GEOS F436/636 Beyond the Mouse

Christine (Chris) Waigl
University of Alaska Fairbanks – Fall 2018
Week 4: Control structures II: iteration

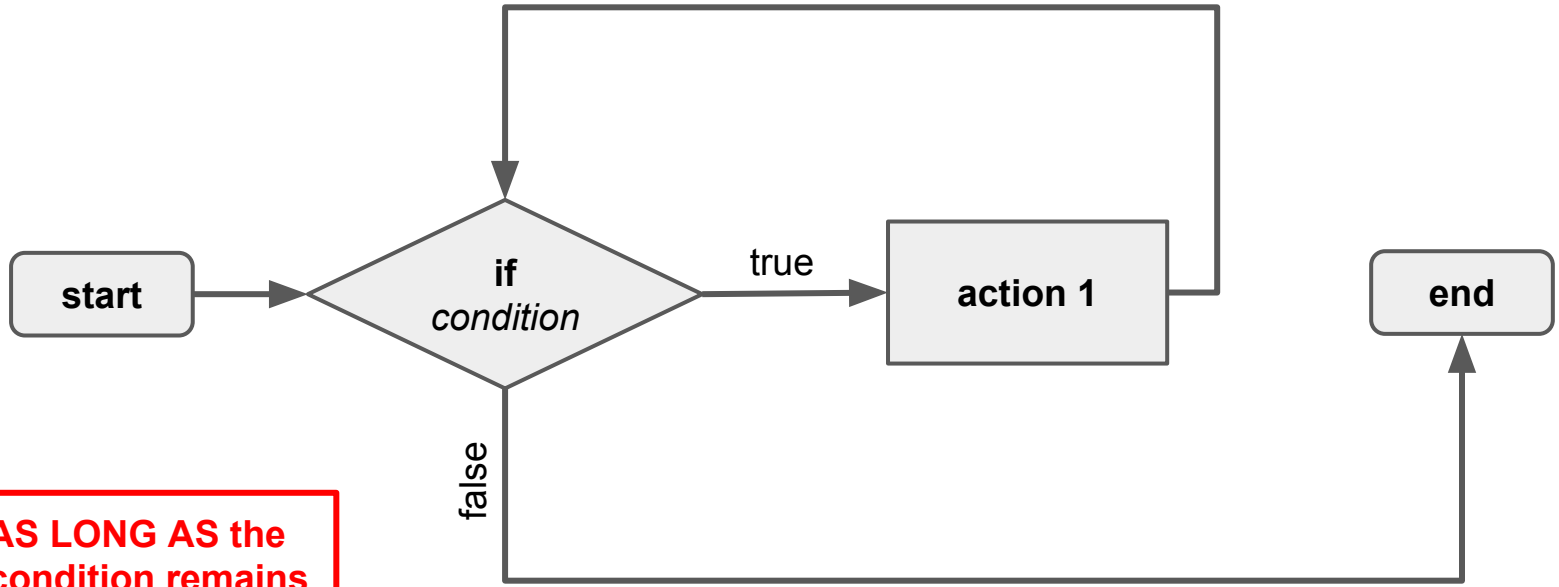
Topics for week 4

- Control structures 2: iteration (= loops)
  - while-end
  - for-end
- (Maybe) consolidation from week 3:
  - reading data from text files
  - structures to store your scientific data: matrices/vectors, structs, cell arrays, map containers
  - a different kind of conditional control structure: try-except

With week 4, we have covered a large amount of ground - enough to enable you to write many useful programs ... if you practice!

