16), int(color[5:7], 16)
            total_r += r
            total_g += g
            total_b += b

    average_r = total_r // num_sets
    average_g = total_g // num_sets
    average_b = total_b // num_sets

    average_color = f'#{average_r:02X}{average_g:02X}{average_b:02X}'
    return average_color

# Generate all combinations of sets
combinations_list = list(combinations(range(1, 8), 1)) + list(combinations(range(1, 8), 2)) + \
    list(combinations(range(1, 8), 3)) + list(combinations(range(1, 8), 4)) + \
    list(combinations(range(1, 8), 5)) + list(combinations(range(1, 8), 6)) + \
    list(combinations(range(1, 8), 7))

# Calculate average color for each section
for combination in combinations_list:
    binary_combination = [1 if i+1 in combination else 0 for i in range(7)]
    average_color = calculate_average_color(binary_combination)
    print(f"Combination: {combination}, Average Color: {average_color}")
```

Response:

The provided code defines a function to calculate the average color of selected sets from a given set of colors represented by hexcodes. It then generates all possible combinations of the sets and calculates the average color for each combination.

Let's understand the code step by step:

1. The colors dictionary contains the names and corresponding hex color values of each set.
2. The function calculate_average_color takes a combination (represented as a list of 1's and 0's) and calculates the average color of the selected sets.
3. The list combinations_list contains all possible combinations of the sets from 1 to 7.
4. For each combination in combinations_list, it converts the combination into a binary representation, calls the calculate_average_color function to compute the average color, and then prints the combination along with its average color.

Note: The code uses the `itertools.combinations` function to generate the combinations.

If you run this code, it will output all the possible combinations of sets and their corresponding average colors. Keep in mind that the number of combinations is large, so there will be many lines of output.

For example, if you want to select 3 sets and find their average color, you will find combinations like:

yamlCopy code

Combination: (1, 2, 3), Average Color: #FF7F00 Combination: (1, 2, 4), Average Color: #BF7F00 Combination: (1, 2, 5), Average Color: #BF3F7F Combination: (1, 2, 6), Average Color: #FF3F7F ...

These lines indicate that for each combination of three sets, the average color is calculated and displayed as a hexcode.

Prompt:

can you write a program that takes the out put of the program I just sent, (a list of hexcodes) and replaces the hexcodes of a given svg (like the program you wrote above)

Response:

Sure! We can modify the program to take the output of the previous program (the list of hexcodes) and replace the hexcodes of a given SVG file with those extracted hexcodes. We'll update the SVG file directly, so make sure to keep a backup of the original SVG file just in case.

Here's the modified program:

```
import re def extract_fill_hexcodes(svg_content): # Regular expression pattern to find fill attributes with hexcodes pattern = r'fill="#([A-Fa-f0-9]{6})|[A-Fa-f0-9]{3})'" fill_hexcodes = re.findall(pattern, svg_content) return fill_hexcodes def replace_fill_hexcodes(svg_content, new_hexcodes): # Regular expression pattern to find fill attributes with hexcodes pattern = r'fill="#([A-Fa-f0-9]{6})|[A-Fa-f0-9]{3})'" def replace_func(match): return f'fill="{new_hexcodes.pop(0)}"' # Replace hexcodes using re.sub and the custom replace_func new_svg_content = re.sub(pattern, replace_func, svg_content) return new_svg_content def main(): input_svg_file = "input.svg" # Replace with your input SVG file path output_file = "fill_hexcodes.txt" # Replace with the output file from the previous program # Read the SVG file with open(input_svg_file, "r") as file: svg_content = file.read() # Read the list of hexcodes from the output file with open(output_file, "r") as hexcode_file: new_hexcodes = [line.strip() for line in hexcode_file] # Ensure the number of extracted hexcodes matches the number of sets in the SVG if len(new_hexcodes) != 7: print("Error: The number of extracted hexcodes does not match the number of sets in the SVG.") return # Replace the hexcodes in the SVG new_svg_content = replace_fill_hexcodes(svg_content, new_hexcodes) # Write the updated SVG content back to the file with open(input_svg_file, "w") as file: file.write(new_svg_content) print("Hexcodes replaced in the SVG file.") if __name__ == "__main__": main()
```

To use this program, make sure you have Python installed on your machine, and replace "input.svg" with the path to your SVG file and "fill_hexcodes.txt" with the path to the output file generated by the previous program.

The program reads the SVG content from the input file, reads the list of new hexcodes from the output file, and then replaces the hexcodes in the SVG content using the replace_fill_hexcodes function. The updated SVG content is then written back to the input SVG file.

Please use this program with caution and make sure to back up your original SVG file before running it.

Prompt:

can you combine the programs so that it is one program that takes as input an svg file and seven base hexcodes

Response:

Sure! Here's a combined program that takes an SVG file as input and seven base hexcodes. It will extract the fill hexcodes from the SVG, calculate the average colors for all combinations of base hexcodes, and then replace the original hexcodes in the SVG with the calculated average colors.

