loops part 2

what kind of loop would you use?

- you want to select one variable (average reaction time) from a set of files you have, where you already know how many files there are
- you want to read in all the data from a file, and each time you do this the file may be a different length
- you want to run a simulation of disease until a certain percent of the population is sick
- you want to run a genetic model to understand how the number of generations affects an outcome

some loops can be translated!

```
a = 0;
while a < 5
    a = a+1;
    messagel = 'stil counting! im at ';
    numbera = num2str(a);
    full_message = strcat(messagel, numbera);
    disp(full_message)
end
disp('final a:')
disp(a)
```

```
stil counting! im atl
stil counting! im at2
stil counting! im at3
stil counting! im at4
stil counting! im at5
final a:
     5
```

how would we translate this to a for loop?

some loops can be translated!

```
a = 0;
while a < 5
    a = a+1;
    messagel = 'stil counting! im at ';
    numbera = num2str(a);
    full_message = strcat(messagel, numbera);
    disp(full_message)
end
disp('final a:')
disp(a)
```

```
stil counting! im at1
stil counting! im at2
stil counting! im at3
stil counting! im at4
stil counting! im at5
final a:
     5
```

```
for a=1:5
    messagel = 'stil counting! im at ';
    numbera = num2str(a);
    full_message = strcat(messagel, numbera);
    disp(full_message)
end
disp('final a:')
disp(a)
```

```
stil counting! im at1
stil counting! im at2
stil counting! im at3
stil counting! im at4
stil counting! im at5
final a:
     5
```

Complex example

Write a program that will use a while loop to find the smallest number divisible by both 3 and 5, with the square of the number greater than 3325.

hint! you can use your rem() function from last class

Complex example

Write a program that will use a while loop to find the smallest number divisible by both 3 and 5, with the square of the number greater than 3325.

Steps:

1. Initialize variables for **square** and **number**
2. Loop while square < 3325
3. Advance number by 3 in each pass
4. Check if the **number** is divisible by 5.
5. If yes, square the number
6. Repeat until square > 3325
7. Loop stops, we have our answer :)

```
n=0; sq=0;          %initialize variables: # & square
while sq<=3325      %loop until square >3325
   n=n+3            %advance # by 3 each pass
   if rem(n,5)==0   %check if # is divisible by 5
      sq=n^2;       %if yes, square #
   end              %end if statement
end                 %end while loop

disp(n)             %display the # that meets criteria
```

Maybe you now ?

Write a while loop that **displays** on every step the **result** of adding all the numbers from 1 to n. For example the first line would be 3 (1+2), the second one 6 (1+2+3), …. Until 1+2+3+…+n

Think – what will the input to the function be (something you change)? What will be the output (in this case, all we need to do is display)? What will change on each step of the loop? What can you keep track of?

If you aren't sure where to start, pick a number for n and try just writing the outputs on paper yourself (1, 3, 6, 10 …). How did you know what came next?

Maybe you now ?

Write a while loop that **displays** on every step the **result** of adding all the numbers from 1 to n. For example the first line would be 3 (1+2), the second one 6 (1+2+3), ... Until 1+2+3+...+n

```
function sum_to_n(n)
% display the sums of all integers, from 1 to n
% input: n, an integer (will not work for decimals)
% output: none (all work is displayed)

current_total = 0; % start a counter at 0
for i=1:n % loop through what we're going to add to our counter
    current_total = current_total + i; % increase our count by the part of the loop we're on
    disp(current_total) % show the current value to the command window
end
% nothing else! not returning anything
end
```

meta “comment”

this week, your coding in context will be about commenting code! if you take a look at it ahead of time, you can work on it while you work on your code and comment as you go.

word to the wise: commenting is much easier to do as you go rather than afterwards. sometimes if you wait too long, it will look like it’s not even your own code

super bowl week!

if you watch football, you may have seen virtual lines which appear before (and sometimes during) a play to indicate how far the team needs to move the ball to trigger a first down.

let’s imagine we’re tasked with writing code which decided whether we should move the yellow line, and to where it should go

Football Loops

if you aren't familiar with american football rules: teams have 4 tries (called downs) to get the ball across the line each time it moves. if they do, you move the line 10 yards from the football's location. so, if the ball is at 0, and they move it to the 4 yard line, the line stays at 10 and they only have 3 downs left. if the next down gets them to the 12 yard line, the line moves to 22 (10+12) and they start again with 4 downs.

if they don't make it in 4 downs, the other team gets the ball and they don't score. if they get the ball past the 100 yard line, they score.

imagine your starting position of the ball is always 0, so the initial position of the line is at 10. you want to write code which tracks the location of the yellow line until the team either scores or loses the ball.

using whatever other variables, loops, and logic you need, write code which follows where the ball is.

start with variables:

```
ball_location = 0;
yard_line = 0;
downs_left = 4;
```

and imagine you have a function `nyards = play_down()` which takes no variables and returns how many yards the team moves on each play.

whenever you move the line, you want to `disp('first down!')`. if the team scores, `disp('touchdown!')`. if the team loses the ball, `disp('loss of possession')`. as you write the code, you should have it output whatever else helps you debug.

hint: you should combine boolean logic with loops! you may need to use 2 loops

olympics week!

you are watching the olympics and want to cheer on your competitor! every time they do a jump, you want to cheer “go!” for the number of spins in a jump. so, if you seem them do three spins, you should cheer “go! go! go!”

you don’t know their whole routine (how many jumps they will do), but you know it might end with the never before seen quintuple-turn jump. so when they do 5 spins, you don’t want to just cheer go – instead you want to yell “world record!” to emphasize their final touch.

but, if the skater doesn’t feel ready, they will instead just skate until the total number of spins goes over 25. if they hit this, you should cheer “you did it!” instead.

finally, you want to track how many points they got – they get 3 points per jump they did, plus 1 point per spin in the jumps (so a 2 spin jump is worth 5 points).

your skater will do 3 performances, and you should output the score on each performance and the total at the end.

write code to cheer on your skater during their routine. you should keep track of
`number_of_jumps = 0;`
`number_of_spins = 0;`
and any other variables you think you’ll need. imagine you have a function
`nspins = do_jump()` which takes in no inputs but outputs the number of spins they do on any jump (from 1 to 5).

hint: you will need to use boolean logic with your loop! you’ll need two loops!