Making a Logo in Scratch

Scratch is a visual programming platform, so you'll be making a logo using its drawing tools or by coding sprite costumes.

Steps:

1. Open Scratch

   Go to scratch.mit.edu and create a new project.

2. Delete the Cat Sprite (Optional)

   Right-click the cat sprite → choose delete if you want a blank workspace.

3. Add a New Sprite for Your Logo
   - Click the paintbrush icon under "Choose a Sprite" → select Paint.
   - This opens the Scratch paint editor.

4. Design Your Logo
   - Use the shapes tool (circle, rectangle, line) to create simple graphics.
   - Use the text tool to add letters or initials.
   - Change colors, outlines, and fills to make it stand out.

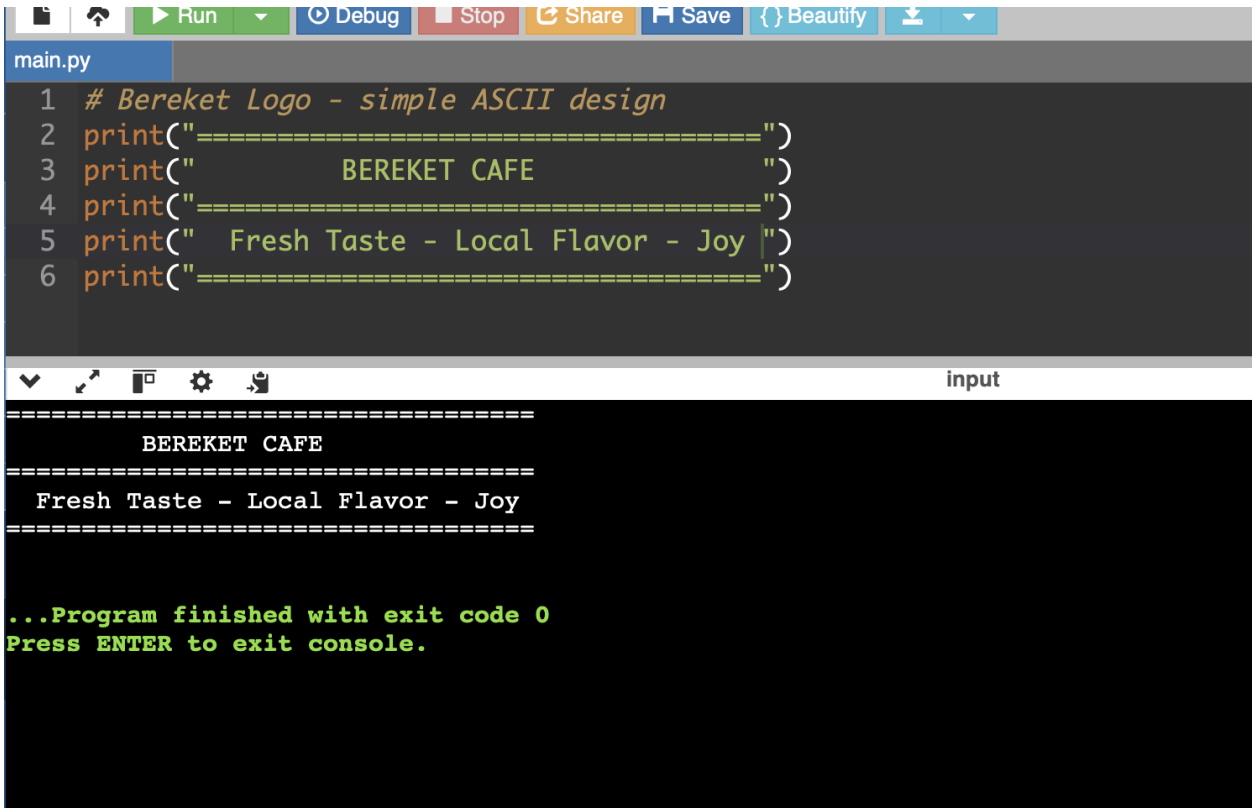
5. Adjust Size and Position
   - Drag your logo sprite to where you want it.
   - Use the Size field in the sprite menu to make it bigger or smaller.

6. Add Effects with Code (Optional)
   - For animation, click the Code tab and add blocks like:
     - "when green flag clicked → forever → turn 15 degrees" (spins logo).
     - "when green flag clicked → change color effect by 25" (color shifting).

