While

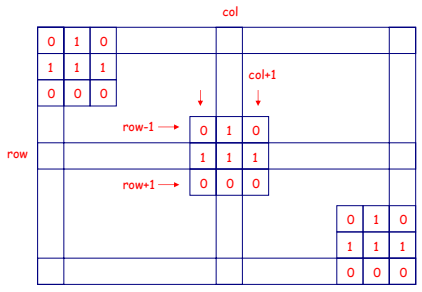
Nested loops and whiles

CS112 Scientific Computation
Department of Computer Science
Wellesley College

Remember me?

- One for loop can be nested inside another for loop. This need arises in many context.
- For example, stepping through the rows and columns of a matrix to perform a computation at every location.

Where's Waldo?



Where's Waldo: The Code

```
function [findRow findCol] = findWaldo (waldo, image)


[numRows numCols] = size(image);
for row = 2:numRows-1
  for col = 2:numCols-1
    if all(waldo == image(row-1:row+1, col-1:col+1))
      findRow = row;
      findCol = col;

    end
  end


end
```

Where's Waldo: The Movie

```
function [findRow findCol] = findWaldo (waldo, image)
findRow = 0;                          % if no Waldo, return 0,0
findCol = 0;
[numRows numCols] = size(image);      % dimensions of the image
for row = 2:numRows-1                 % step through each row
  for col = 2:numCols-1               % step through each column
    if all(waldo == image(row-1:row+1, col-1:col+1))
      findRow = row;                  % Eureka! Save location
      findCol = col;
      break;
    end
  end
  if (findRow ~= 0)                   % if found,
    break;                            % jump out of outer loop
  end
end
```


Bighorn National Forest Needs You

- Nested loops for also useful for performing a computation for every combination of two or more values.
- For example, we create a distance table for the National Park Service.

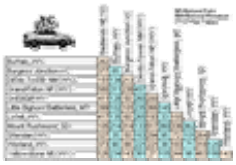


Image that

- We are given the (x, y) coordinates of each city in a table named `coords`.

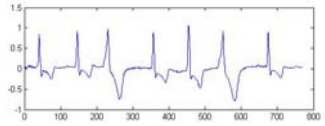
coords	1	2	3	4	5	
1	232	208	097	017	399	x coordinates
2	362	090	055	035	277	y coordinates
	locations					

Creating a distance table

```
function distTable = makeDistanceTable (coords)
% distTable = makeDistanceTable(coords)
% constructs & returns table with the distances
% between every pair of cities whose coordinates
% are stored in the 2 x n input coords matrix

numCities = size(coords, 2);
% step through each possible pair of cities
for cityA = 1:numCities
  for cityB = 1:numCities
    distance = sqrt((coords(1,cityA)- coords(1,cityB))^2 + ...
                    (coords(2,cityA)-coords(2,cityB))^2);
    distTable(cityA, cityB) = distance;
  end
end
```

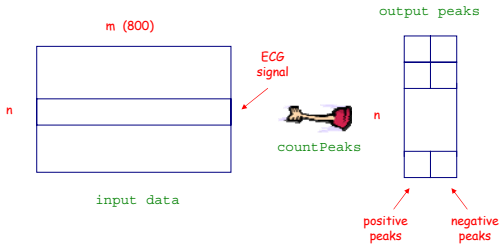
Repeating a repeated computation

- You are given an n x m matrix of data from electrocardiograph (ECG) measurements, where each row is a separate ECG signal:



- Write a function named `countPeaks` with input ECG measurements and output an n x 2 matrix containing the total number of positive and negative peaks in each ECG signal.

Worth a thousand words

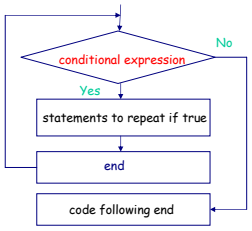


countPeaks

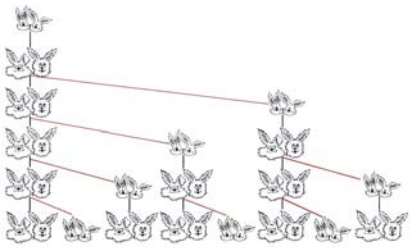
```
function peaks = countPeaks (data)
[rows cols] = size(data);
peaks = zeros(rows, 2);

for row = 1:rows
  for col = 2:cols-1
    if (data(row, col) > 0.5) & ...
       (data(row, col) > data(row, col-1)) & ...
       (data(row, col) > data(row, col+1))
          peaks(row, 1) = 1 + peaks(row, 1);
    elseif (data(row, col) < -0.5) & ...
           (data(row, col) < data(row, col-1)) & ...
           (data(row, col) < data(row, col+1))
            peaks(row, 2) = 1 + peaks(row, 2);
    end
  end
end
```


The `while` statement

while conditional expression
  statements to repeat if conditional expression is true
end



Fibonacci numbers



Finding first Fibonacci number > 100

```
fibo(1) = 1;
fibo(2) = 1;
n = 2;

while (fibo(n) < 100)
   fibo(n+1) = fibo(n) + fibo(n-1);
   n = n+1;
end
disp(['first Fibonacci number > 100: ' num2str(n)]);
```


smallestFactor with a while

```
function factor = smallestFactorWhile (n)
% factor = smallestFactorWhile(n)
% returns smallest integer factor of the input n,
% using a while loop


factor = 2;
while (factor <= sqrt(abs(n))) & (rem(n, factor) > 0)
    factor = factor + 1;
end
if (factor > sqrt(abs(n)))
    factor = n;
end
```