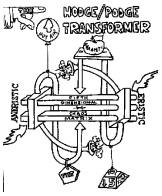
Hodgepodge

"Clumsy mixture of ingredients…"



CS112 Scientific Computation
Department of Computer Science
Wellesley College

Breaking out

Sometimes we'd like to *immediately exit a loop* without stepping through all values of the control variable

We can do this with a break statement:

```
num = 1;
for i = 1:20
    num = 2 * num;
    if (num > 100)
        break;
    else
        disp(['num '  num2str(num)])
    end
end
```

collectGoldenRatios

Write a function named collectGoldenRatios:

(1) input: max number of times to prompt user for hand & forearm values

output: vector of forearm/hand ratios

(2) "for loop" that prompts user for the input number of hand & forearm values, and stores ratios in a vector

(3) stop the loop if the user enters a 0 for the hand length

(4) print message at the end with number of measurements entered

3

```
function ratios = collectGoldenRatios (ntimes)
% prompts user for "ntimes" hand & forearm lengths and stores
% ratios in a vector. The loop stops if the user enters a 0 for
% the hand length. Number of entries is printed at the end
ratios = [];
for index = 1:ntimes
    hand = input('Enter a hand length: ');



    forearm = input('Enter a forearm length: ');
    ratios(index) = forearm/hand;
end
disp(['You entered ' num2str(length(ratios)) ' ratios']);
```

4

Tip on debugging loops

```
% calculate 10! and print the result
factorial = 0;
for num = 10:1:1
    disp('inside loop');
    factorial = factorial * num;
    disp(['num: ' num2str(num) 'factorial : ' num2str(factorial)]
end
disp(['10! = '  num2str(factorial)]);
```

Print statements are your friends!

5

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
```

How can we make the numReps input ***optional***?

6

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
```

7

Multiple optional inputs

```
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 input properties and width

angles = linspace(0, 2*pi, 50);
xcoords = xcenter + radius * cos(angles);
ycoords = ycenter + radius * sin(angles);
plot(xcoords, ycoords, properties, 'LineWidth', width);
```

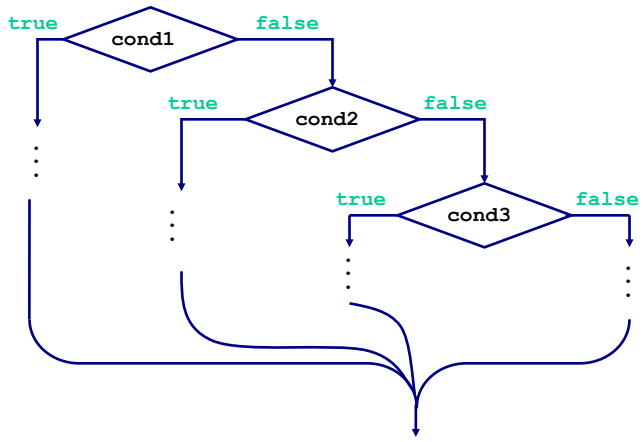
Suppose we want require an input radius, but allow the other 4 inputs to be *optional?*

Starting hint: For what values of nargin do we need to set a default value for width? For properties?

8

A third form of the if statement

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



9

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
```

10

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))
```

11

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*

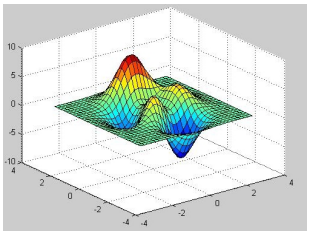
* Don't bother checking locations around the border of the matrix

12

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
```

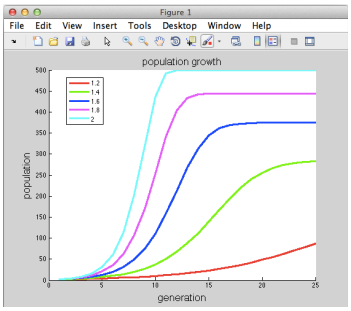


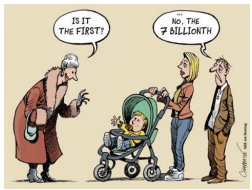
Simulating population growth

Goal: define a function that generates a figure with curves for different rates of population growth over multiple generations, using the *logistic growth* model for population growth:

$$p_{t+1} = r * p_t * (K - p_t)/K$$

p_t: current population
p_{t+1}: population in the next generation
r: growth rate
K: carrying capacity





Guidelines & tips

Define a function named popGrowth with four inputs:
- vector of growth rates to simulate
  - (default [1.2 1.4 1.6 1.8 2.0])
- initial population (default 2)
- number of generations (default 25)
- carrying capacity (default 1000)

For each growth rate:
- create a vector to store the population for each generation, and store initial population in the first location of the vector
- for each new generation, apply the formula to calculate the new population size and store it in the vector
- plot the populations for this growth rate

Add figure embellishments at the end

15