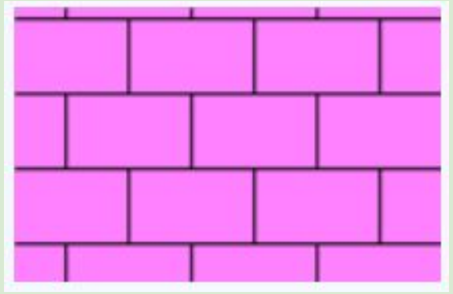
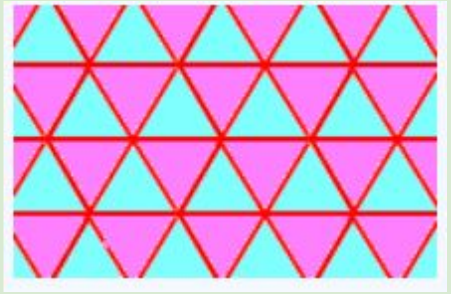
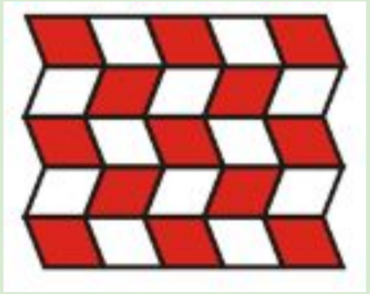
Tessellations

Cross-curricular mission for CodeX

Tessellations

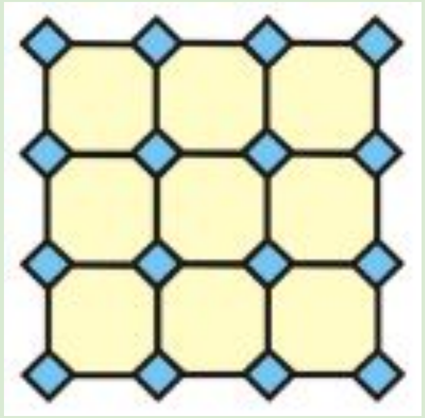
- A tessellation is a tiling over a surface with one or more figures such that the surface is filled with no overlaps and no gaps.

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



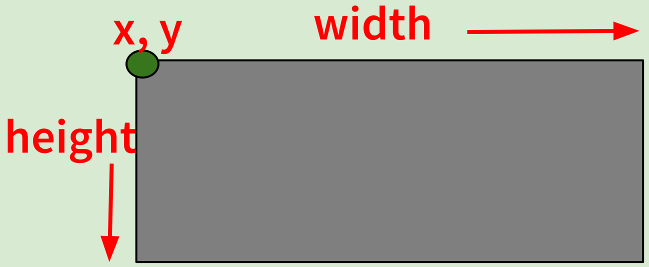
Shape #1: A brick

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

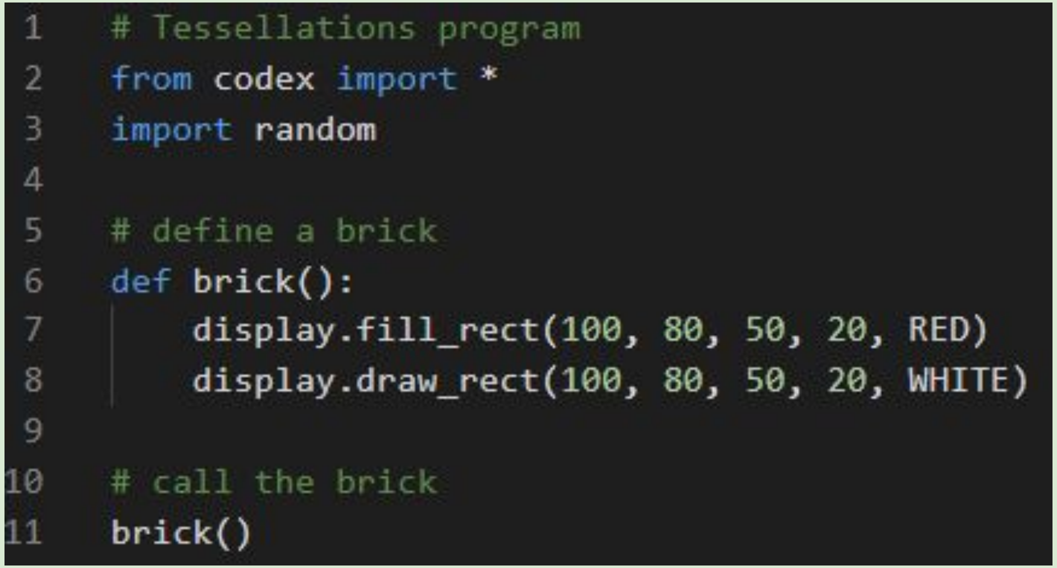
A tessellation can be made with a single rectangle.

- Create a file named **Tessellations**
- Import the codex and random modules
- Define a function that draws a brick
- Call the function
- Run the code

Shape #1: A brick

This function draws a rectangle at (100, 80) that is 50 pixels wide & 20 long.

- Try different numbers for x, y, width and height.

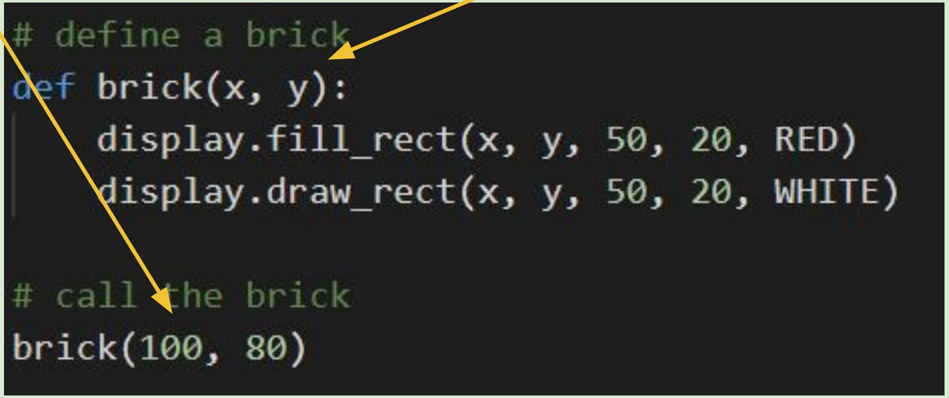


```python
# Tessellations program
from codex import *
import random

# define a brick
def brick():
    display.fill_rect(100, 80, 50, 20, RED)
    display.draw_rect(100, 80, 50, 20, WHITE)

# call the brick
brick()
```

Shape #1: A brick

You want to draw the brick around the surface without explicitly drawing every single brick.

- Use variables for the x and y position – these are parameters
- Try calling the brick with different x and y positions – these are arguments
- *NOTE: the width and height of the brick will stay the same (no variables)*

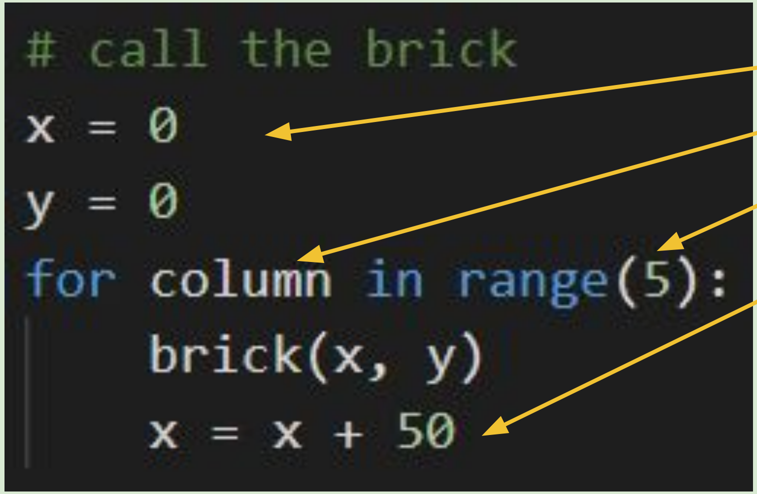


```
# define a brick
def brick(x, y):
    display.fill_rect(x, y, 50, 20, RED)
    display.draw_rect(x, y, 50, 20, WHITE)

# call the brick
brick(100, 80)
```

