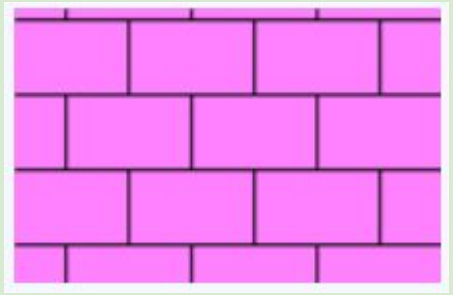
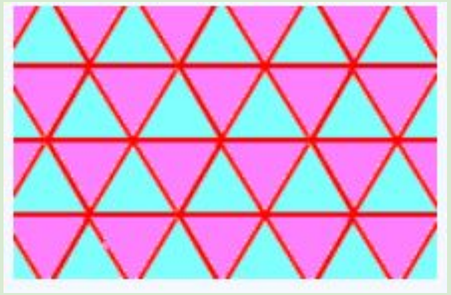
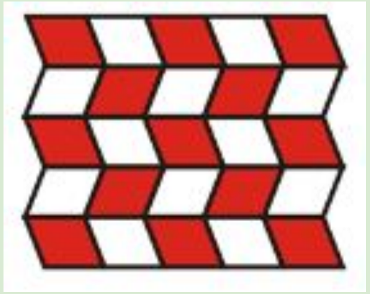
Tessellations

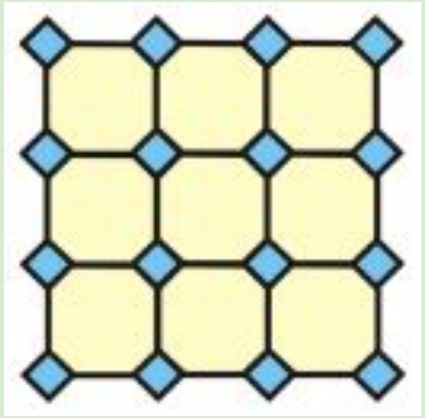
Cross-curricular mission for CodeX

Tessellations

- A tessellation is a tiling over a surface with one or more figures such that the surface is filled with no overlaps and no gaps.

Mission: Tessellations



For this mission you will:

- Create a one-shape tessellation
- Create a two-shape tessellation
- Create a triangle tessellation
- Discuss some math that goes into a tessellation
- Create your own tessellation

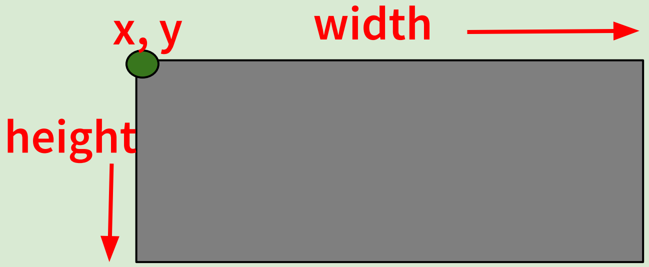
Shape #1: A brick

A tessellation can be made with a single rectangle.

- Codex uses these functions:

```
display.fill_rect(x, y, width, height, RED)
display.draw_rect(x, y, width, height, WHITE)
```

- x, y are the location (column, row) of the upper left corner
- width of the rectangle is how far across in pixels
- height of the rectangle is how far down in pixels



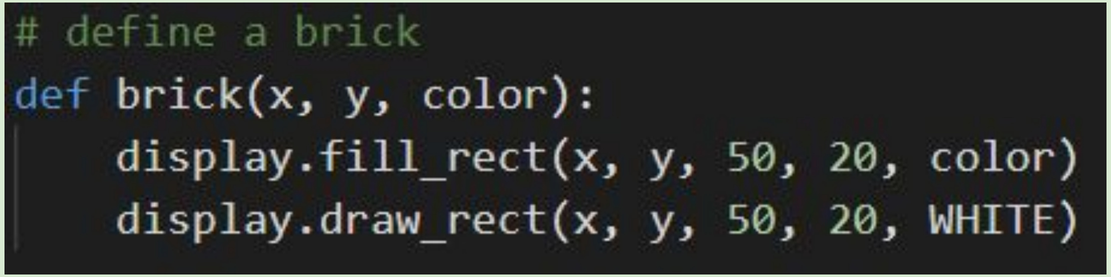
Shape #1: A brick

A tessellation can be made with a single rectangle.

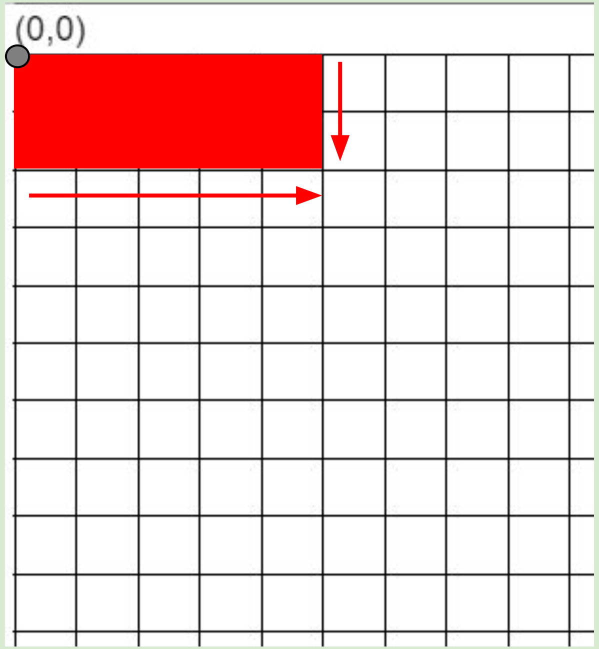
- Create a file named **Tessellations**
- Copy and paste the code for tessellations
- Run the code to make sure it works without errors
- Look at the code to see how the row of bricks is created

Shape #1: A brick

This function draws a rectangle at location (x, y) that is 50 pixels wide & 20 pixels high.



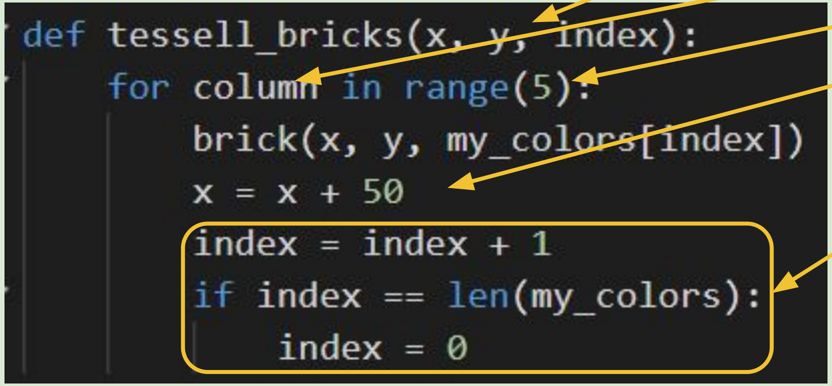
```
# define a brick
def brick(x, y, color):
    display.fill_rect(x, y, 50, 20, color)
    display.draw_rect(x, y, 50, 20, WHITE)
```



Tessellating the brick

A tessellation is filling a surface with a shape, so we need more than one brick.

