#9: For Loops and Range

SAMS SENIOR NON-CS TRACK

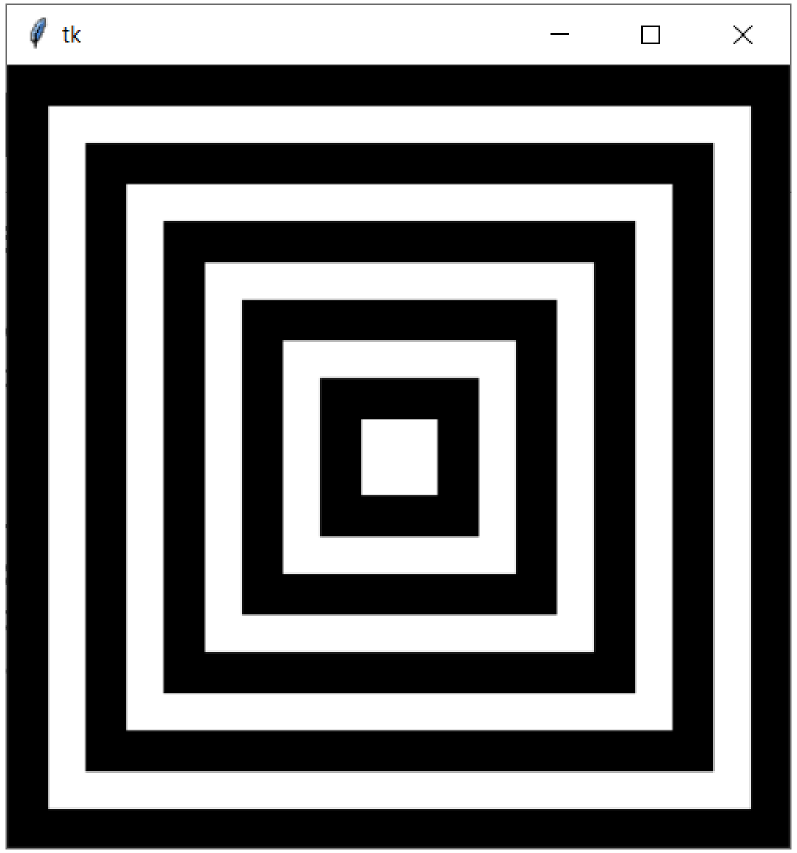
Last Time

Use a **while loop** to repeat actions until a certain condition is met

Use **break** and **nesting** to change the control flow of while loops

Ex 5-1 Feedback

Problems 1-3 went really well, but many of you didn't finish #4. Let's solve that now!



Today's Learning Goals

Use **for-range loops** to iterate a specific number of times

Use **for-each loops** to iterate over strings

Use **nested loops** to create two-dimensional patterns

For-Range Loops

For Loops for Repeated Actions

We've learned how to use while loops and loop variables to iterate until a certain condition is met. When that condition is straightforward (increase a number until it reaches a certain limit), we can use a more standardized structure instead.

A **for-range loop** tells the program exactly how many times to repeat an action. The loop variable is updated by the loop itself!

```
for <loop_variable> in range(<max_num_plus_one>):
    <loop_body>
```

While Loops vs. For Loops

To print the numbers from 0 to 5 in a while loop, we would write the following:

```
i = 0
while i <= 5:
    print(i)
    i = i + 1
```

In a for-range loop, we'll automatically start the loop variable at 0, and it will automatically increase by 1 each loop.

```
for i in range(6):
    print(i)
```

Note that we have to go up to 6 because for-range goes **up to but not including** the given number. It's like saying while i < 6.

Exercise 1: 0 to 10

Go to the schedule page and download the starter file for today's lecture. You'll write exercise code under the comment with the exercise's number.

Exercise 1: write a few lines of code that prints the numbers from 0 to 10 using a **for loop**.

Range creates variable values

When we say `for i in range(10)`, `range(10)` generates all the values that i will eventually hold. This is how i knows which value it should update to hold next each iteration.

We can update range to give it more information! If we call range on two numbers, it will start i at the first number and end i just before the last. So the following code would generate the numbers 3, 4, 5, 6, and 7.

```
for i in range(3, 8):
    print(i)
```

Range has a step

If we want to get really fancy, we can add a third argument to the range() function. This last argument is the **step** of the range, or how much the number should increase by each time. The following example would print the even numbers from 1 to 10:

```
for i in range(2, 11, 2):
    print(i)
```

Anything we can do in a for loop can also be done in a while loop. In a while loop, this would be equivalent to:

