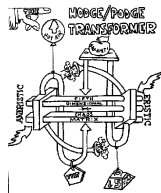


Hodgepodge

"Clumsy mixture of ingredients..."



CS112 Scientific Computation

Department of Computer Science
Wellesley College

The return of Peter Piper

```
function peterPiper (numReps)
% peterPiper(numReps)
% repeats a tongue twister "numReps" times

for count = 1:numReps
    disp('Peter Piper picked a peck of pickled peppers');
end
```



Can we make the `numReps` input *optional*?

Optional input arguments

Inside a user-defined function, `nargin` returns the number of inputs entered when the function was called

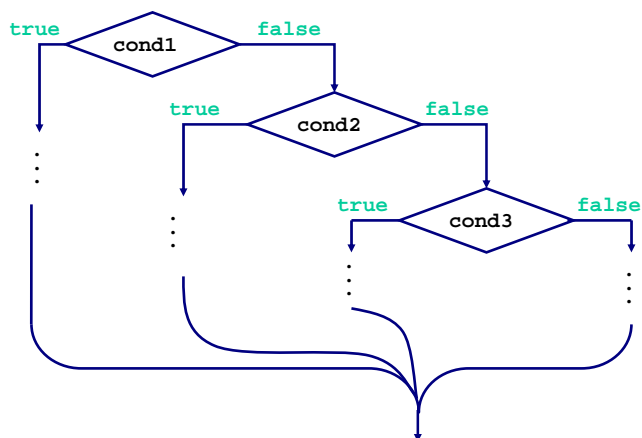
```
function peterPiper (numReps)
% repeats a tongue twister multiple times
% numReps input is optional

if (nargin == 0)
    numReps = 5;
end
for count = 1:numReps
    disp('Peter Piper picked a peck of pickled peppers');
end
```

Hodgepodge 3

A third form of the if statement

```
if (cond1)
    ...
elseif (cond2)
    ...
elseif (cond3)
    ...
else
    ...
end
```



Hodgepodge 4

The elseif clause in action

```
function drawCircle (radius, xcenter, ycenter, properties, width)
% drawCircle(radius, xcenter, ycenter, properties, width)
% draws a circle with specified radius, centered on (xcenter, ycenter)
% with the (optional) properties and width

angles = linspace(0, 2*pi, 50);
xcoords = xcenter + radius * cos(angles);
ycoords = ycenter + radius * sin(angles);
if (nargin == 3)
    plot(xcoords, ycoords, 'b', 'LineWidth', 1);
elseif (nargin == 4)
    plot(xcoords, ycoords, properties, 'LineWidth', 1);
else
    plot(xcoords, ycoords, properties, 'LineWidth', width);
end
axis equal
```

Hodgepodge 5

Looping through a 2-D matrix

```
count = 0;
for row = 1:5
    for col = 1:5
        if (nums(row,col) ~= 0)
            count = count + 1;
        end
    end
end
```

3	7	0	0	6
2	0	5	4	0
6	0	2	1	9
0	3	1	0	7
6	9	0	5	2

nums

But why bother with nested loops here?!?

```
count = sum(sum(nums ~= 0))
```

Hodgepodge 6

Counting peaks

1	4	5	2	7
9	12	7	3	5
0	2	6	8	6
4	3	5	10	2
1	8	4	6	2

two peaks



A *peak* is a value that is larger than its 4 neighbors*

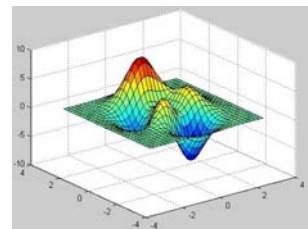
* Don't bother checking locations around the border of the matrix

Hodgepodge 7

How many peaks?

```
function numPeaks = countPeaks (matrix)
% counts the number of peaks in a matrix of numbers, where
% a peak is a value that is larger than its 4 neighbors
```

```
[rows cols] = size(matrix);
numPeaks = 0;
for row = 2:rows-1
    for col = 2:cols-1
        val = matrix(row, col);
        if (val > matrix(row-1, col)) & ...
            (val > matrix(row+1, col)) & ...
            (val > matrix(row, col+1)) & ...
            (val > matrix(row, col-1))
            numPeaks = numPeaks + 1;
        end
    end
end
```




Hodgepodge 8

Bighorn National Forest Needs You

Nested for loops also useful for performing a computation for every combination of two or more values

For example, we can create a *distance table* for the National Park Service



	Battle Mountain NP, SD	Burke, WY	Burgess Junction VC	Devils Tower Nat (WY)	Grand Teton NP (WY)	Greybull, WY	Little Bighorn Battlefield, MT	Lovelock, WY	Mount Rushmore, SD	Shoshone NP, WY	Yellowstone NP (WY)
Battle Mountain NP, SD	282										
Burke, WY	377	85									
Burgess Junction VC	151	134	213								
Devils Tower Nat (WY)	580	311	238	457							
Grand Teton NP (WY)	411	128	50	280	200						
Greybull, WY	399	102	101	235	366	145					
Little Bighorn Battlefield, MT	417	135	60	285	193	33	155				
Lovelock, WY	105	214	314	133	517	380	336	357			
Mount Rushmore, SD	327	35	50	183	294	95	72	101	264		
Shoshone NP, WY	373	90	88	221	221	38	183	71	318	125	
Yellowstone NP (WY)	525	234	143	387	27	105	244	95	453	135	143

Hodgepodge 9

Suppose...

We are given the (x, y) coordinates of each site in a table named **coords**

	1	2	3	4	5	
coords 1	232	208	97	17	399	x coordinates
2	362	90	55	35	277	y coordinates

Locations of national parks

Hodgepodge 10

Creating a distance table

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

numCities = size(coords, 2);
distTable = zeros(numCities, numCities);
% 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
end
```