- This loop draws one row of bricks.



```
def tessell_bricks(x, y, index):
    for column in range(5):
        brick(x, y, my_colors[index])
        x = x + 50
        index = index + 1
        if index == len(my_colors):
            index = 0
```

Initial values for x, y and index (passed into parameters)

Loop counter

How many loops (bricks)

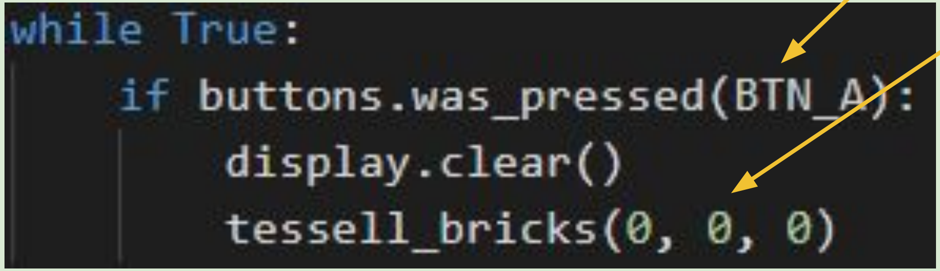
Change x by the **width** of the brick

Change the brick color using the index and the list of colors

Tessellating the brick

A tessellation is filling a surface with a shape, so we need more than one brick.

- Calling the tessellation function:



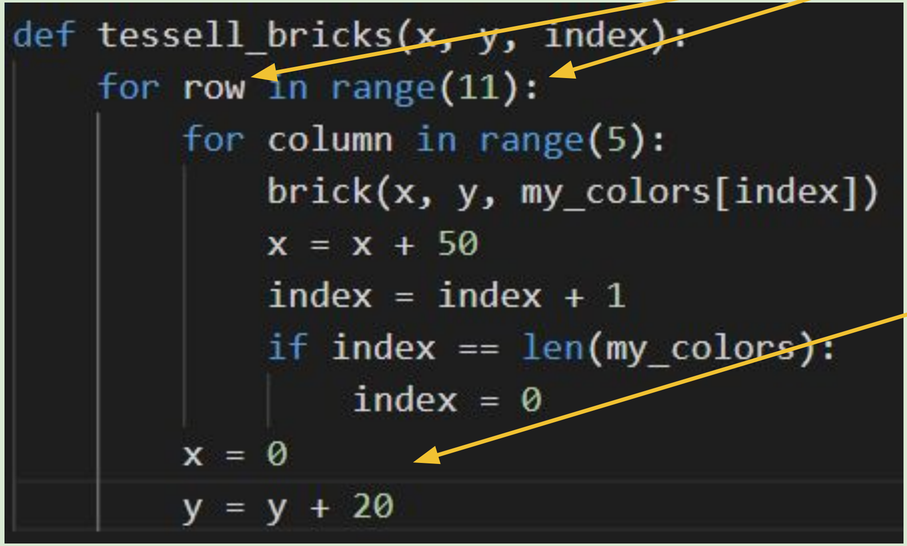
```
while True:
    if buttons.was_pressed(BTN_A):
        display.clear()
        tessell_bricks(0, 0, 0)
```

Call the function when Button A is pressed

Initial values of x, y and index

Tessellating the brick

- Add a second loop to get multiple rows of bricks:



```
def tessell_bricks(x, y, index):
    for row in range(11):
        for column in range(5):
            brick(x, y, my_colors[index])
            x = x + 50
            index = index + 1
            if index == len(my_colors):
                index = 0
        x = 0
        y = y + 20
```

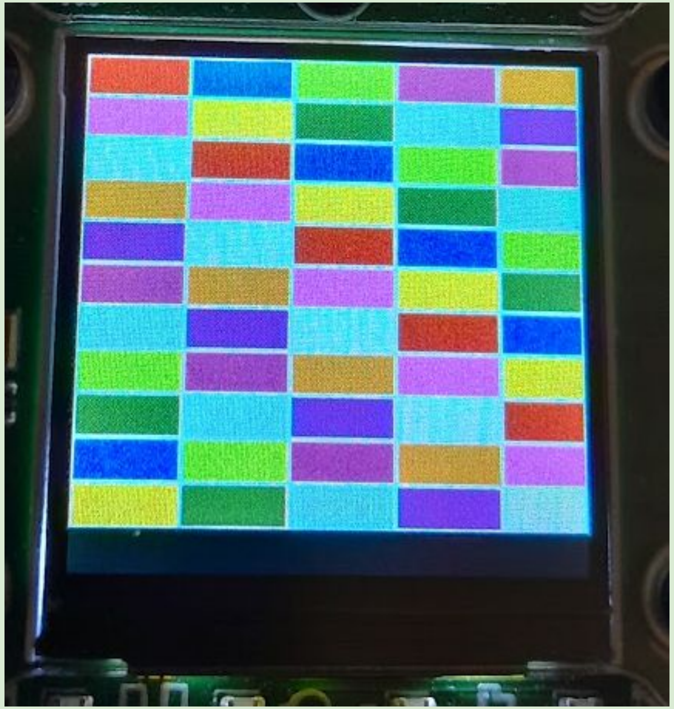
For each row, use a different loop counter and number of loops.

Watch the indenting – you have a loop within a loop (nested loops)

For each new row, reset x to 0 and change y by the **height** of the brick.

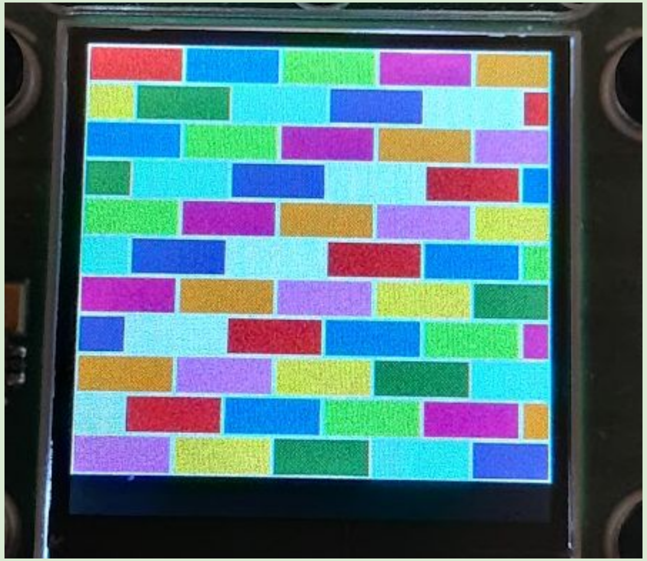
Tessellating the brick

This is good, but not very interesting.