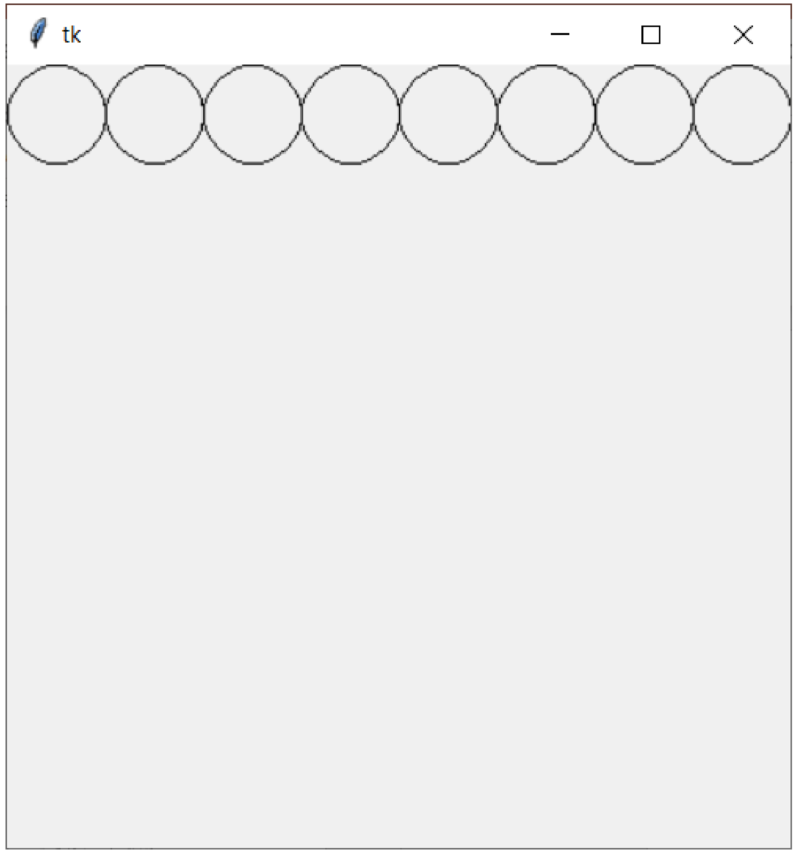
```
i = 2
while i < 11:
    print(i)
    i = i + 2
```

Repeating Graphics

For-range loops can be useful in graphics, to repeat a certain pattern every so-many pixels. We just need to call the canvas function inside the loop to create several graphical shapes.

The following code draws a line of circles at the top of the canvas, each 50px wide:

```
for x in range(0, 400, 50):
    canvas.create_oval(x, 0, x + 50, 50)
```

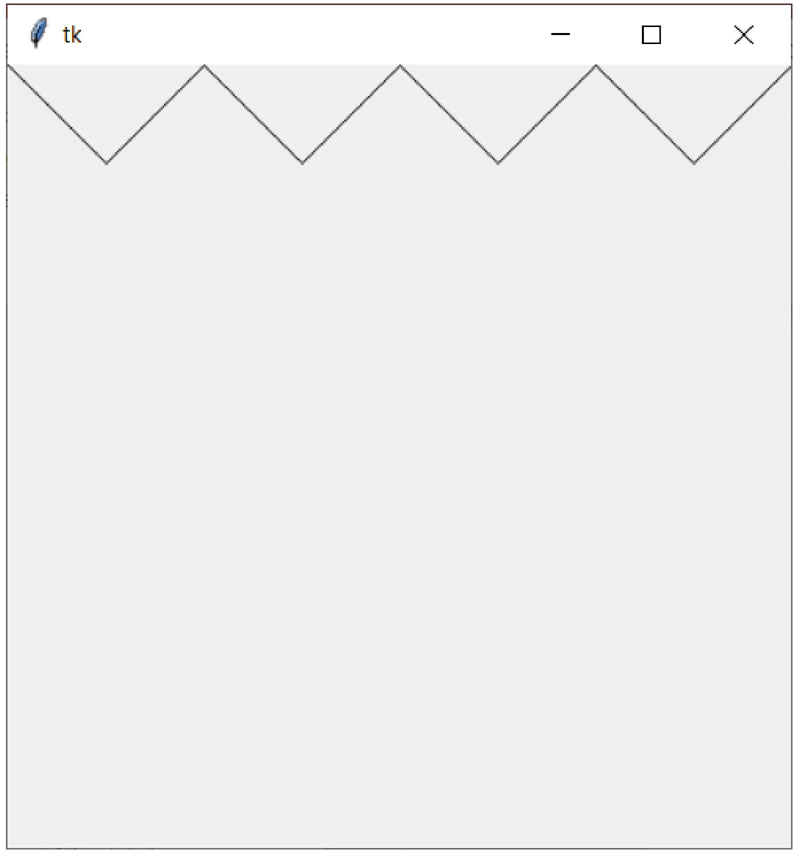


Repeating Graphics with a Twist

By using alternating Boolean variables (as we discussed last week), we can even change the graphics across iterations!