Tessellating the brick

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

- Use a loop to fill the surface with a row of bricks.



```
# call the brick
x = 0
y = 0
for column in range(5):
    brick(x, y)
    x = x + 50
```

Initial values for x, y

Loop counter

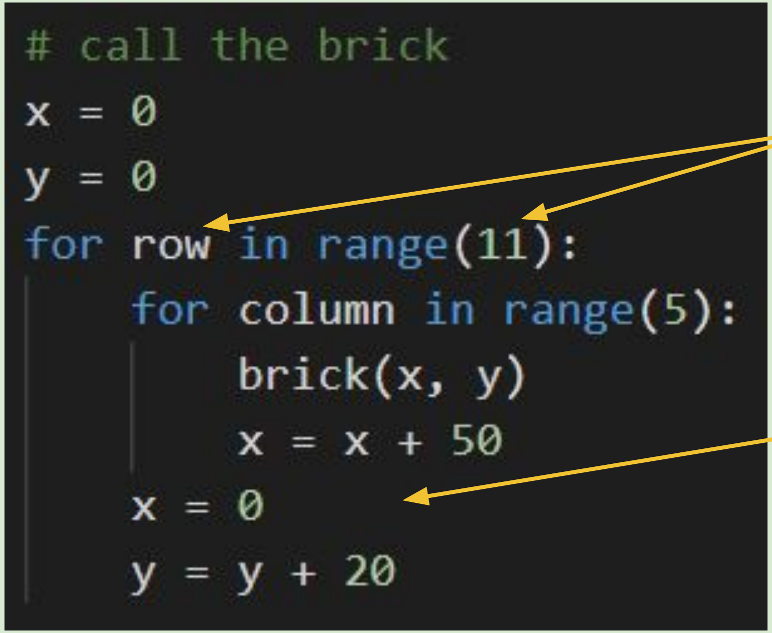
How many loops

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

Experiment with the loop counter until you get a single row of bricks

Tessellating the brick

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



```
# call the brick
x = 0
y = 0
for row in range(11):
    for column in range(5):
        brick(x, y)
        x = x + 50
    x = 0
    y = y + 20
```

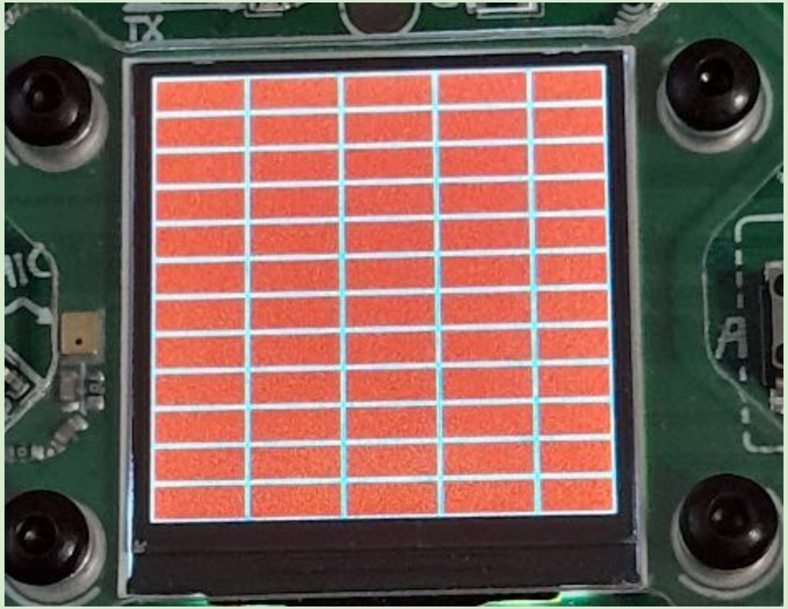
For each row, use a different loop counter and number of loops.

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

- Let’s offset the bricks by half the width

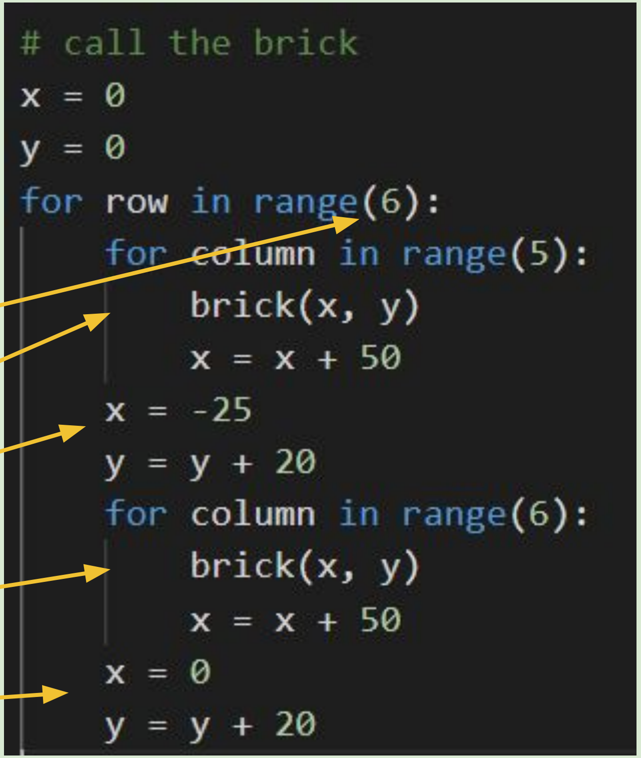
Half as many loops because each loop will draw 2 rows

Regular row of bricks

Offset x, change y

Offset row of bricks

Reset x, change y

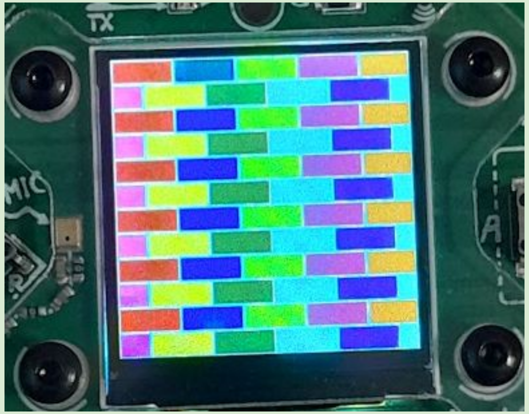


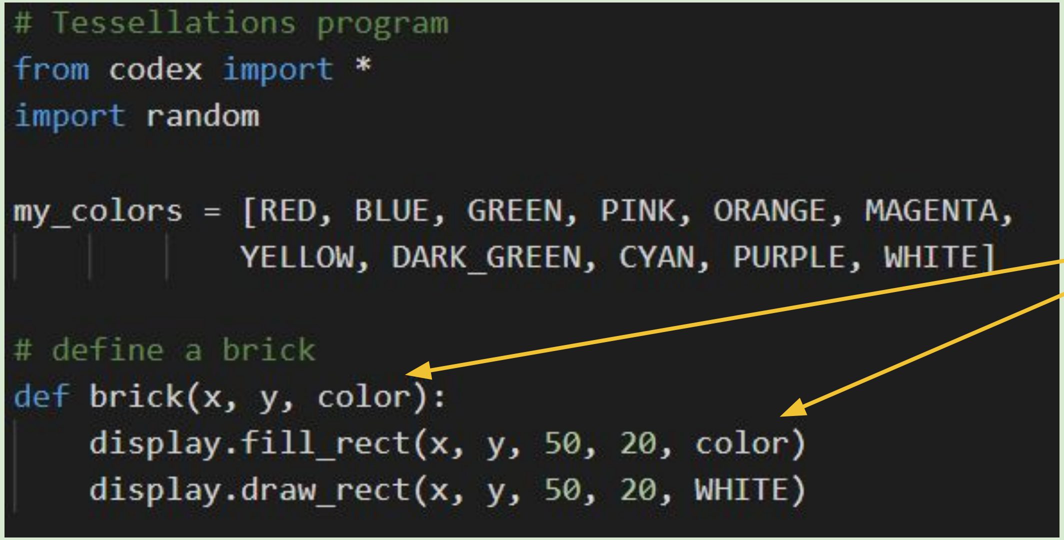
```
# call the brick
x = 0
y = 0
for row in range(6):
    for column in range(5):
        brick(x, y)
        x = x + 50
    x = -25
    y = y + 20
    for column in range(6):
        brick(x, y)
        x = x + 50
    x = 0
    y = y + 20
```

Tessellating the brick

Now add some color!