Tessellating the brick

- We want the tessellation to look more like actual bricks
- One row is a regular row
- The next row is offset by half the width
- Then a regular row
- Offset row
- And so forth…



Tessellating the brick

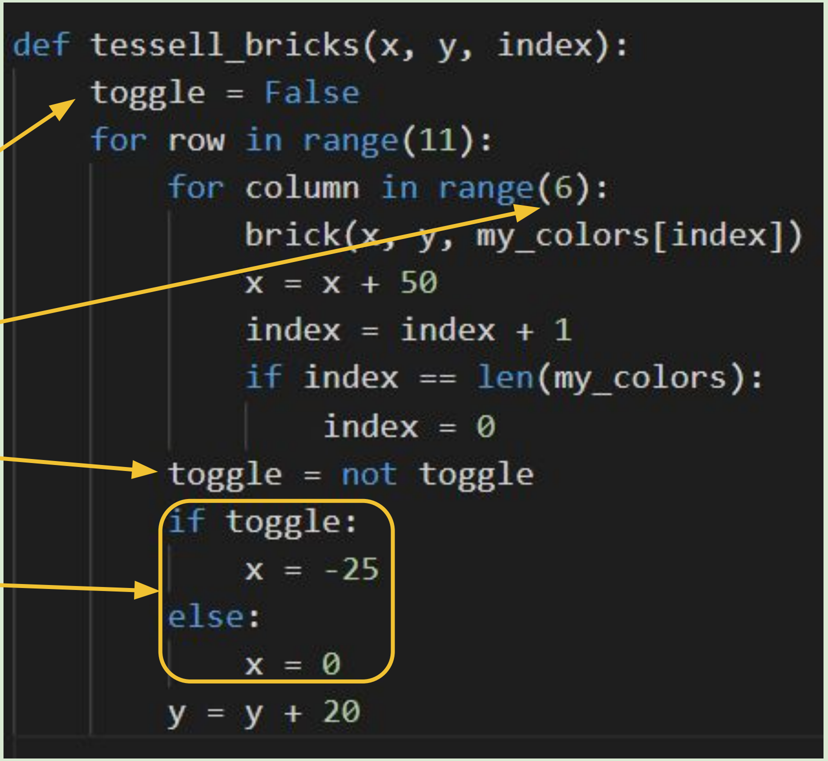
- Use a variable to “toggle” between regular and offset

Use a Boolean variable to toggle

You may need to change the # of loops

After each row of bricks, change the toggle

Determine the starting value of x depending on the toggle

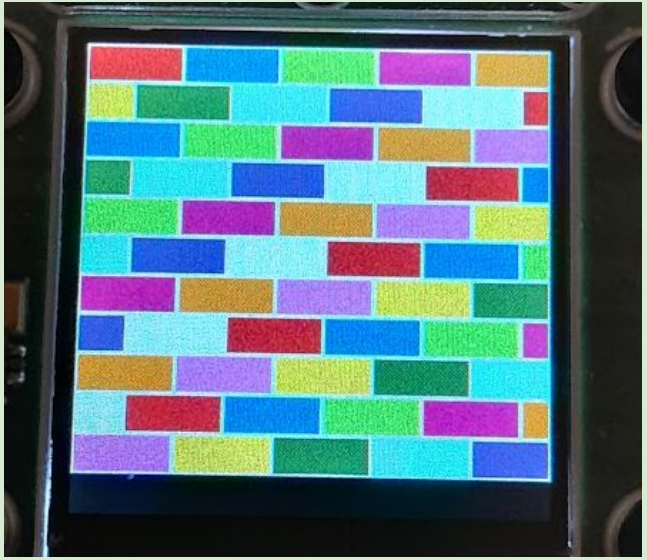


```
def tessell_bricks(x, y, index):
    toggle = False
    for row in range(11):
        for column in range(6):
            brick(x, y, my_colors[index])
            x = x + 50
            index = index + 1
            if index == len(my_colors):
                index = 0
        toggle = not toggle
        if toggle:
            x = -25
        else:
            x = 0
        y = y + 20
```

Tessellating the brick

Great! You have your first tessellation!

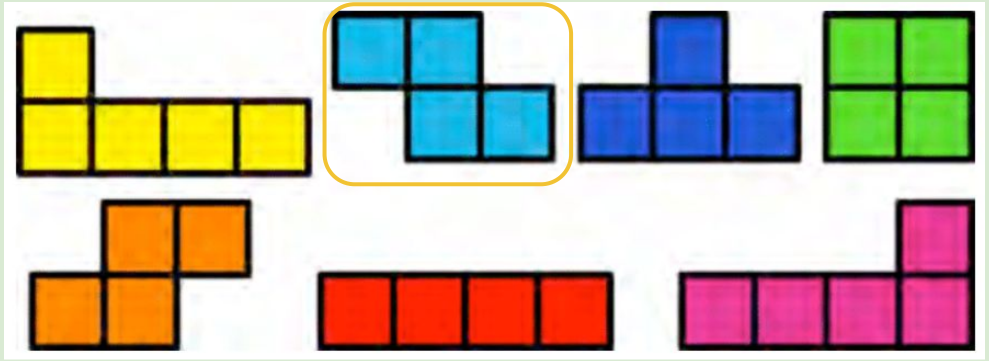
- Review the steps
- You will be repeating them for the next tessellations



Shape #2 - A tetris shape

Have you noticed the shapes in Tetris? They fit together, kind of like a tessellation.

- The light blue shape can be created by putting four squares – or two bricks (rectangles) – together, with one brick offset by the width of the brick.

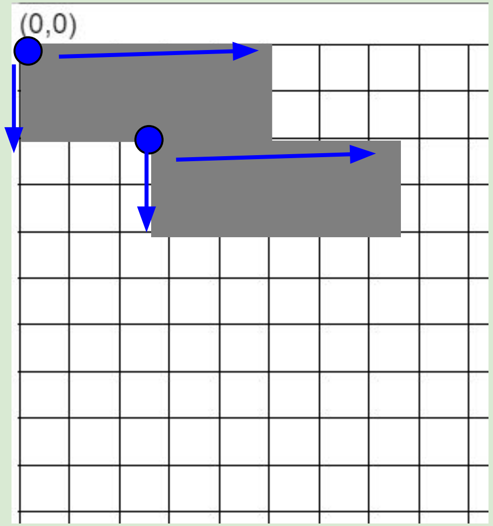


Shape #2 - Skew

Look at the function for “skew”

It is very similar to “brick” but with two rectangles

- Use graph paper to determine the x and y locations for each rectangle

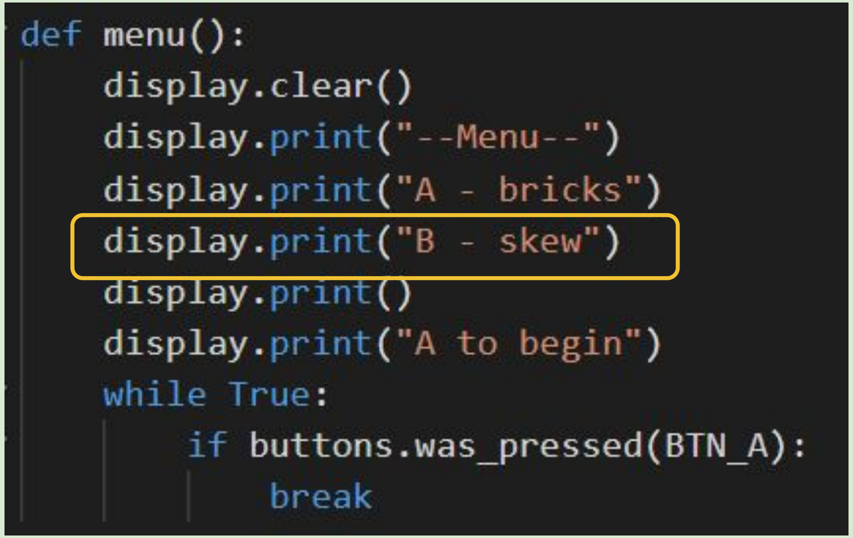


First rectangle:
(0, 0, 50, 20)
Second rectangle:
(25, 20, 50, 20)
Using x and y for both locations:
(x, y, 50, 20, color)
(x+25, y+20, 50, 20, color)