The following code draws a jagged line by changing the y coordinate every other iteration while updating the x coordinate by a constant amount. Note that we still need to update isTopLine, which is not affected by range().

```
isTopLine = True
for x in range(0, 400, 50):
    if isTopLine:
        canvas.create_line(x, 0, x + 50, 50)
    else:
        canvas.create_line(x, 50, x + 50, 0)
    isTopLine = not isTopLine
```

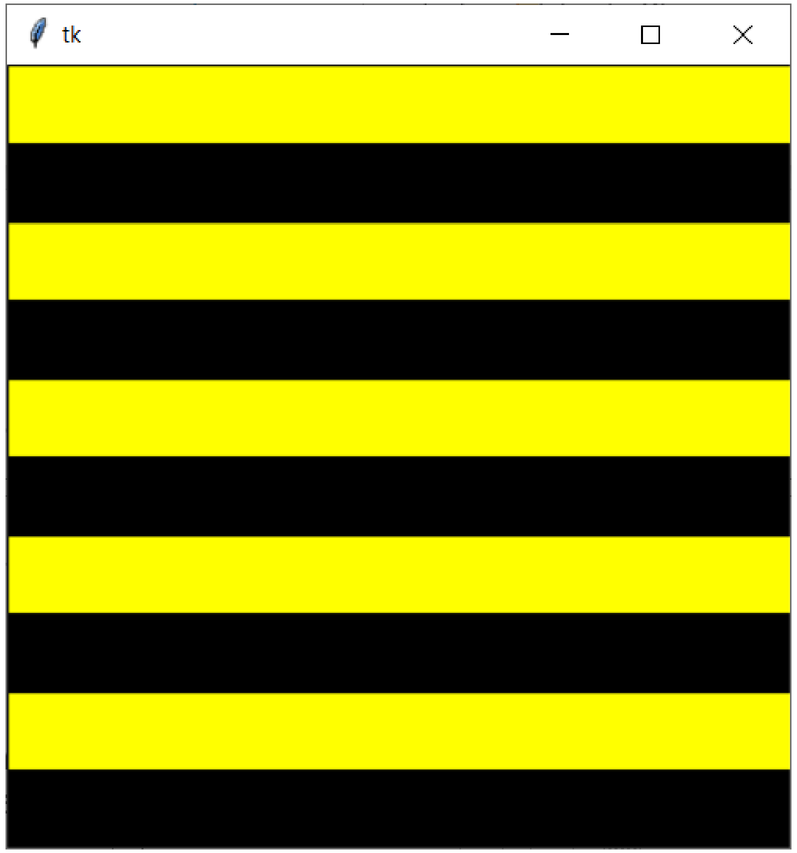


Exercise 2: stripes

Exercise 2: write a few lines of tkinter code that draws a flag with 10 stripes, as is shown to the right. You must use a for loop to get full credit.

Note that you'll need to calculate the height of a stripe such that seven stripes fit on the screen.

Hint: there are two common ways to approach this problem. One method uses two for loops (one for the yellow stripes, one for the black); the other method uses one for loop with an alternating variable.



For-Each Loops

For Loops with Strings

We can also use for loops to iterate over data that can be thought of as multiple parts put together in a whole (**iterable**). A string can be thought of as a sequence of letters (**characters**). Using a for loop, we can write a program that loops over each of the characters in order.

```
for <character_variable> in <string>:
    <character_action_body>
```

For example, if we run the following code, it will print out each character of the string individually.

```
for c in "Hello":
    print(c)
```

Example String Loop

Prediction Exercise: what do you think the following code prints?

```
s = "Hello"
t = ""
for c in s:
    t = c + t
print(t)
```

Casing on Characters

Usually, when we loop over a string, we want to case out different characters using if statements, then react to them in different ways. For example, this program prints out just the lowercase characters in the given string.

```
s = "Hello Everyone How Are You?"
for c in s:
    if "a" <= c and c <= "z": # if c is lowercase
        print(c)
```

Exercise 3: secret message

Exercise 3: a string has been stored in the variable message in your starter file. It has a *secret message* inside of it, which can be extracted by removing all the lowercase letters from the string. Write a few lines of code that combine together just the uppercase letters and print them out as a whole. Your code should still work if the secret message is replaced by a different secret message.

Note: don't just print out the uppercase characters immediately! Make sure they're all printed out on the same line, as a single string.

Nested Loops

Nesting Loops

Just as we can nest conditionals in loops, we can also nest loops inside loops! We mostly do this with for-range loops, and mostly when we want to loop over *multiple dimensions*.

```
for <loop_var_1> in range(<end_num_1>):
    for <loop_var_2> in range(<end_num_2>):
        <both_loops_body>
    <just_outer_loop_body>
```

When we nest loops, we repeat the inner loop **every time** the outer loop takes a step.