- Create a list of bright colors
- Use the list as a parameter in brick()





```
# Tessellations program
from codex import *
import random

my_colors = [RED, BLUE, GREEN, PINK, ORANGE, MAGENTA,
             YELLOW, DARK_GREEN, CYAN, PURPLE, WHITE]

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

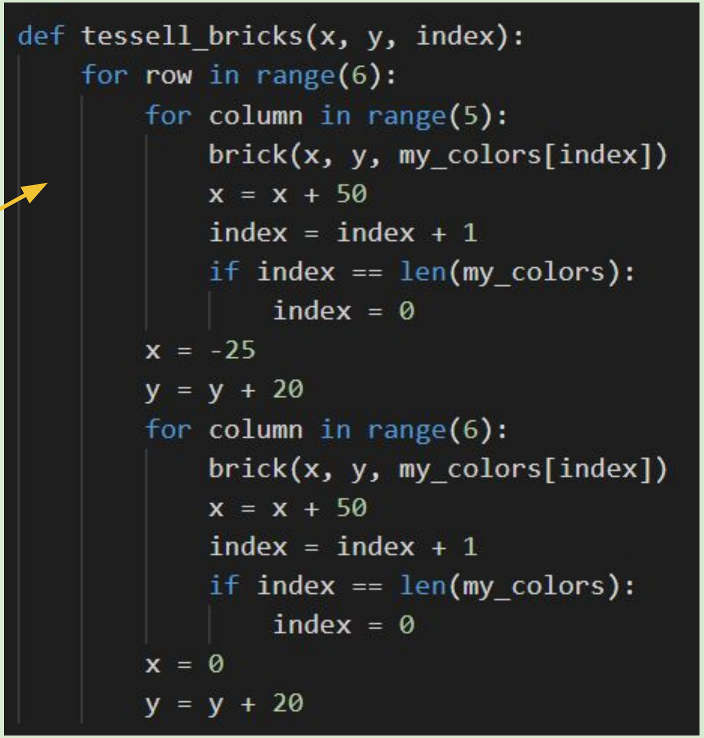
Create the list (use any colors you want, as many as you want)

Use a parameter for color

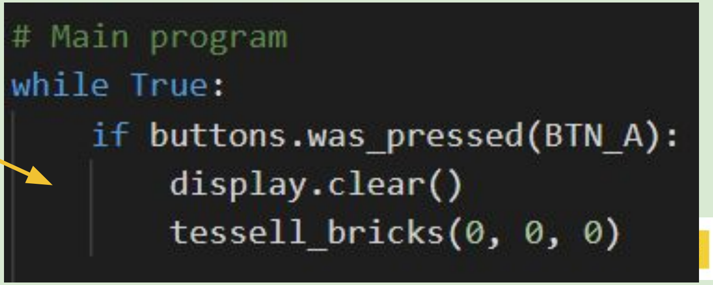
Shape #1 - Brick

Pretty cool!

- There is a lot of code here
- Let's package it into a function
- Use parameters and arguments for x, y and index
- Call the function when a button is pressed



```python
def tessell_bricks(x, y, index):
    for row in range(6):
        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 = -25
        y = y + 20
        for column in range(6):
            brick(x, y, my_colors[index])
            x = x + 50
            index = index + 1
            if index == len(my_colors):
                index = 0
        x = 0
        y = y + 20
```

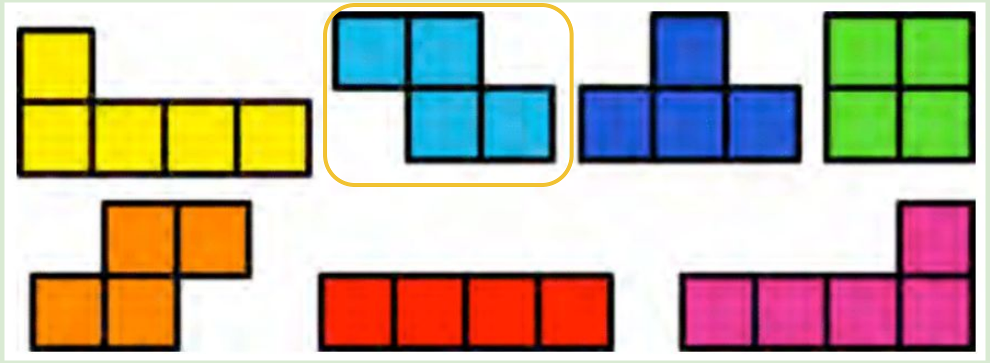


