word = 'lead';

disp(word(1));

disp(word(2));

disp(word(3));

disp(word(4));

it doesn’t scale: if we want to print characters of a string of a hundred letters, it will be easier to type them in.

it’s fragile: if we change word to a longer string, it will only print part of it, and if it is shorter, error.

word = 'tin';

repeat!

word = 'lead';

for letter = 1:4

disp(word(letter))

end

1. General form of a loop is:

for variable = collection

do things with variable

end

The for loop executes the commands in the [loop body](https://swcarpentry.github.io/matlab-novice-inflammation/reference.html#loop-body) for every value in the array collection. This value is called the [loop variable](https://swcarpentry.github.io/matlab-novice-inflammation/reference.html#loop-variable), and we can call it whatever we like. In our example, we gave it the name letter.

We have to terminate the loop body with the end keyword, and we can have as many commands as we like in the loop body. But, we have to remember that they will all be repeated as many times as there are values in collection.

word = 'tin';

for letter = 1:3

disp(word(letter));

end

word = 'aluminium';

for letter = 1:length(word)

disp(word(letter));

end

example 1

len = 0

for vowel = 'aeiou'

len = len + 1;

end

disp(['Number of vowels: ', num2str(len)])

It is work trying to explain what is happening inside the loop step by step. So the variable len is created with a value equals to zero, then we enter the loop, where the variable vowel is created, and the first value (in this case a) is used, and we enter inside the loop, and update the variable len, so in this case the variable len is equal to the value of len at that point, so zero, plus one, which will update the value of len to 1. This will be repeated until wovel is equal to u.

Note that a loop variable is just a variable that’s being used to record progress in a loop. It still exists after the loop is over, and we can re-use variables previously defined as loop variables as well

disp(vowel)

disp(5^3)

Students exercise.

* Let a variable b be the base of the number and x the exponent. Write a loop to computeb^x. Check your result for b = 4 and x = 5.
* Write a loop that spells the word “aluminum,” adding one letter at a time:
* Using this, write a loop to print the letters of “aluminum” in reverse order, one letter per line.

for idx = 1:12

file\_name = sprintf('inflammation-%d.csv', idx);

disp(file\_name);

end

This is close, but not quite right. The sprintf function is useful when we want to generate MATLAB strings based on a template. In our case, that template is the string inflammation-%d.csv. sprintf generates a new string from our template by replacing the %d with the data referred to by our second argument, idx.

So it seems the problem is in rhe second line where we get our value to display, so %d will be replaced by the value of the variable idx, in this case one, and it will results in what we previously obtained.

Notice that our files are named inflammation-01.csv, so to get it right, we need to indicate to matlab that we want the new value to be formatted using two digits rather than just one, so we modify it in this way.

for idx = 1:12

file\_name = sprintf('inflammation-%02d.csv', idx);

disp(file\_name);

end

% script analyze\_loops.m

for idx = 1:3

% Generate strings for file and image names:

file\_name = sprintf('data/inflammation-%02d.csv', idx);

img\_name = sprintf ('patient\_data-%02d.png', idx);

patient\_data = csvread(file\_name);

disp(['Maximum inflammation: ', num2str(max(patient\_data(:)))]);

disp(['Minimum inflammation: ', num2str(min(patient\_data(:)))]);

disp(['Standard deviation: ', num2str(std(patient\_data(:)))]);

ave\_inflammation = mean(patient\_data, 1);

figure('visible', 'off')

subplot(2, 2, 1);

plot(ave\_inflammation);

ylabel('average')

subplot(2, 2, 2);

plot(max(patient\_data, [], 1));

ylabel('max')

subplot(2, 2, 3);

plot(min(patient\_data, [], 1));

ylabel('min')

print('-dpng', img\_name);

close();

end

So, yes the maxima of these data shows exactly the same ramp and their minima show the same staircase structure, but we have now automated the analysis, and confirmed that all data show the same artefact.

In the case that we have files with not so regular pattern, we could still automate the process by getting a list of files

ls \*.csv

files = dir(‘\*.csv’)

name = files(1).name;

disp(name)

mod\_date = files(3).date;

disp(mod\_date)

Information about the other fields like bytes and isdir is available in the documentation of the dir function:

help dir

Using dir, rewrite the analyze script to analyze all csv files in the current directory.