Iteration with loops

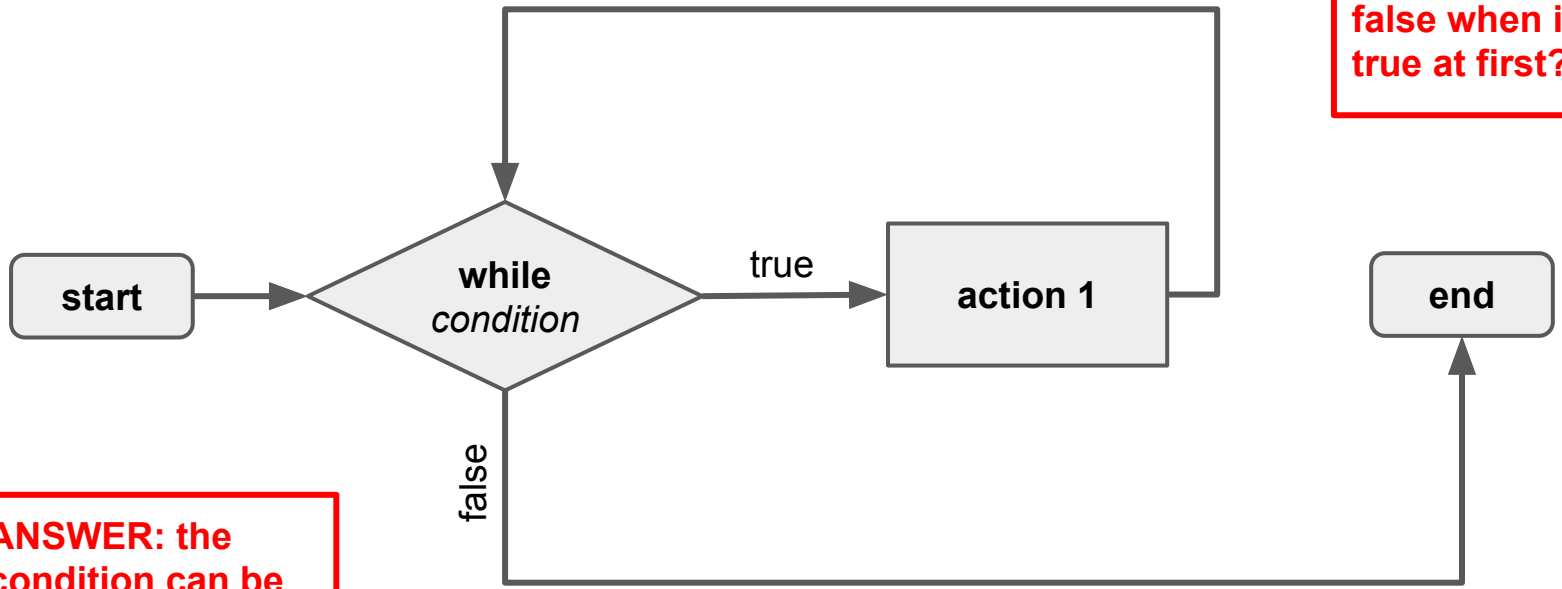
The simplest loop



AS LONG AS the condition remains true, action 1 happens over and over!

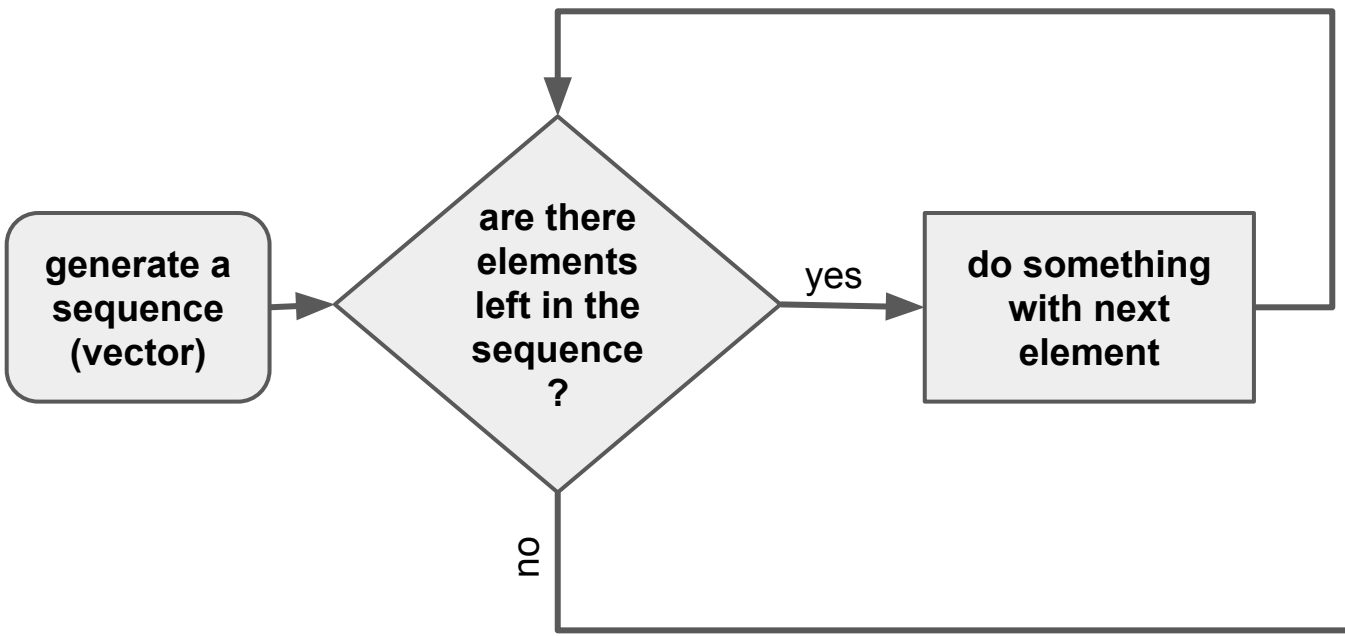
The simplest loop (WHILE loop)

THINK QUESTION: How can the condition become false when it was true at first?