```python
# Main program
while True:
    if buttons.was_pressed(BTN_A):
        display.clear()
        tessell_bricks(0, 0, 0)
```

Shape #2 - A tetris shape

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

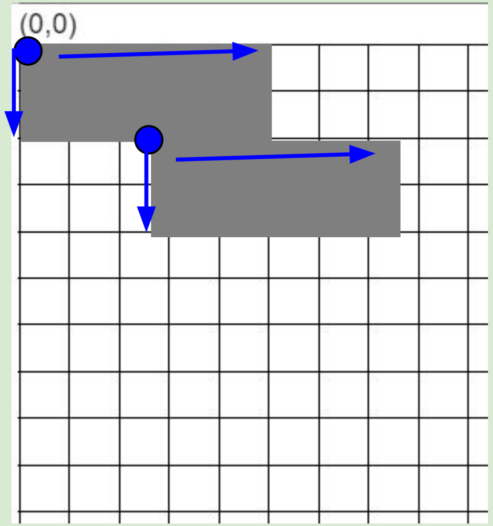
- The light blue shape can be created by putting two bricks together.
- Create a function for “skew”



Shape #2 - Skew

This code is very similar to the brick, but with the second rectangle offset

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

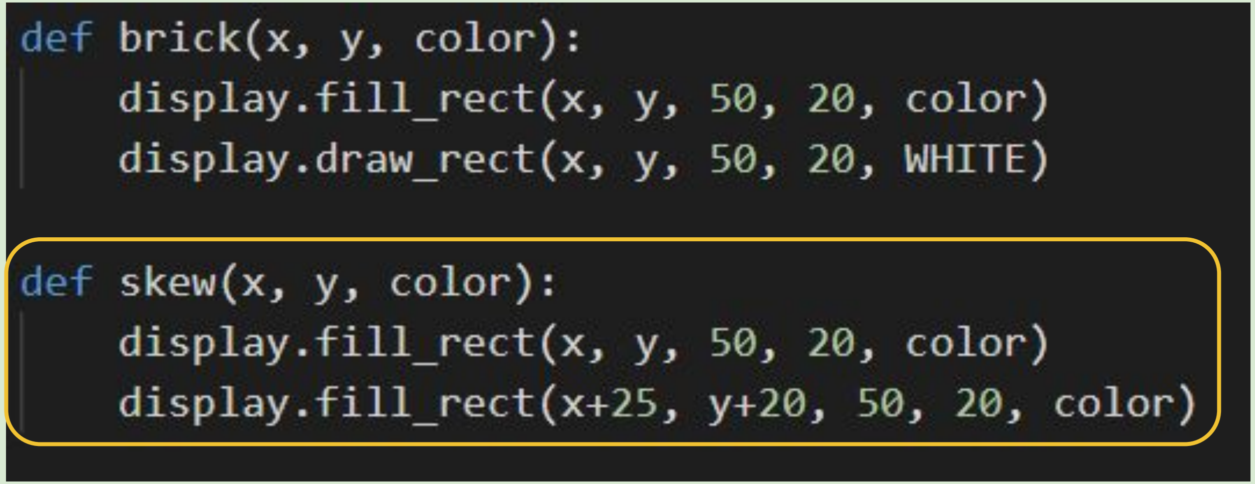


First rectangle:
(0, 0, 50, 20)
Second rectangle:
(25, 20, 50, 20)
Using x and y for both locations:
(x, y, 50, 20, color)
(x+25, y+20, 50, 20, color)

Shape #2 - Skew

This function is very similar to the brick

- No need to outline the shape, just use two fill_rect() functions
- Don't forget the parameters



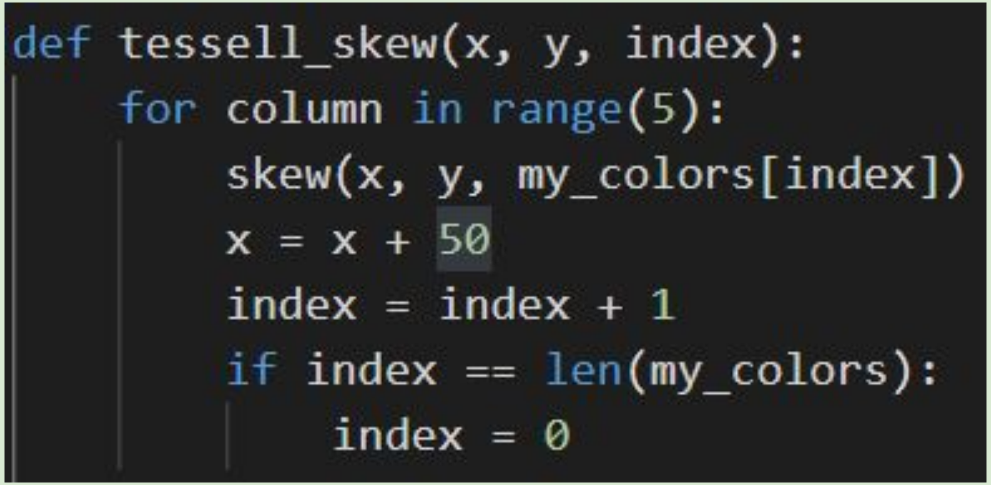
```
def brick(x, y, color):
    display.fill_rect(x, y, 50, 20, color)
    display.draw_rect(x, y, 50, 20, WHITE)

def skew(x, y, color):
    display.fill_rect(x, y, 50, 20, color)
    display.fill_rect(x+25, y+20, 50, 20, color)
```

Shape #2: Skew

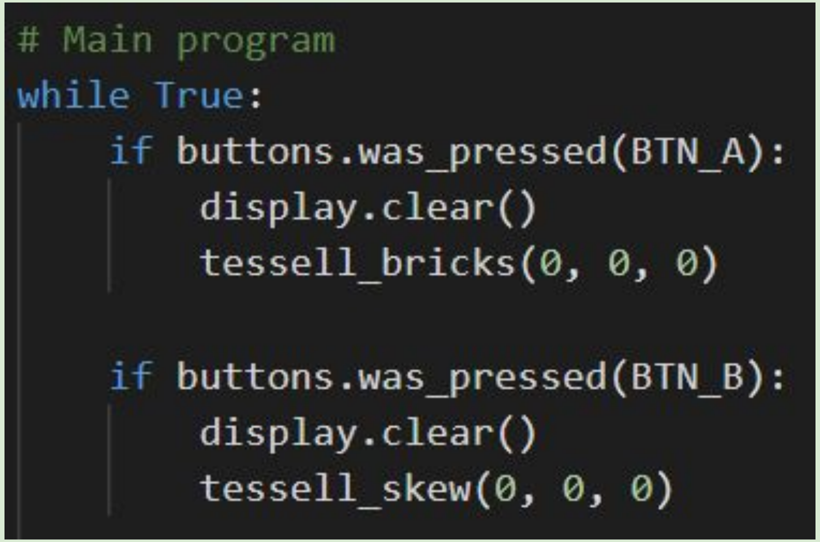
Define a function for tessellating the skew.

- Start by trying just one row.
- It will be very similar to the brick, so you can even copy and paste the code and make the needed change.
- Call the function in the main program, when a button is pressed.

Shape #2: Skew



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

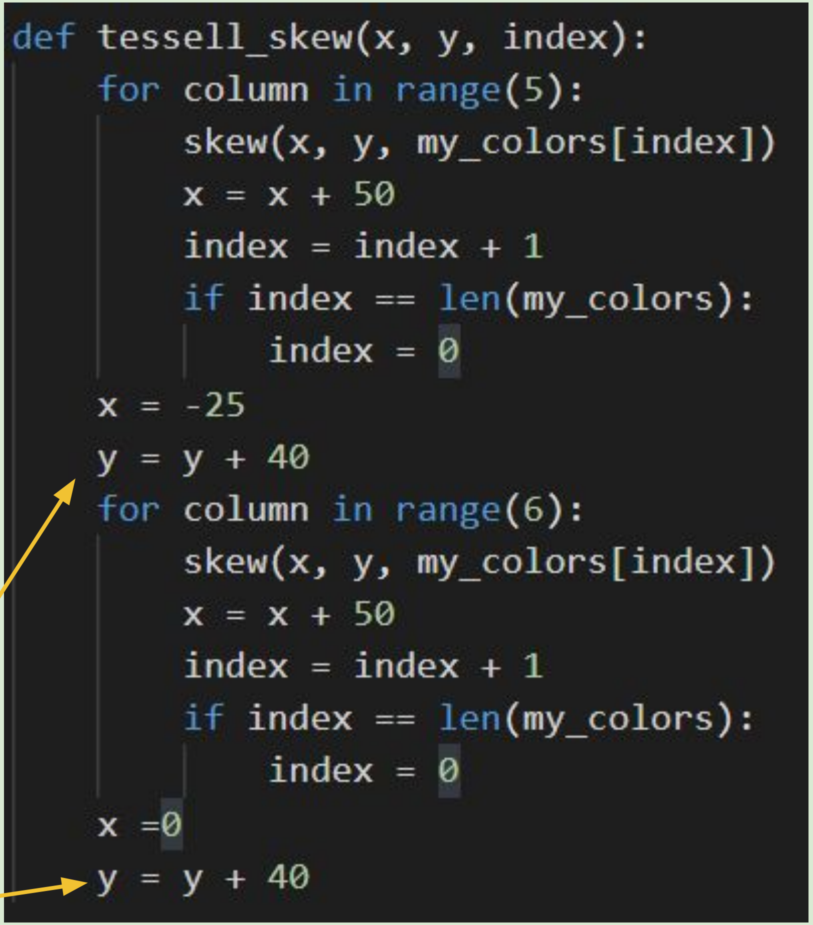


```python
# Main program
while True:
    if buttons.was_pressed(BTN_A):
        display.clear()
        tessell_bricks(0, 0, 0)

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

Shape #2: Skew

Add code to make a second off-set row of skews

- Add code to the tessell_skew() loop to offset x and make another row of the shape.
- This is very similar to the tessell_brick() function, so you can copy and paste and then make the needed changes
    - *Note: y has twice the height*