7. Save & Share
    - Click File → Save now.
    - Share if you want others to see your logo.

Making a Logo in OnlineGDB

EASY LOGO: Print Statement



OnlineGDB is mainly a coding IDE, so here you'll create a logo using ASCII art or by writing a program to print a text-based logo.

A.) Static ASCII (Quick and simpler)

Steps:

1. Open OnlineGDB

Go to onlinegdb.com → choose Python as your language.

2. Clear the Starter Code

   Delete the template main function code (if any).

3. Write Your Logo Code
4. Edit the block between R"(...)" to your design. Use #, @, *, _ and spaces; keep a single-spaced font.

```
#include <iostream>
int main() {
    std::cout << R"(
  __  __  _____   _____  _______          _
 |  \/  |/ __ \ / ____|/ / ____| /\    | |
 | \  / | |  | | |      | | |     /  \   | | ___
 | |\/| | |  | | |      | | |    / /\ \  | |/ _ \
 | |  | | |__| | |____ | | |___/ ____ \| |  __/
 |_|  |_|\____/ \_____||_|\____/_/    \_\_|\___|
    )" << '\n';
    return 0;
}
```

B.) Generated Block-letter Text (Reusable)

1. Pick a grid size

- Use at least 5×5 (better readability).
- Larger (7×7, 8×8) = smoother curves and diagonals.
- Each row is a string of # (filled pixel) and spaces (empty pixel).

Example for “A” in 5×5:

```
  ###
 #   #
 #####
 #   #
 #   #
```

2. Store each glyph in a dictionary

In Python:

```
FONT = {
 'A': [" ### ",
       "#   #",
       "#####",
       "#   #",
       "#   #"],
 # add more letters here
}
```

3. Write a render function

- Loop through the text, grab each letter's glyph.
- Stitch together row by row.
- Replace # with any symbol you like (█, *, @, etc.).
- Add a space or two between letters for clarity.

```
def render(text, ch="█", space=2):
    rows = [""] * 5
    for c in text.upper():
        glyph = FONT.get(c, FONT[' '])
        g = [row.replace("#", ch) for row in glyph]
        for i in range(5):
            rows[i] += g[i] + " " * space
    print("\n".join(rows))
```

4. Scale if you want bigger text

- Multiply each "pixel" by scale.
- Example: scale=2 → each █ becomes a 2×2 square.
- This keeps proportions but makes the text bolder.

5. Output in a monospaced font

- Use Courier, Consolas, or Scratch's default monitor font.
- If you paste into a non-monospaced environment, spacing will break.

6. Test & tweak

- Experiment with different characters:
    - ▉ = solid, bold.
    - # = standard block.
    - * = light texture.
    - Adjust space to avoid letters looking squished.

Full Example:

```
FONT = {
 'M': ["# #",
       "###",
       "# #"],
 'O': ["###",
       "# #",
       "###"],
 'S': ["###",
       "## ",
       "###"],
 'T': ["###",
       " # ",
       " # "],
 'A': ["###",
       "# #",
       "# #"],
 'F': ["###",
       "## ",
       "#  "],
 ' ': ["   ",
       "   ",
       "   "]
}

def render(text, ch="#", space=1):
    rows = ["", "", ""]
    for c in text.upper():
        g = FONT.get(c, FONT[' '])
        g = [row.replace("#", ch) for row in g]
        for i in range(3):
            rows[i] += g[i] + " " * space
    print("\n".join(rows))

render("MOSTAFA", ch="#", space=1)
```

7. Print Statement

Finally, in order to display your slogan, all you need is a render statement that will print and show off your slogan! Essentially, the render function stitches the rows of each letter together and prints them out as ASCII art. Below is an example with the letters H, E, Y so you can copy, paste, and run it directly in any Python compiler!

```
FONT = {
 'H': ["█   █",
       "█   █",
       "█████",
       "█   █",
       "█   █"],
 'E': ["█████",
       "█    ",
       "████ ",
       "█    ",
       "█████"],
 'Y': ["█   █",
       " █ █ ",
       "  █  ",
       "  █  ",
       "  █  "],
 ' ': ["     ",
       "     ",
       "     ",
       "     ",
       "     "]
}

def render(text, ch="█", space=2):
    rows = [""] * 5
    for c in text.upper():
        glyph = FONT.get(c, FONT[' '])
        g = [row.replace("█", ch) for row in glyph]
        for i in range(5):
            rows[i] += g[i] + " " * space
    print("\n".join(rows))

# Change the word here:
render("HEY", ch="▓")
```

The render statement is a key part of the entire code, and if you look closely, it is used in the other full examples as well!