Nested Loops Example: Coordinates

For example, let's say we want to print all the coordinates on a plane from (0,0) to (5,5)

```
for x in range(5):
    for y in range(5):
        print("(", x, ",", y, ")")
```

Note that every iteration of y happens anew in each iteration of x.

Nested Loops in Graphics

We generally use nested loops when we work with naturally two-dimensional data. Right now, that means **pixels in graphics!**

We can think of graphics in terms of x,y coordinates, or in terms of rows and cols in a grid. Rows and cols are just larger versions of pixels- instead of taking up only one pixel of space, each cell of a grid might be 5px x 5px, or 20px x 20px.

Calculating row/col coordinates

Let's say we want to fill a 400px x 400px window with an 8 x 5 grid. We need to figure out the top-left and bottom-right coordinates of each cell of the grid.

Note that to fill the window, each cell needs to be **400 / 5** pixels wide. The first cell starts at x coordinate 0, then ends at 400 / 5. That means that the next cell starts at x coordinate 400 / 5, and ends at 400 / 5 * 2!

In other words, if we start counting columns at 0, **each col starts at coordinate col * 400 / 5**. The same logic can be used to see that each row starts at coordinate row * 400 / 8, since we want the window to have eight rows.

Drawing the grid

The following program draws out a 5 x 5 grid with text, using the coordinate logic from the previous slide. Each cell of the grid is represented by a (row, col) text. Note how we average the left/right and top/bottom coordinates to find the middle coordinate.

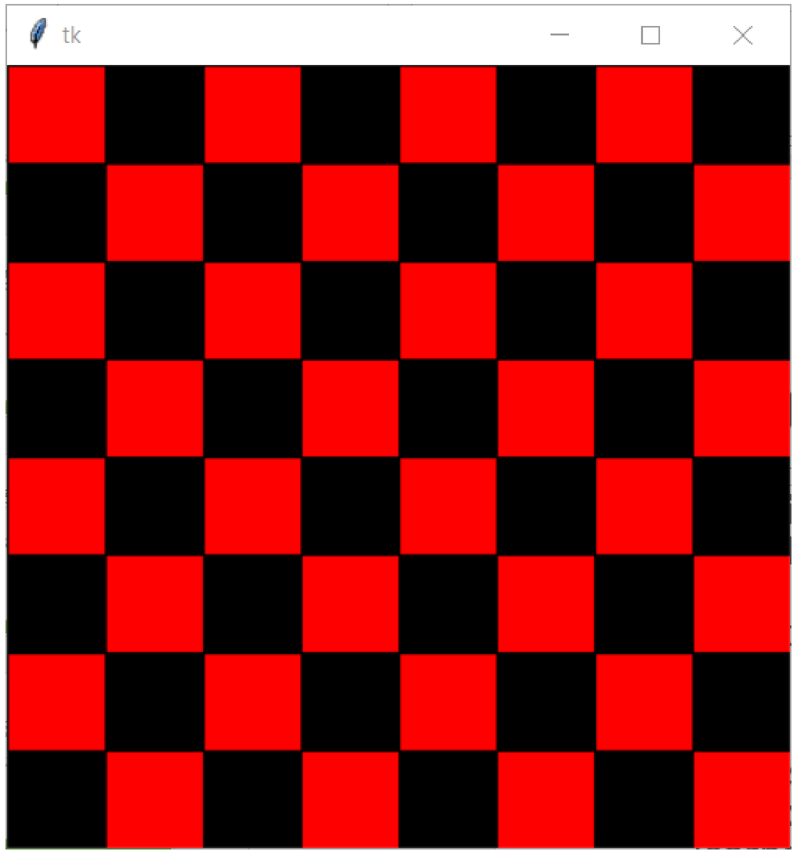
```
for row in range(5):
    for col in range(5):
        left = col * 400 / 5
        top = row * 400 / 5
        right = (col + 1) * 400 / 5
        bottom = (row + 1) * 400 / 5
        canvas.create_text((left + right)/2, (top + bottom) / 2,
                           text="(" + str(row) + "," + str(col) + ")",
                           font="Arial 20 bold")
```

Exercise 4: checkerboard

Exercise 4: write a few lines of tkinter code that draw an 8 x 8 checkerboard on the canvas, as is shown to the right. You must use a nested loop for full credit.

Hint: start by just drawing a checkerboard as an 8 x 8 grid. Then consider what we've learned about this week that will make it possible to alternate the colors...

Hint 2: if you end up with columns that are the same color, think about where else you can alternate the colors in the code.



Today's Learning Goals

Use **for-range loops** to iterate a specific number of times

Use **for-each loops** to iterate over strings

Use **nested loops** to create two-dimensional patterns