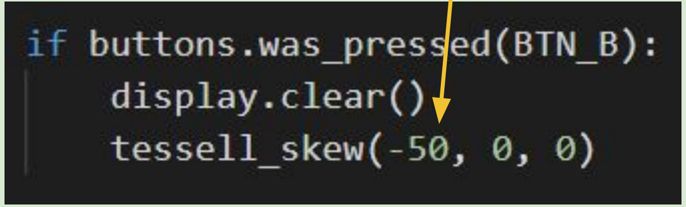


```
def tessell_skew(x, y, index):
    for column in range(5):
        skew(x, y, my_colors[index])
        x = x + 50
        index = index + 1
        if index == len(my_colors):
            index = 0
    x = -25
    y = y + 40
    for column in range(6):
        skew(x, y, my_colors[index])
        x = x + 50
        index = index + 1
        if index == len(my_colors):
            index = 0
    x =0
    y = y + 40
```

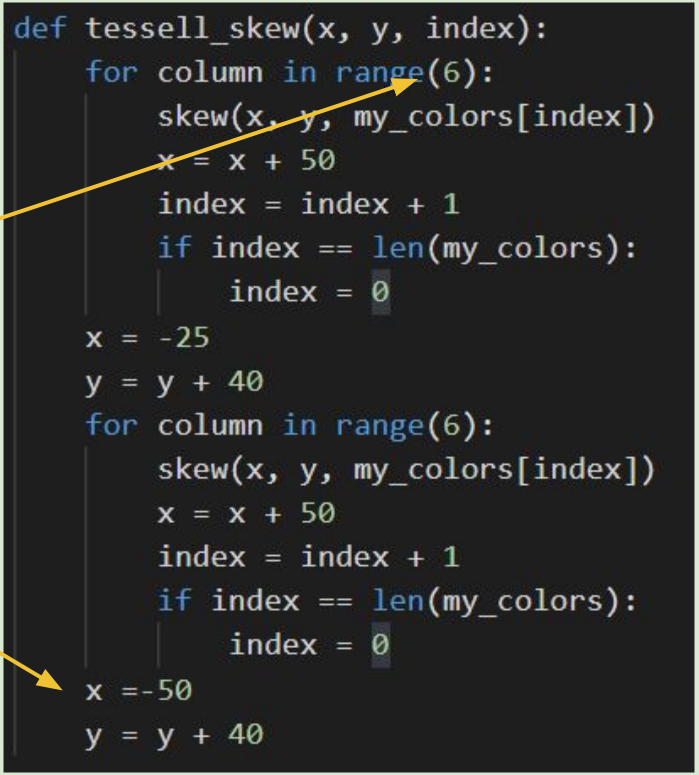
Shape #2: Skew

You may notice there is a little gap in the first row.

- To fix it, we can offset the first row
- Increase the loop count
- This is an easy fix to fill the gap.



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



```python
def tessell_skew(x, y, index):
    for column in range(6):
        skew(x, y, my_colors[index])
        x = x + 50
        index = index + 1
        if index == len(my_colors):
            index = 0
    x = -25
    y = y + 40
    for column in range(6):
        skew(x, y, my_colors[index])
        x = x + 50
        index = index + 1
        if index == len(my_colors):
            index = 0
    x =-50
    y = y + 40
```

Shape #2: Skew

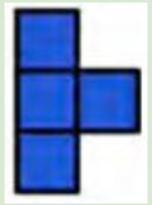
Now add the second loop for more rows, and you have your second complete tessellation.

- Do this part on your own.
- How many rows will you need in the second loop?
    - *NOTE: each iteration of the second loop will draw two rows.*

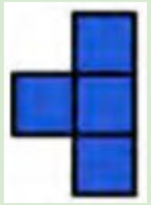
Shape #3: Half-cross

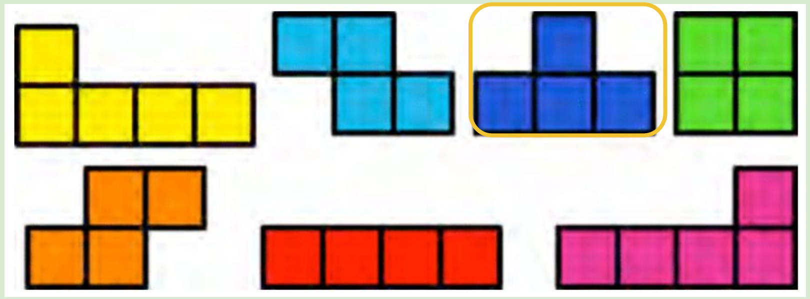
Try another Tetris shape.

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



or



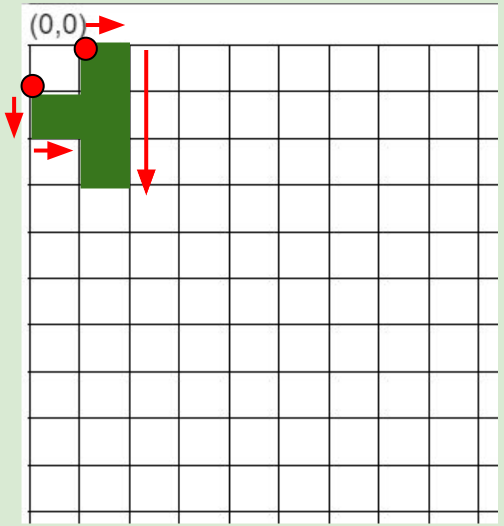


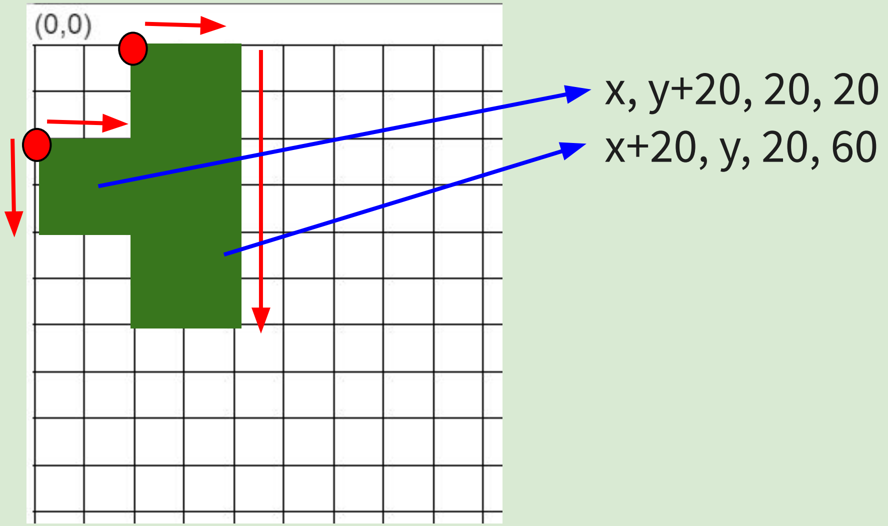
Shape #3: Half-cross

Call this Tetris shape "halfcross".

- Determine the size of your half-cross (use two rectangles)
- Determine the x and y locations needed for your shape

Examples:

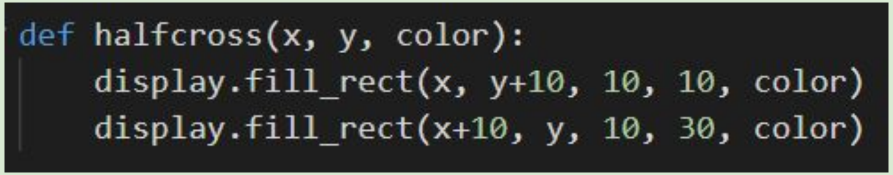




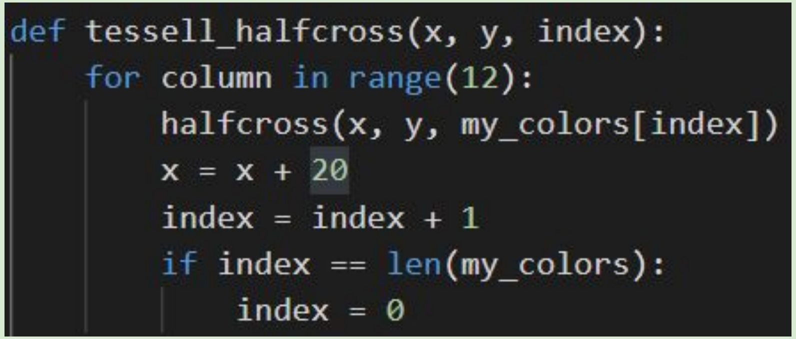
Shape #3: Half-cross

Create a function to draw the halfcross.