ANSWER: the condition can be changed in action 1! Or randomly! Or with time!

We can make a FOR loop out of a while loop



Some programming languages have a GOTO statement for this "arrow back". Never use it!

MATLAB looping syntax (see also live demo)

```
for n = 1:5
   disp(n)
end
```

```
for v = [1 5 8 17]
   if mod(v, 2) == 0
       continue
   end
   disp(v)
end
```

```
ii = 1
while ii < 10
   disp(ii);
   ii = ii + 1
end
```

```
while 1
    tmp = rand;
    if tmp > limit
        break
    end
    s = s + tmp;
end
```

MATLAB looping notes

- `while` loops are useful when you *don't* know how many times you want to execute an iteration, but know the condition when to stop
- `while` loops are also useful for the paradigm "start looping indefinitely, and stop when a condition is reached". Infinite loops can be broken with CTRL-C
- Every `for` loop can be expressed as a while loop
- `for` loops are useful if you want to loop through a known vector,
- `break` ends loops prematurely
- `continue` jumps to next iteration

GOOD:

```
for element = vector
    do_something(element)
end
```

BAD:

```
N = length(vector):
for ii = 1:N
    do_something(vector(ii))
end
```

Loops can be *nested* (= loops inside loops)

For example, loop through the rows and columns of a matrix:

```
>> A = [1 2 3; 4 3 2; 1 1 1];
>> B = magic(3)                    ← We defined two 3x3 matrices
>> for row = 1:size(A, 1)
>>     for col = 1:size(A, 2)
>>        C(row, col) = A(row, col) * B(row, col)
>>     end
>> end
```

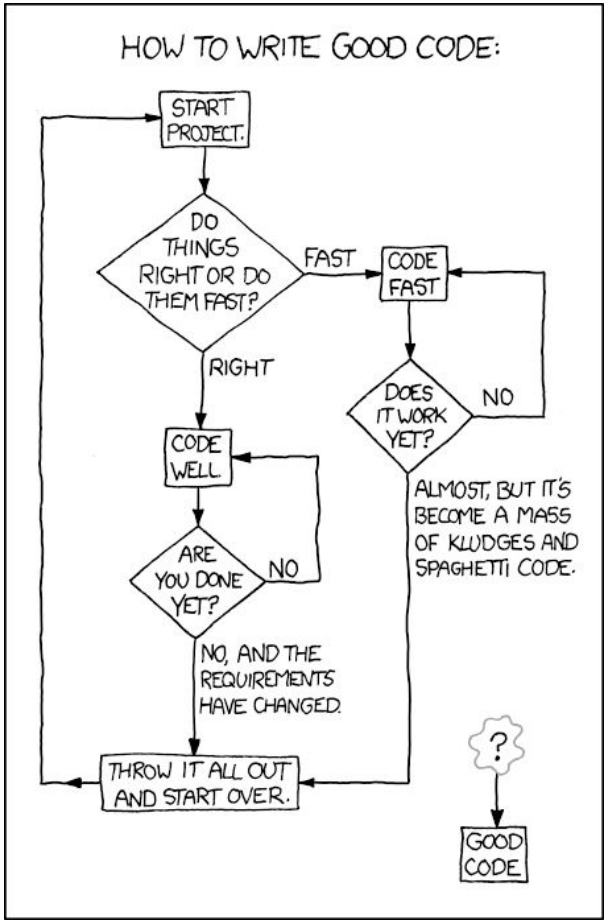
HOWEVER: In many cases you don't need to do this. Use vectorized operations instead: They're shorter, easier to read, less error-prone, and faster. Here:

```
>> C = A .* B                      ← element-wise multiplication
```

Control flow structures are

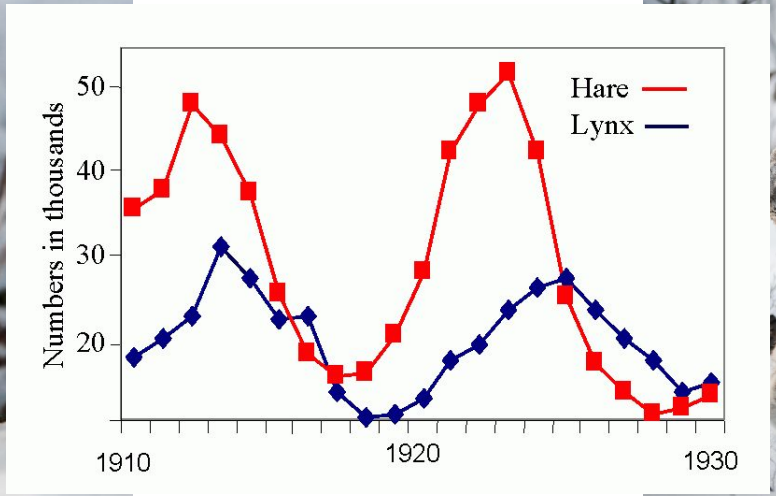
- conditional branching
- iteration (loops).

