% prodby2.m
% Patrick Utz, 2/16/18, 6.1

% Problem: Write a function prodby2 that will receive
% a value of a positive integer n and will calculate and
% return the product of the odd integers from 1 to n (or from
% 1 to n?1 if n is even). Use a for loop. Test your function by
% calling it using the following values of n: 1, 2, 3, 9, 10,
% respectively. Attach your results.

% Variables: product = the outcome of the function, intVec = the vector
% created using n integers, k = used for reference in the for loop

```
function product = prodby2(n)
% prodby2 calculates the product of
% Format of call: prodby2( number of integers )
% Returns product of all odd numbers in vector
product = 1;
intVec = 1:n;
if n < 1
   fprintf('Invalid value. Please enter an integer.')
elseif rem(numel(n),2) == 0
   for k = 1:2:n-1
      product = product*intVec(k);
   end
else
   for k = 1:2:n
      product = product*intVec(k);
   end
end
```

```
>> prodby2(1)
ans =
    1
>> prodby2(2)
ans =
    1
>> prodby2(3)
ans =
    3
>> prodby2(9)
ans =
  945
>> prodby2(10)
ans =
  945
```

% approx_inv_e.m
% Patrick Utz, 2/16/18, 6.2

The inverse of the mathematical constant e can be approximated as $\frac{1}{e} \approx \left(1 - \frac{1}{n}\right)^n$, where n is a positive integer.

% Problem: Create a MATLAB function named approx_inv_e to
% find the minimum positive integer n needed for the approximated
% value to have a specified accuracy. The function should have one
% input argument as the maximum tolerable difference between the real
% value of 1/e and its approximation, and the output argument is the
% minimum value of n required for achieving the required accuracy.
% Your function should still print the approximated value and the actual
% value, but not the number n, which is returned in the output argument.
% Test your function using 0.1, 0.01 and 0.001 as the input argument,
% respectively, and attach your results

% Variables: minInt = the test variable for the minimum integer that
% returns the desired accuracy, n = the output minimum integer
% of the function maxDiff = the input argument specifying
% the max difference desired, approx = the approximated value, actual =
% the actual value

```
function n = approx_inv_e(maxDiff)
% approx_inv_e calculates the minimum integer that yields an approximate
% value of 1/e within a certain tolerable difference
% Format of call: approx_inv_e( desired tolerable difference )
% Returns the desired integer value
minInt = 1;
while ( ((1/exp(1)) - (1-(1/minInt))^minInt) ) > maxDiff
   minInt = minInt + 1;
end
n = minInt - 1;
approx = (1-(1/n))^n;
actual = (1/exp(1));
fprintf('The approximated value is %f and the actual value is %f\n', approx, actual)

>> approx_inv_e(.1)
The approximated value is 0.250000 and the actual value is 0.367879
ans =
    2
>> approx_inv_e(.01)
The approximated value is 0.357417 and the actual value is 0.367879
ans =
   18
>> approx_inv_e(.001)
The approximated value is 0.366877 and the actual value is 0.367879
ans =
  184
```

% find_max_N.m
% Patrick Utz, 2/16/18, 6.3

$$2^2 - \frac{3}{1} + 4^2 - \frac{5}{3} + \ldots + (2N)^2 - \frac{2N+1}{2N-1} \leq M$$

% Problem: Create a MATLAB function named find_max_N to find the
% largest integer N that satisfies the following inequality, where
% M is the upper bound. The function should take one input argument ?
% the upper bound value, and output one single output argument - the
% largest N value. Use the following examples: M=0; 1; 10; 100 and
% 1000 to test your function. Attach the results.

% Variables: N = output the largest integer, M = input the upper bound
% value, intN = place holder for N, total = used to calculate
% total of the series

```
function N = find_max_N(M)
% find_max_N calculates the max integer that causes the series to be
% less than the upper bound inputted
% Format of call: find_max_N( desired upper bound )
% Returns the desired integer value
intN = 1;
total = 0;
if M < 1
   fprintf('Please enter an upper bound greater than 1\n');
else
   while round(total) <= M
      for k = 1:intN
         total = total + (2*k)^2 - ((2*k+1)/(2*k-1));
      end
   intN = intN + 1;
   end
   N = intN - 2;
end
```

```
>> find_max_N(0)
Please enter an upper bound greater than 1
>> find_max_N(1)
ans =
    1
>> find_max_N(10)
ans =
    1
>> find_max_N(100)
ans =
    3
>> find_max_N(1000)
ans =
    6
```

% ismagic.m
% Patrick Utz, 2/16/18, 6.4

% Problem: A magic matrix is a square matrix (with N rows and
% N columns) constructed from the integers 1 through N^2, and the
% sums along each row, each column, and the two diagonal lines
% are all equal. Write a MATLAB function called ismagic
% that accepts a square matrix as input and determine if this
% matrix is magic. Its output is ?0? (logic false) if the matrix
% is not a magic matrix; and ?1? (logic true) if it is a magic matrix.

(logic true) if it is a magic matrix. Test your function using $\begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$ and then a randomly generated 4-by-4 integer matrix as input. Attach your results.
Note that not only do you need to check if the sums along rows, columns, and diagonal lines are equal, but also need to check if all the elements are from 1 to N^2.

% Variables: result = logical value of whether the matrix is magic
% or not, sqMatrix = input square matrix, N = length of the matrix,
% size = size of the matrix, notSquare = temp value to check if
% it is not a square matrix, sumCheck = matrix that holds sums of the
% columns and rows and diaganols, totalCol = temp total variable for
% the summing loop, offset = used to offset index of matrix storing
% summed values, diag = identity matrix to text diaganols, copy =
% copy of sumCheck, check1 = final reference matrix to check if
% the matrix is magic

```
function result = ismagic(sqMatrix)
% ismagic calculates whether the inputted square matrix is magic
% or not
% Format of call: ismagic( square matrix )
% Returns a logical false if matrix is not magic and logical true
% if it is

N = length(sqMatrix);
size = numel(sqMatrix);
notSquare = 0;
sumCheck = [];
totalCol = 0;
for k = 1:size
   if sqMatrix(k) > N^2
      notSquare = 1;
   end
end
if notSquare ~= 0
   result = logical(0);
else
   for i = 1:N
      for j = 1:N
      totalCol = totalCol + sqMatrix(i,j);
      end
      sumCheck(i) = totalCol;
      totalCol = 0;
   end
```

```
    offset = length(sumCheck);
    for i = 1:N
        for j = 1:N
        totalCol = totalCol + sqMatrix(j,i);
        end
        sumCheck(offset + i) = totalCol;
        totalCol = 0;
    end
    offset2 = length(sumCheck);
    diag = eye(N,N);
    sumCheck(offset2 + 1) = sum(sum(sqMatrix.*diag));
    sumCheck(offset2 + 2) = sum(sum(sqMatrix.*flip(diag)));

    sameTester = sumCheck(1);
    copy = sumCheck;
    check1 = copy - sameTester;
    if (check1 | 0) == 0
        result = logical(1);
    else
        result = logical(0);
    end
end


>> ismagic([16 2 3 13; 5 11 10 8; 9 7 6 12; 4 14 15 1])
ans =
  logical
   1
>> x = randi([1,5],4,4);
>> x
x =
     5     4     5     5
     5     1     5     3
     1     2     1     5
     5     3     5     1
>> ismagic(x)
ans =
  logical
   0
>>
```