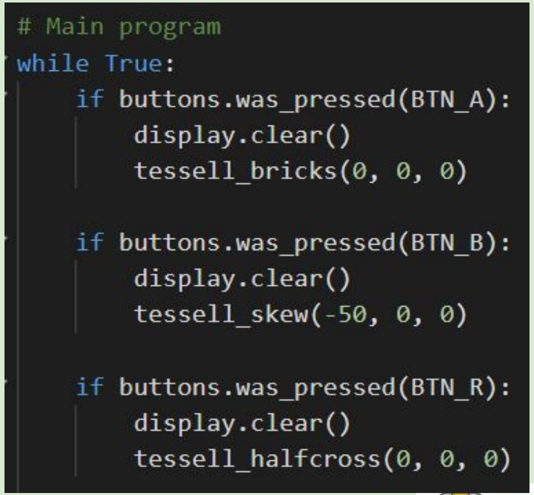
- Use your x, y width and height to draw the two rectangles.
- Create a tessell_halfcross function to call halfcross one time.
- Make sure the halfcross is the way you want it before continuing.



```
def halfcross(x, y, color):
    display.fill_rect(x, y+10, 10, 10, color)
    display.fill_rect(x+10, y, 10, 30, color)
```

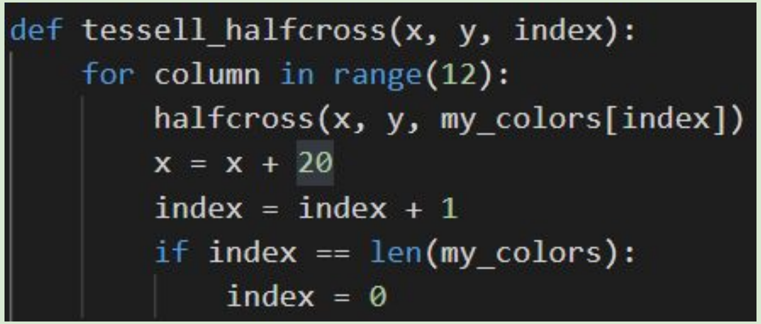


```
def tessell_halfcross(x, y, index):
    for column in range(12):
        halfcross(x, y, my_colors[index])
        x = x + 20
        index = index + 1
        if index == len(my_colors):
            index = 0
```

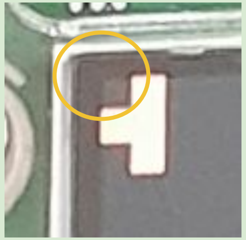


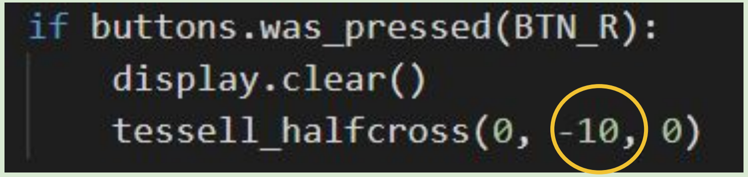
```
# Main program
while True:
    if buttons.was_pressed(BTN_A):
        display.clear()
        tessell_bricks(0, 0, 0)

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

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

Shape #3: Half-cross



```
def tessell_halfcross(x, y, index):
    for column in range(12):
        halfcross(x, y, my_colors[index])
        x = x + 20
        index = index + 1
        if index == len(my_colors):
            index = 0
```





```
if buttons.was_pressed(BTN_R):
    display.clear()
    tessell_halfcross(0, -10, 0)
```

Draw a row of halfcross shapes.

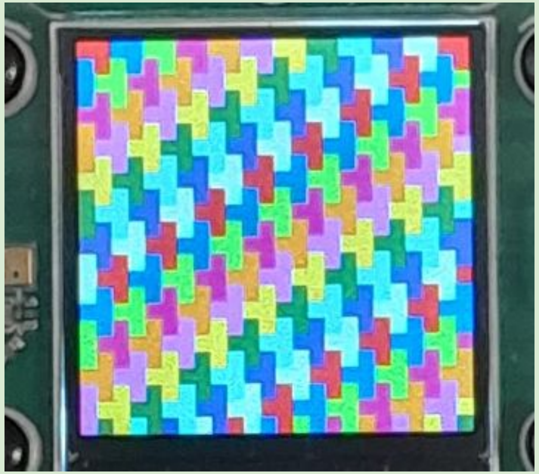
- Add a loop to create one row of shapes.
- How many loops are needed to go across the screen?
- You may notice a gap in the upper left corner.
- With skew, you offset the x value to fill in the gap. This time adjust the y value to fill the gap.
- Use an argument that works for your shape and shape size.

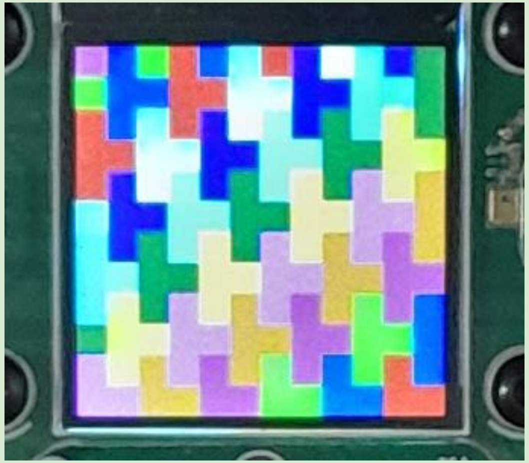
Shape #3: Half-cross

Draw a tessellation of halfcross shapes.

Once you have a row of shapes the way you want it:

- Add another row of offset shapes in the loop.
- Add another loop to add rows to fill the surface.
- Call the function in the main program, when a button is pressed.





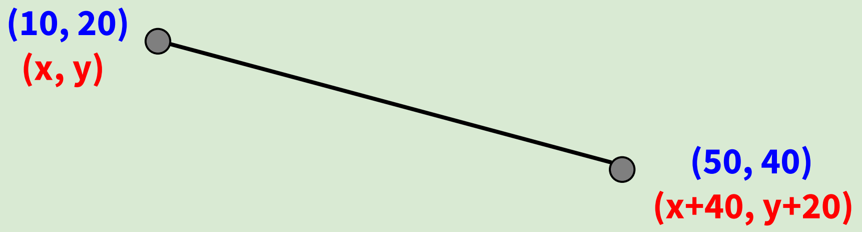
Shape #4: Isosceles triangle

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)
```

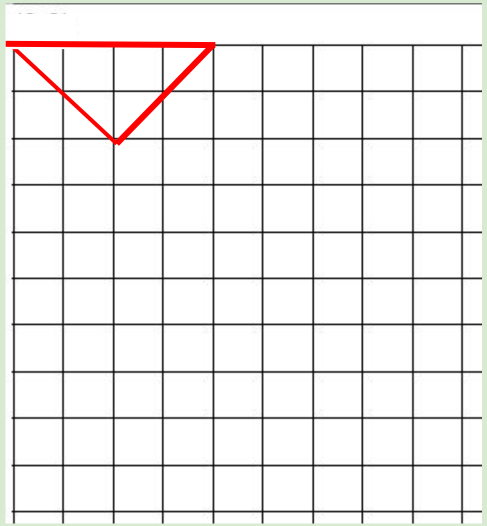
- x1, y1 are the location (column, row) of the first line end
- x2, y2 are the location (column, row) of the second line end
- x2 & y2 can be defined in relation to x1 & y1

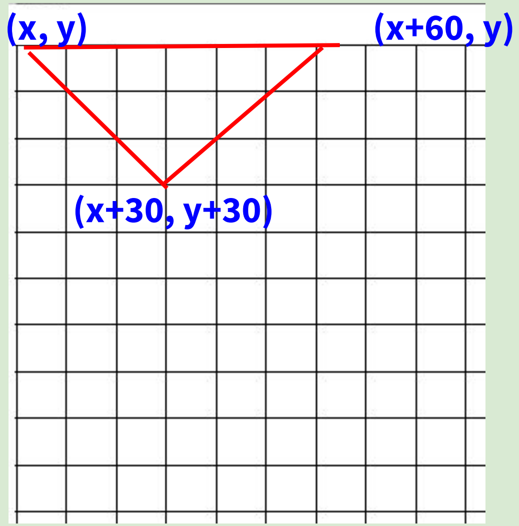


Shape #4: Isosceles triangle

Use graph paper to design an isosceles triangle.