Shape #2 - Skew

Add to the menu

- You are working on the second tessellation
- Include a display.print() statement in the menu

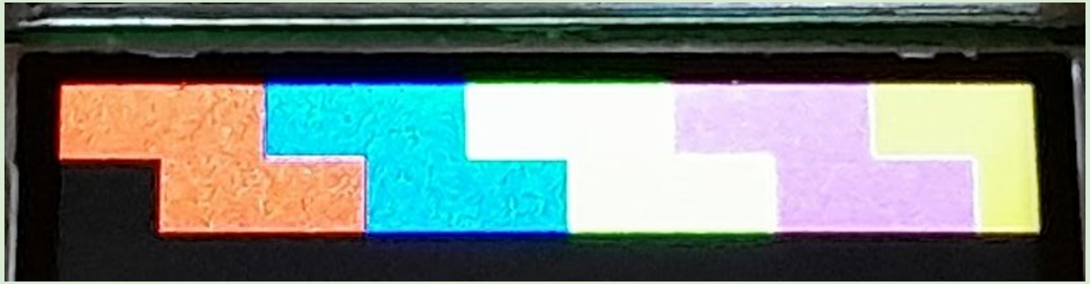


```
def menu():
    display.clear()
    display.print("--Menu--")
    display.print("A - bricks")
    display.print("B - skew")
    display.print()
    display.print("A to begin")
    while True:
        if buttons.was_pressed(BTN_A):
            break
```

Shape #2: Skew

Look at the function for tessellating the skew.

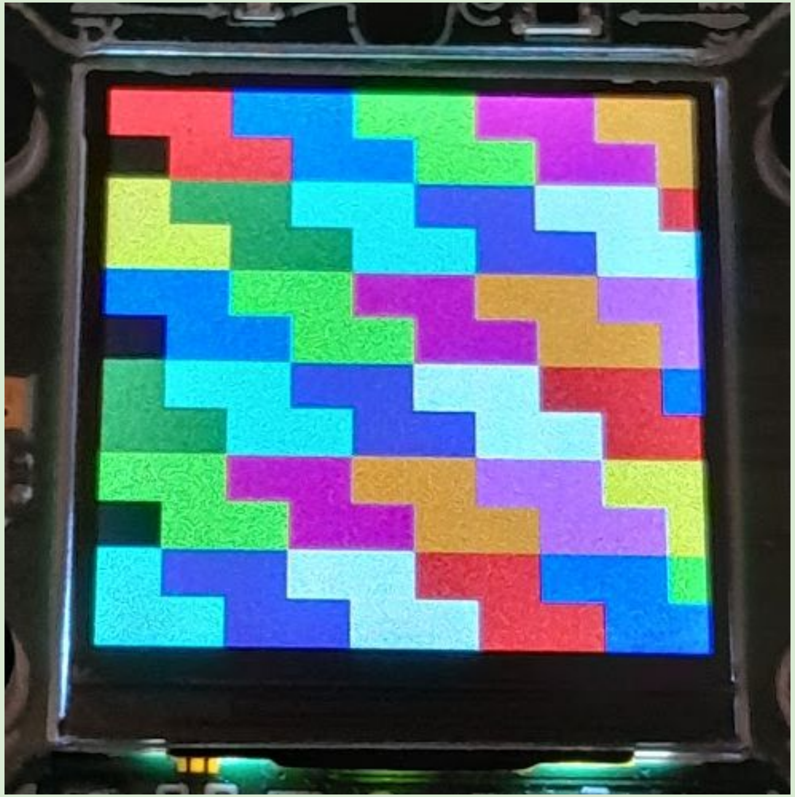
- Add a loop to draw a single row.
- It will be very similar to the brick, so you can even copy and paste the code and change the function call.
- Try different loop numbers until you get just the right number.



Shape #2: Skew

Make the tessellation

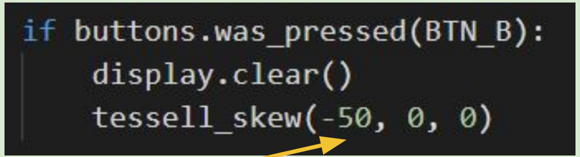
- Add another loop to make multiple rows
- Use a toggle variable, like the tessell_brick() function
- After each column loop, reset x using toggle, and change y
- *NOTE: y height is 40, not 20!*



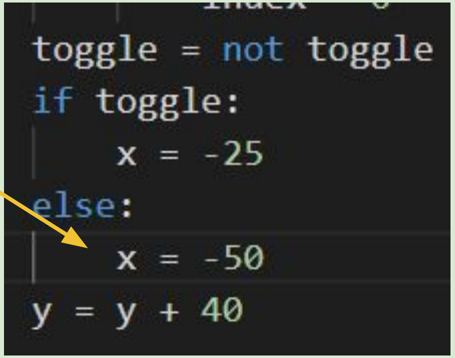
Shape #2: Skew

You may notice there is gap on the lower left of each regular row.

- To fix it, we can offset each regular row by the width of the first rectangle
- Increase the loop count if needed



```
if buttons.was_pressed(BTN_B):
    display.clear()
    tessell_skew(-50, 0, 0)
```

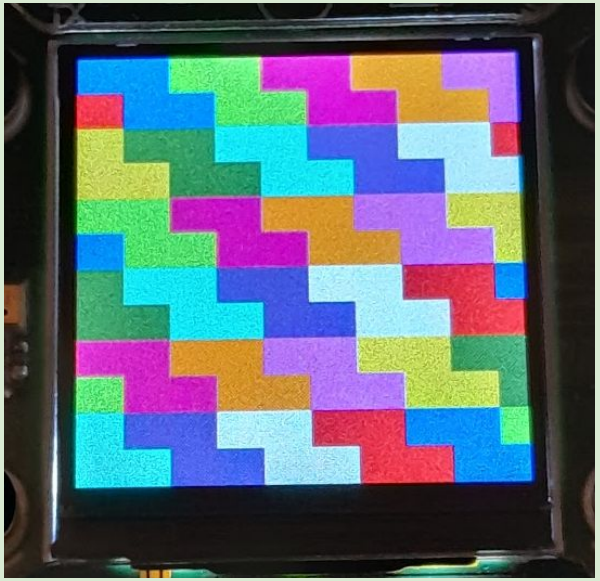


```
toggle = not toggle
if toggle:
    x = -25
else:
    x = -50
y = y + 40
```

Tessellating the skew

Great! You have your second tessellation!

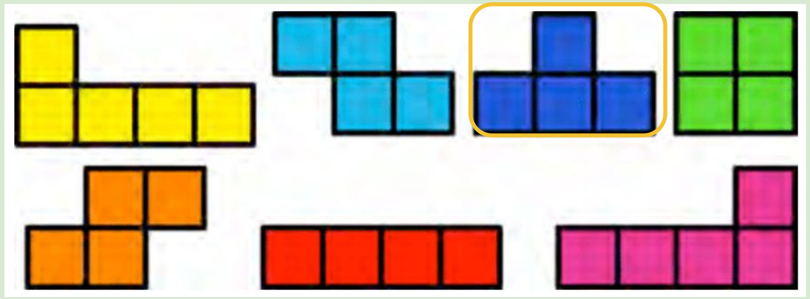
- Review the steps
- You will be repeating them for the next tessellations

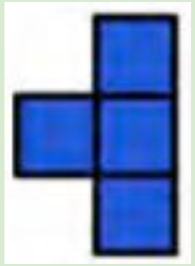


Shape #3: Half-cross

Try another Tetris shape.