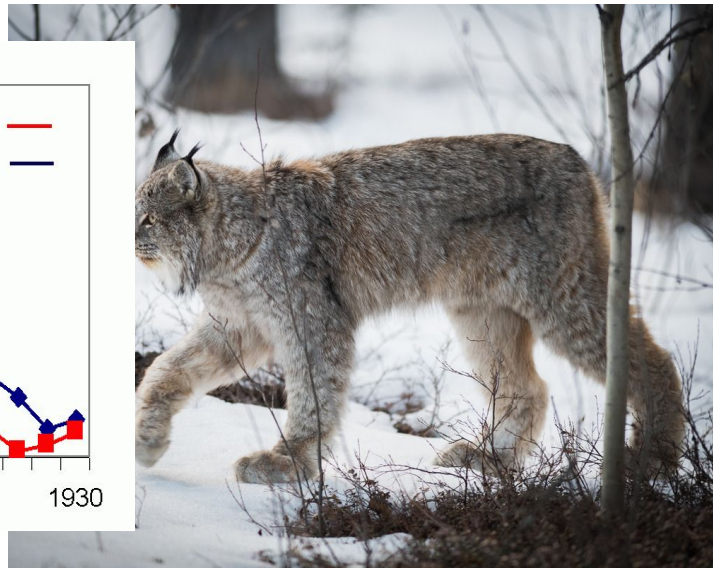
They appear in just about *any* script or program.



source:
https://xkcd.com/844/

What we can do with loops (advanced): Modeling time series, for example: population dynamics.





https://ipmworld.umn.edu/radcliffe-population-ecology

Image source: NPS.

Image source: NPS.

Hypothesis: The population of hares and lynx can be explained by a predator-prey relationship.

Step 1: Design a mathematical model: The number of lynx (hares) at time step t+1 is the number of lynx (hares) at time step t minus the number of animals that died plus number of the animals that were born. Lynx have a constant death rate, but the death rate of hares is proportional to the number of lynx. Also, birth dates are proportional to the availability of food.

```
H(t+1) = H(t)  +  br * H(t)  -  a * H(t) * L(t)

L(t+1) = L(t)  +  c * H(t) * L(t)  -  df * L(t)
```

Often we have `c = a` ("coupling factor") ,

STEP 2 and following: see live example in predprey_discrete.m

Reading data from text files

We have seen two ways of reading data column by column from a delimited text file:

Read data from file (open/read/close file) into a cell array

```
>> fileid = fopen('fname.txt')
>> C = textscan(fileid, formatstring)  ← cell array.
>> fclose(fileid)
>> [var1, ..., varN] = C{:};

OR into a matrix
>> M = dlmread('fname.txt')  ← matrix. Doesn't need fopen
>> var1 = M(:, 1);     ← etc.
```

We can also read data line by line using fgetl, a while loop and a test on ischar ....

```
fileid = fopen('fname.txt')
myline = fgetl(fid);
while ischar(myline)
  disp(myline)
  myline = fgetl(fileid);
end
fclose(fileid);
```

.... or fread to read the whole file in one go:

```
fileid = fopen('fname.txt')
fread(fileid);
fclose(fileid);
```

Don't forget to close the file identifier after use!

Try-except blocks for exception handling

A different conditional: try-catch-end

Sometimes we have the condition "if an error happens... do this".

This is very useful when avoiding to crash your program!

```
try
    a = notaFunction(5,6);
catch
    warning('Problem using function.  Assigning a value of 0.');
    a = 0;
end
```

This is particularly useful when opening files that may not exist.

```
fileid = fopen('fname.txt');
try
    mytext = fread(fileid);
catch
    warning('File does not seem to exist. Skipping this.');
end
fclose(fileid)
```

We have encountered structs and cell arrays to store mixed scientific data.

We can create structs directly with the . operator, or with the struct function:

```
>> data.temperature = [67, 68, 37, 45, 68, 79];
>> data = struct('temperature', [67, 68, 37, 45, 68, 79]);
```

See https://www.mathworks.com/help/matlab/matlab_prog/cell-vs-struct-arrays.html for many examples.

Yet another option are map containers:

```
>> data = containers.Map('temperature', [67, 68, 37, 45, 68, 79]);
>> data('temperature')
```

The general syntax is `containers.Map(keySet, valueSet)`.

Optional reading

- Hahn & Valentine ch. 8.1, 8.2, 6.1 (review), or Attaway ch. 4, ch 5 (review