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





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.
- Create a tessell_triangle function to draw a row of triangles.
- Add a button press to call tessell_triangle
- Make sure the triangle is the way you want it.

Shape #4: Isosceles triangle

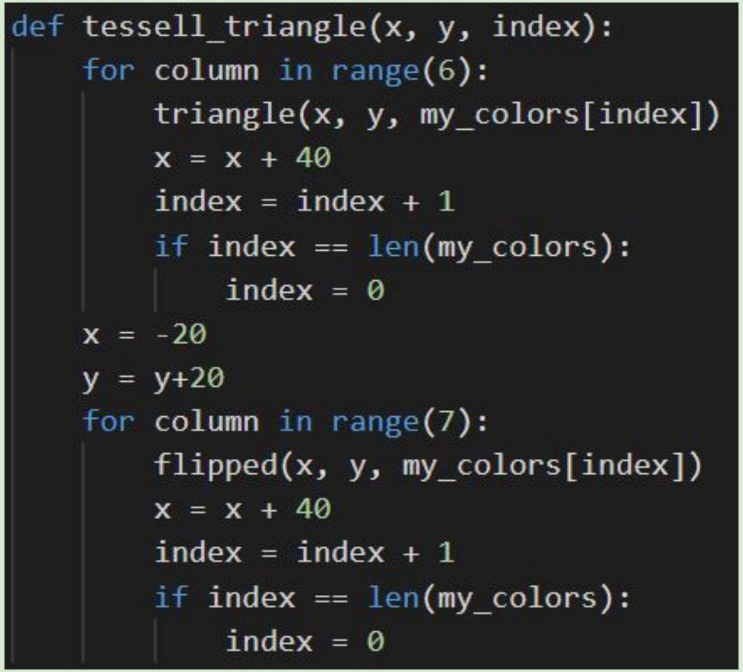
The need for a flipped triangle

- After drawing one row of triangles, you determine that just offsetting the next row will not fill in the gaps, and will cause overlap – not a tessellation.
- To fill the gap you will need a flipped triangle.
- Use the graph paper to determine the values needed to draw a flipped triangle.
- Create another function for the flipped triangle.

Shape #4: Isosceles triangle

Draw two rows of triangles

- Add a second loop to draw a row of flipped triangles.
- Determine what values are needed for x and y before starting the second row.
- Your code could look similar to the example.

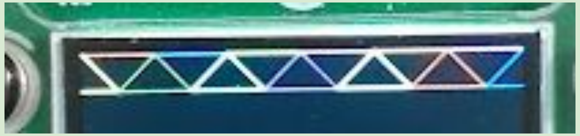


```
def tessell_triangle(x, y, index):
    for column in range(6):
        triangle(x, y, my_colors[index])
        x = x + 40
        index = index + 1
        if index == len(my_colors):
            index = 0
    x = -20
    y = y+20
    for column in range(7):
        flipped(x, y, my_colors[index])
        x = x + 40
        index = index + 1
        if index == len(my_colors):
            index = 0
```

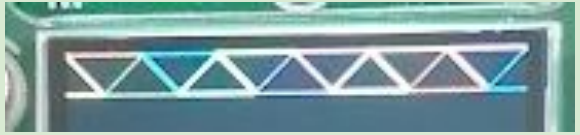
Shape #4: Isosceles triangle

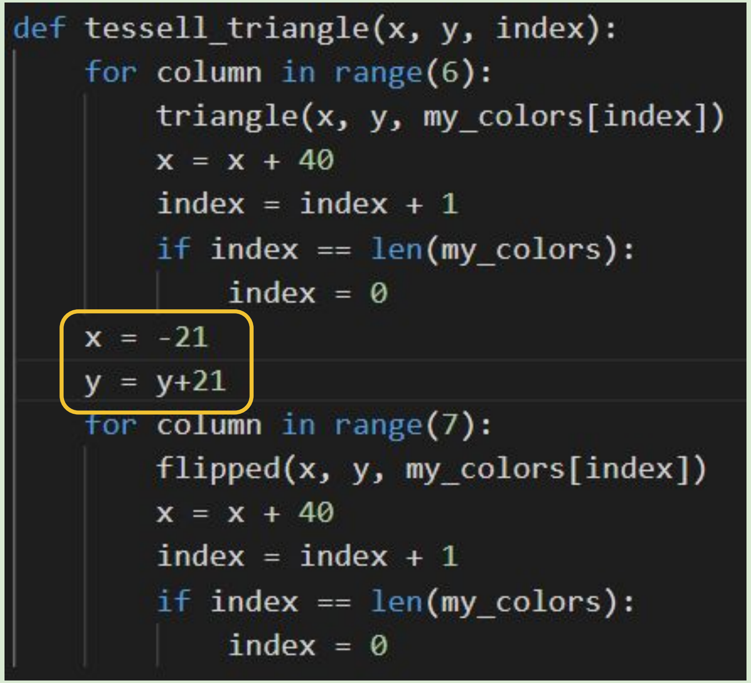
Draw two rows of triangles

- You notice that the second row of triangles draws over part of the first row of triangles.



- Offset the x and y values by 1 pixel to see more of each triangle.



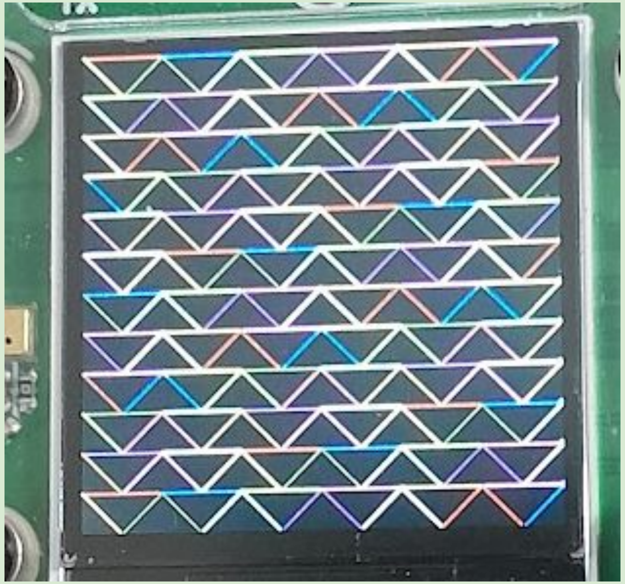


```
def tessell_triangle(x, y, index):
    for column in range(6):
        triangle(x, y, my_colors[index])
        x = x + 40
        index = index + 1
        if index == len(my_colors):
            index = 0
    x = -21
    y = y+21
    for column in range(7):
        flipped(x, y, my_colors[index])
        x = x + 40
        index = index + 1
        if index == len(my_colors):
            index = 0
```