- Use graph paper to design your shape.
- Rotate the shape, as shown:

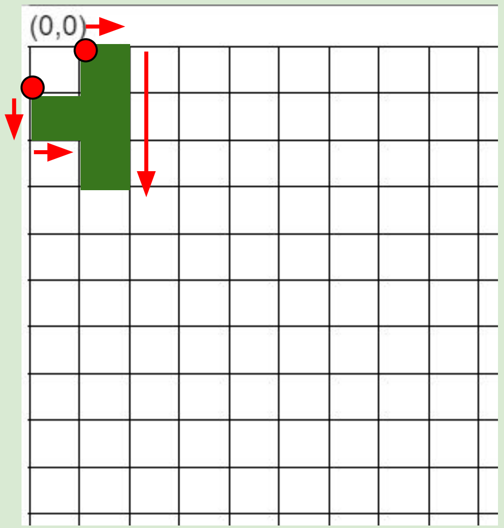


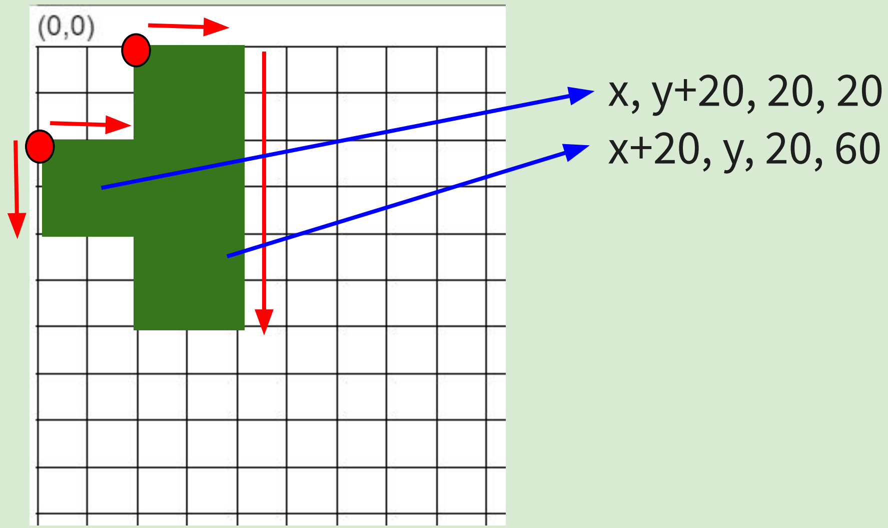


Shape #3: Half-cross

Call this Tetris shape "halfcross".

- You can do a small or large halfcross (use two rectangles)
- The function in the program is for the larger halfcross

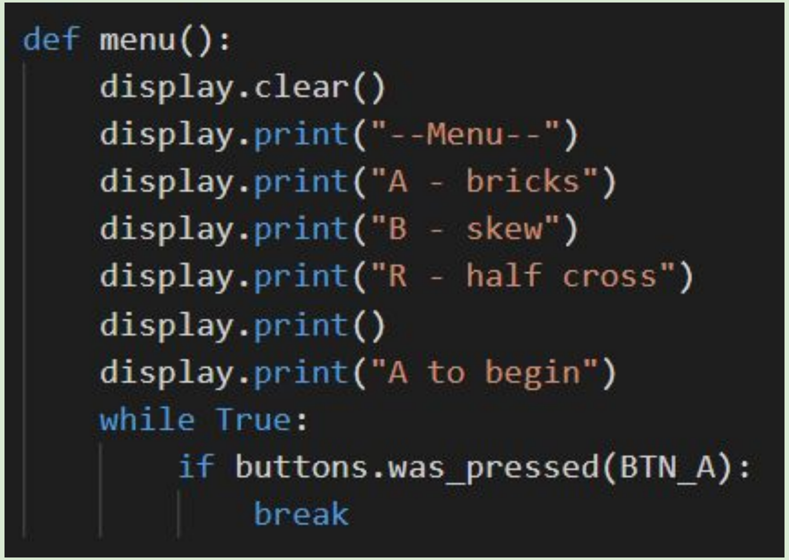




Shape #3: Half-cross

Draw a row of the halfcross shape.

- Look at the function for halfcross and understand the code.
- Add a display.print() statement to menu for the half cross shape.

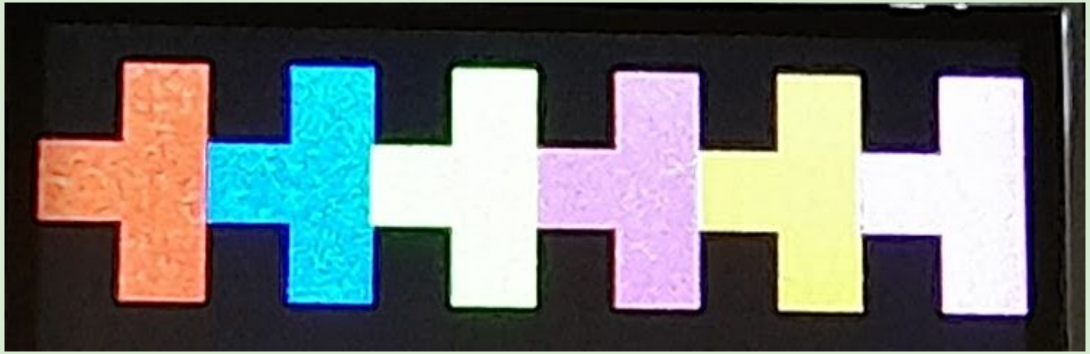


```
def menu():
    display.clear()
    display.print("--Menu--")
    display.print("A - bricks")
    display.print("B - skew")
    display.print("R - half cross")
    display.print()
    display.print("A to begin")
    while True:
        if buttons.was_pressed(BTN_A):
            break
```

Shape #3: Half-cross

Draw a row of halfcross shapes.

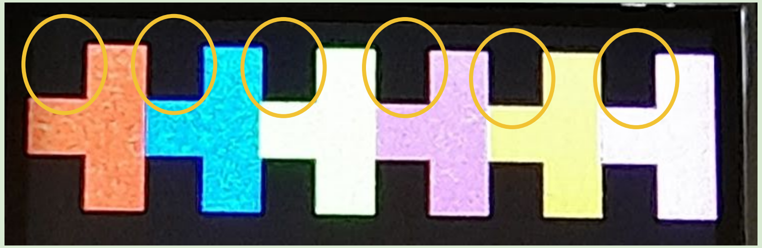
- Add a loop to create one row of shapes.
- *NOTE: the x width is different than the first two shapes*

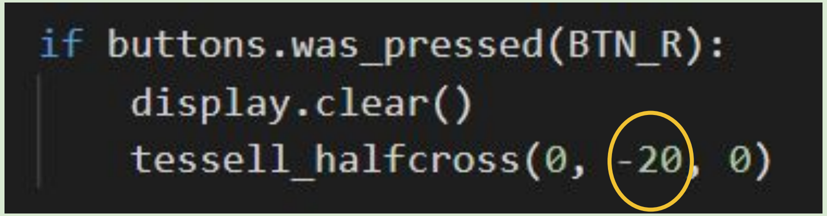


Shape #3: Half-cross

Draw a row of halfcross shapes.