```
import re from itertools import combinations
def extract_all_hexcodes(svg_content): # Regular expression pattern to find hexcodes in fill and stroke attributes
    pattern = r'(#(?:[A-Fa-f0-9]{6})|[A-Fa-f0-9]{3}))'
    all_hexcodes = re.findall(pattern, svg_content)
    return all_hexcodes

def calculate_average_color(combination, base_hexcodes):
    total_r, total_g, total_b = 0, 0, 0
    num_sets = sum(combination)
    for i, present in enumerate(combination):
        if present:
            color = base_hexcodes[i]
            r, g, b = int(color[1:3], 16), int(color[3:5], 16), int(color[5:7], 16)
            total_r += r
            total_g += g
            total_b += b
    average_r = total_r // num_sets
    average_g = total_g // num_sets
    average_b = total_b // num_sets
    average_color = f'#{average_r:02X}{average_g:02X}{average_b:02X}'
    return average_color

def replace_all_hexcodes(svg_content, new_hexcodes):
    # Regular expression pattern to find hexcodes in fill and stroke attributes
    pattern = r'(#(?:[A-Fa-f0-9]{6})|[A-Fa-f0-9]{3}))'
    def replace_func(match):
        return new_hexcodes.pop(0)
    new_svg_content = re.sub(pattern, replace_func, svg_content)
    return new_svg_content

def main():
    input_svg_file = "input.svg" # Replace with your input SVG file path
    base_hexcodes = [ '#FF0000', # C1
                      '#FFAA00', # C2
                      '#FFFF00', # C3
                      '#00FF00', # C4
                      '#00B1FF', # C5
                      '#8000FF', # C6
                      '#FF00CC', # C7 ]
    # Read the SVG file with open(input_svg_file, "r") as file: svg_content = file.read()
    # Extract all the hexcodes from the SVG
    extracted_hexcodes = extract_all_hexcodes(svg_content)
    # Ensure the number of extracted hexcodes matches the number of sets in the SVG
    if len(extracted_hexcodes) != len(base_hexcodes):
        print("Error: The number of extracted hexcodes does not match the number of base hexcodes.")
        return
    # Generate all combinations of sets
    combinations_list = list(combinations(range(1, len(base_hexcodes) + 1), 1)) + \
        list(combinations(range(1, len(base_hexcodes) + 1), 2)) + \
        list(combinations(range(1, len(base_hexcodes) + 1), 3)) + \
        list(combinations(range(1, len(base_hexcodes) + 1), 4)) + \
        list(combinations(range(1, len(base_hexcodes) + 1), 5)) + \
        list(combinations(range(1, len(base_hexcodes) + 1), 6)) + \
        list(combinations(range(1, len(base_hexcodes) + 1), 7))
    # Calculate average color for each section and create a list of average hexcodes
    average_hexcodes = []
    for combination in combinations_list:
        binary_combination = [1 if i+1 in combination else 0 for i in range(len(base_hexcodes))]
        average_color = calculate_average_color(binary_combination, base_hexcodes)
        average_hexcodes.append(average_color)
    # Replace all the hexcodes in the SVG with the calculated average hexcodes
    new_svg_content = replace_all_hexcodes(svg_content, average_hexcodes)
    # Write the updated SVG content back to the file with
    open(input_svg_file, "w") as file:
        file.write(new_svg_content)
    print("Hexcodes replaced in the SVG file.")

if __name__ == "__main__":
    main()

This updated program uses the extract_all_hexcodes function to extract all the hexcodes from both fill and stroke attributes of the input SVG file. It then calculates the average color for each combination of sets and replaces all the hexcodes in the SVG with the calculated average colors using the replace_all_hexcodes function.

The program reads the SVG content from the input file, extracts the hexcodes, and replaces them with the calculated average colors. The input_svg_file variable should be set to the path of your input SVG file.
```