Shape #4: Isosceles triangle

Draw a tessellation of triangles

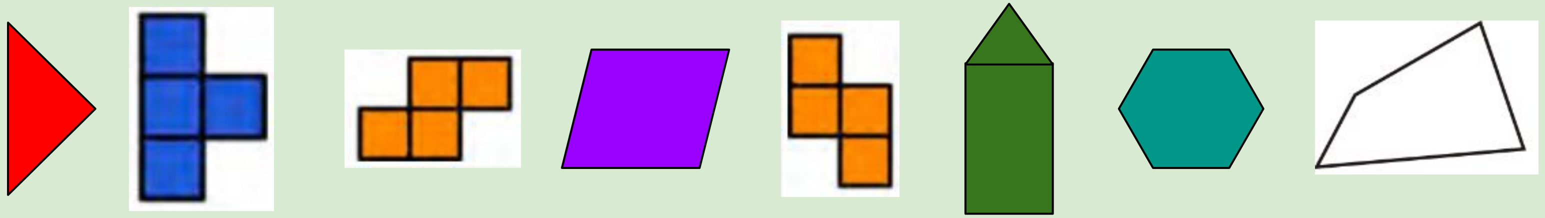
- Add another loop to add rows to fill the surface.
- You can use some math to determine the values of x and y, or use trial and error
- *HINT: The flipped triangles are on the same row as the next regular triangles.*



Shape #5: Your own shape

Decide on a shape to tessellate.

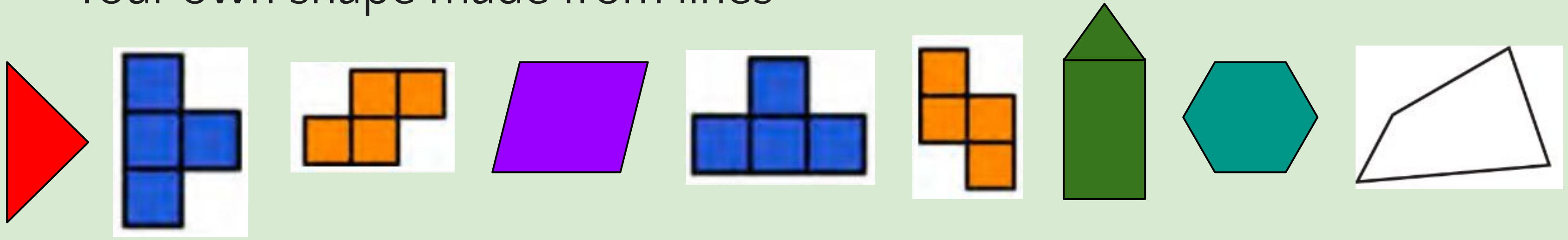
- Use graph paper to design your shape.
- Make sure your shape is able to tessellate –
  - It needs to fill the surface without gaps or overlaps
  - Your shape could require a flip or rotation



Shape #5: Your own shape

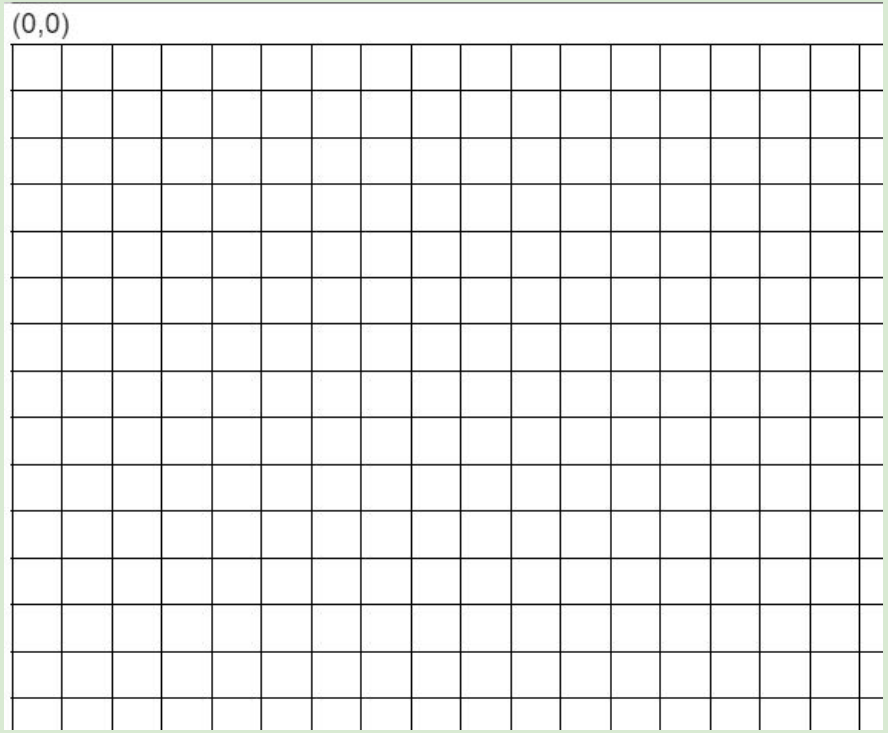
Decide on a shape to tessellate.

- 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.

- Create a my_shape() function to draw your own shape
- Create a tessell_my_shape() function to draw a single row of your shape
- When the row looks like a tessellation row, complete the code by adding an offset row and a loop
- Call the function in the main program, when a button is pressed.
- Test and debug as needed

Tessellation Extensions

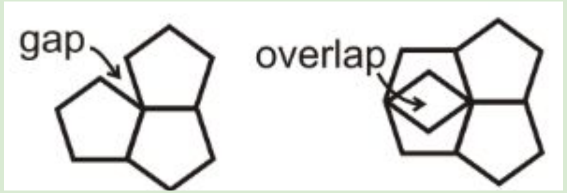
Geometry emphasis:

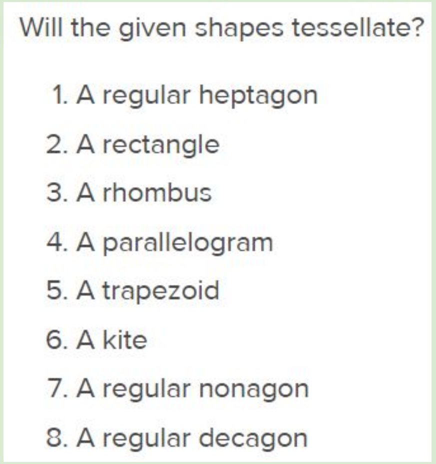
- Scaling – use graph paper to draw the shape and then code it on CodeX. Measure it on the shape on the Codex and calculate the scaling. Use different types or sizes of graph paper to demonstrate different scales of the shape and calcuate the scaling.

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.



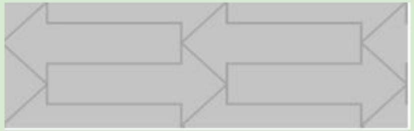
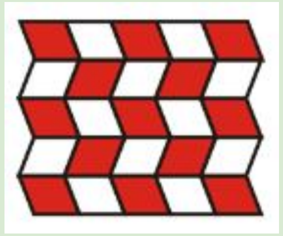
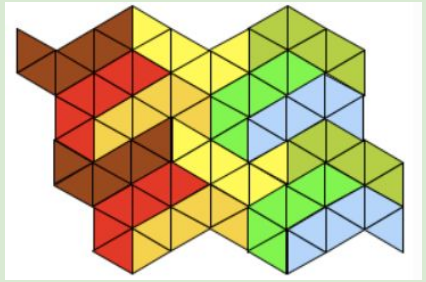
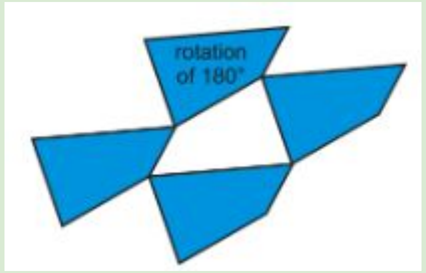

Will the given shapes tessellate?

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.

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