- You (again) notice a gap in the shapes.
- This time it is in the upper left corner
- Offset y this time



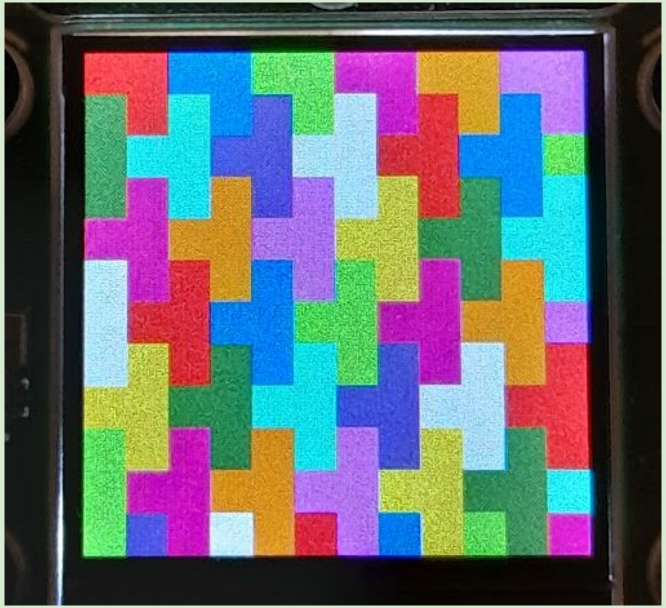


```
if buttons.was_pressed(BTN_R):
    display.clear()
    tessell_halfcross(0, -20, 0)
```

Shape #3: Half-cross

Draw a tessellation of halfcross shapes.

- Complete the tessellation by adding the second loop and the toggle.
- Adjust the loop counts as needed until your tessellation fills the screen without gaps or overlaps.



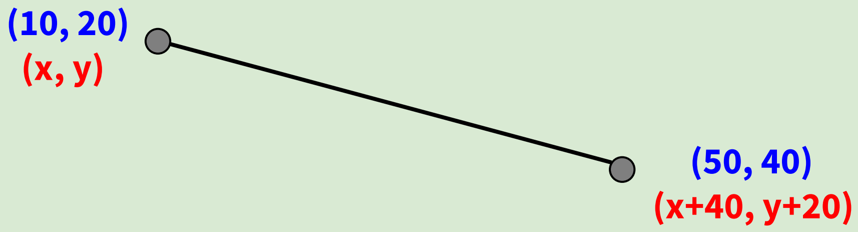
Shape #4: Isosceles triangle

CodeX doesn't have a built-in function that draws a triangle.

- Use the display.draw_line() function to draw three lines.

```
display.draw_line(x1, y1, x2, y2, color)
```

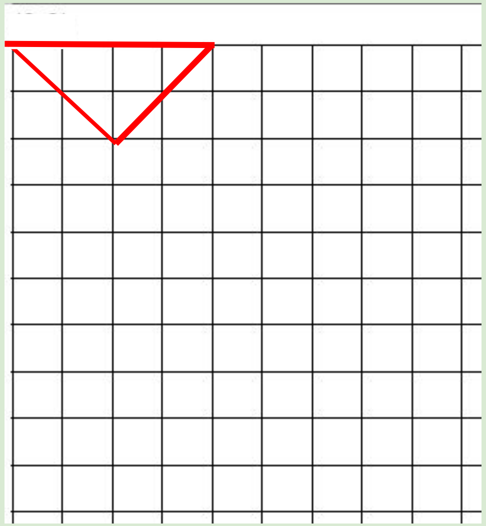
- x1, y1 are the location (column, row) of the first line end
- x2, y2 are the location (column, row) of the second line end
- x2 & y2 can be defined in relation to x1 & y1

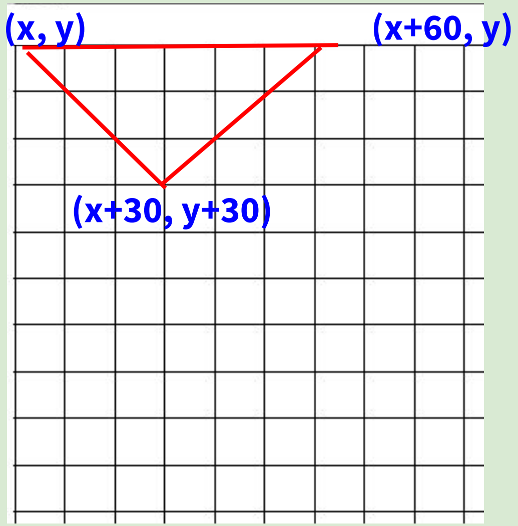


Shape #4: Isosceles triangle

Use graph paper to design an isosceles triangle.

- Determine the size of your triangle
- Determine the x1, y1 and x2, y2 locations



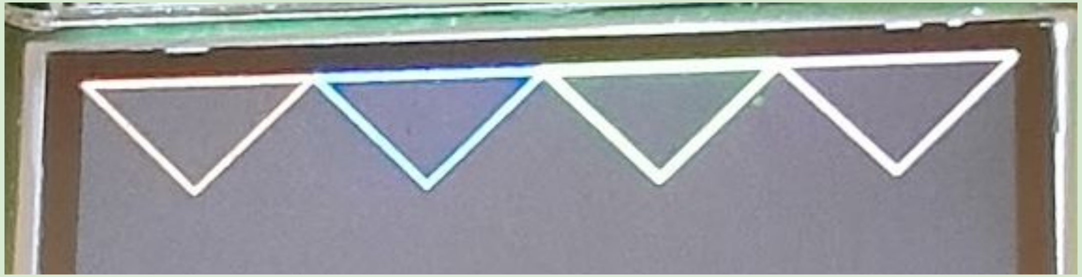


- Your triangle can be any size
- This program uses the larger triangle

Shape #4: Isosceles triangle

Create a function to draw the triangle.

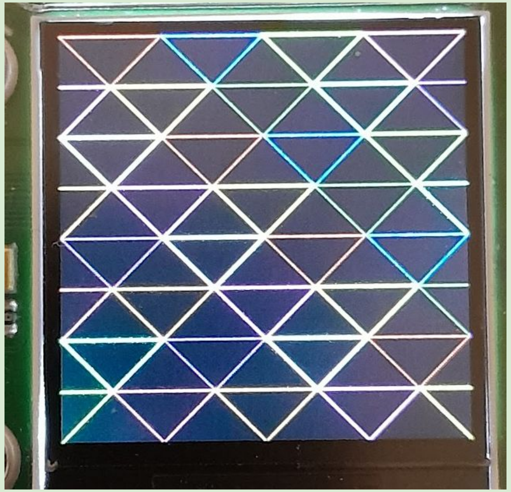
- Use the information from your graph to draw three lines to form a triangle.
- The function has been included in your starter code.
- Add a display.print() to the menu.
- Add a loop to the tessell_triangle() function to draw one row of triangles.



Shape #4: Isosceles triangle

Create a tessellation of the triangle.

- Add another loop and toggle to create the triangle tessellation.
- Looks good, right?
- But if you look closely, you actually have a lot of gaps. You are only drawing every other triangle.
- You need a flipped triangle!



Shape #4: Isosceles triangle

Add the code for a flipped triangle

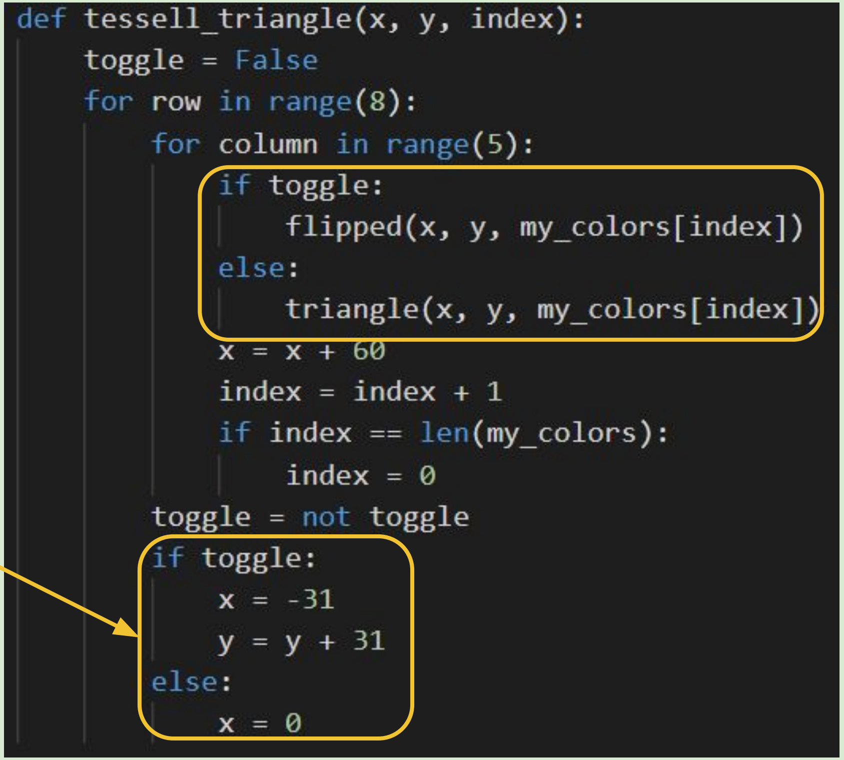
- The starter code includes a starter function for a flipped triangle.
- What will be flipped?
- Instead of the y values going down (adding), they will go up (subtracting)
- Add three lines of code to flipped() to draw a flipped triangle
- NOTE: the x values will stay the same; just change the “y + “ to “y – “

Shape #4: Isosceles triangle

Complete the tessellation

- Every other row will need to be flipped
- You already have a toggle variable
- Use it to determine which row of triangles to draw

  - Make adjustments here to see the triangles
  - Offset by 1 pixel to see the triangles better
  - Thought question: why change y only if toggle is True, and not when False?

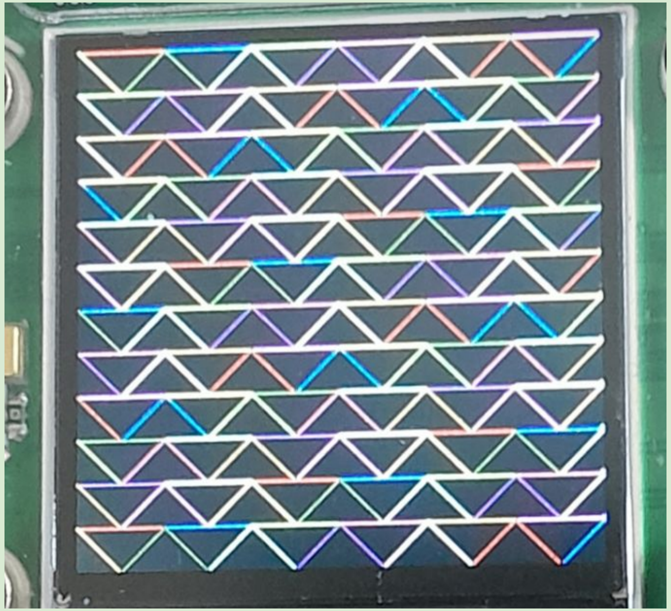


```
def tessell_triangle(x, y, index):
    toggle = False
    for row in range(8):
        for column in range(5):
            if toggle:
                flipped(x, y, my_colors[index])
            else:
                triangle(x, y, my_colors[index])
            x = x + 60
            index = index + 1
            if index == len(my_colors):
                index = 0
        toggle = not toggle
        if toggle:
            x = -31
            y = y + 31
        else:
            x = 0
```

Tessellating the triangle

Great! You have your fourth tessellation!

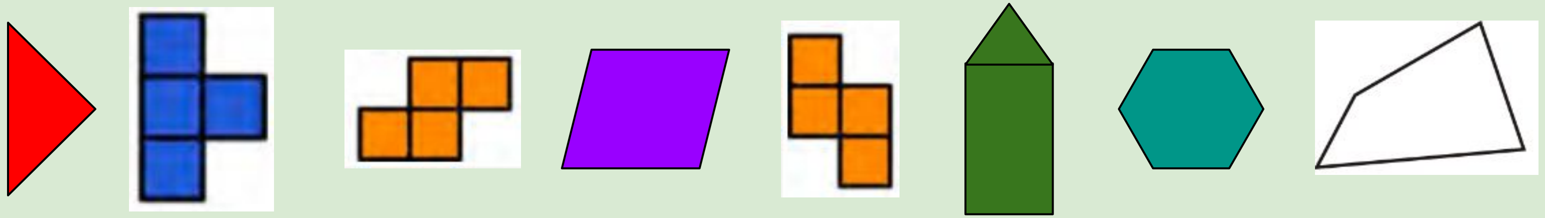
- Are you ready to try your own?



Shape #5: Your own shape

Decide on a shape to tessellate.

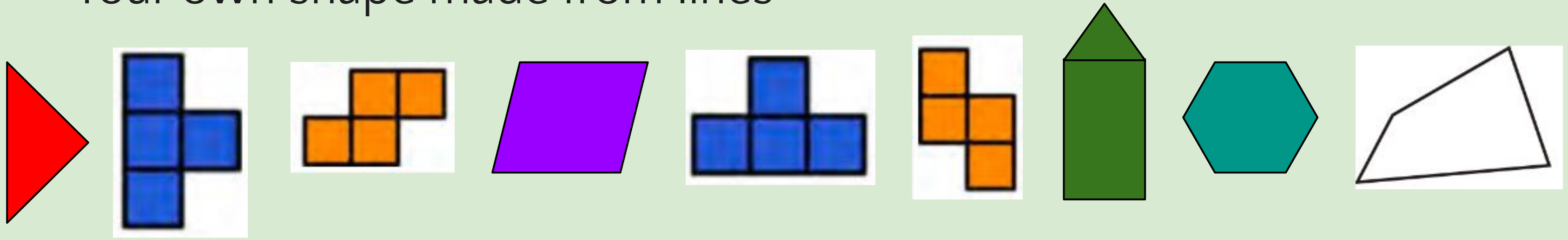
- Use graph paper to design your shape.
- Make sure your shape is able to tessellate –
  - It needs to fill the surface without gaps or overlaps
  - Your shape could require a flip or rotation



Shape #5: Your own shape

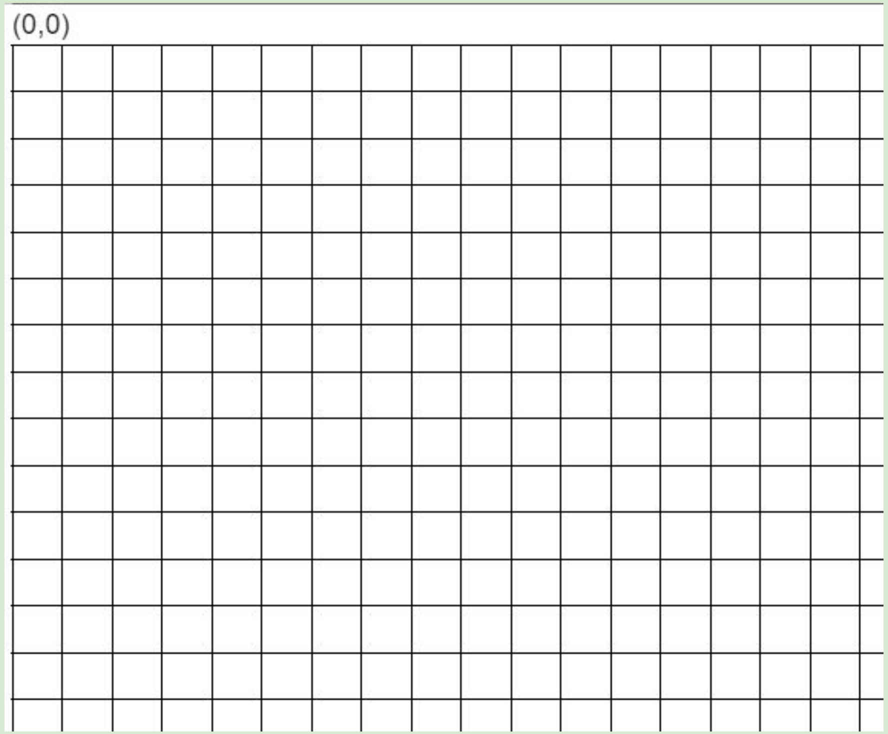
Decide on a shape to tessellate.

- Possibilities:
  - An isosceles triangle with a straight side
  - A half cross facing a different direction
  - A different Tetris shape
  - Your own shape made from rectangles
  - Your own shape made from lines



Shape #5: Your own shape

Use graph paper to design your own tessellation shape.



Shape #5: Your own shape

Write the code.

- Add code to the my_shape() function to draw your own shape
- Add a display.print() statement to menu()
- Run the code as written to see one shape
- When it looks the way you want it to, add code to tessell_my_shape() to draw a single row of your shape
- When the row looks like a tessellation row, complete the code by adding another loop and toggle
- Test and debug as needed

Tessellation Extensions

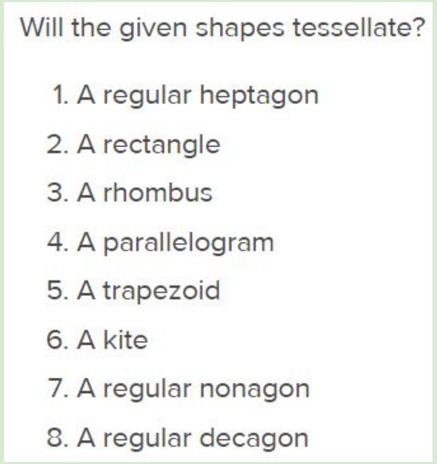
Geometry emphasis:

- Scaling – use graph paper to draw the shape and then code it on CodeX. Measure it on the shape on the Codex and calculate the scaling. Use different types or sizes of graph paper to demonstrate different scales of the shape and calcuate the scaling.

Tessellation Extensions

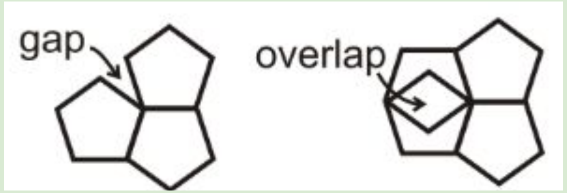
Geometry emphasis:

- Tessellation shapes – review the requirements for a tessellation. Have students look at several different shapes and determine if they tessellate. If so, what would the configuration of shapes look like? This website gives a good explanation of the math involved in a tessellation.



Will the given shapes tessellate?

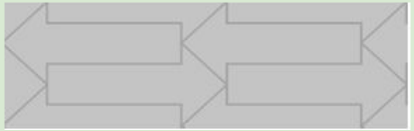
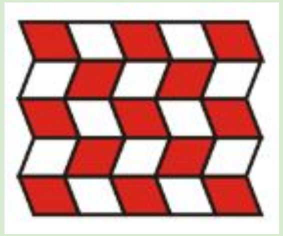
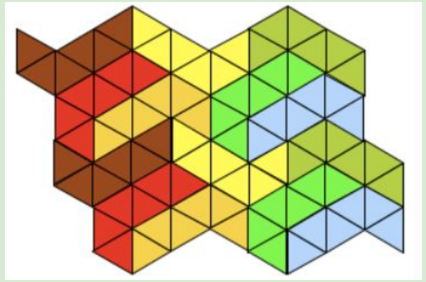
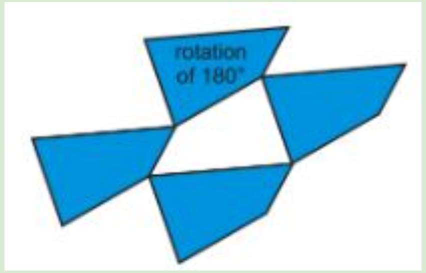
1. A regular heptagon
2. A rectangle
3. A rhombus
4. A parallelogram
5. A trapezoid
6. A kite
7. A regular nonagon
8. A regular decagon



Tessellation Extensions

Geometry emphasis:

- Flip and rotate – create a tessellation that requires a flipped shape, or a tessellation that requires a rotation of the shape. Show the math for the flip / rotation.
- This can be done on CodeX, or with a paper model.

Tessellation Extensions

Algebra emphasis:

- Focus on the math used to create the shape on a Cartesian graph, in relation to x and y. Use the graph to rotate or flip the shape and show the math of how to go from one shape to another.
- As a challenge, try writing code for a tessellation with a flipped shape without creating a second function for flipped, but just using math.

Tessellation Extensions

Art emphasis:

- Create a shape and use a color palette that sets a mood. Use RGB colors to specify the colors instead of using the built-in colors.
- Use this slide deck for more information about RGB colors and CodeX.

Tessellation Extensions

Art emphasis:

- Use two different color lists – one for the regular row and one for the offset row.
- Options for lists:
  - One of primary colors and one of secondary colors
  - One list that has shades of one color, and the other list has shades of a different color
  - Use only two or four colors
  - Compare a tessellation with a lot of color with a tessellation that is black and white