1)

```bash

% Initializing the variables for the calculation

m = input('Enter a real number: ');

p = input('Enter a value greater than zero: ');

x = 1;

% Generating if statement to make sure values work and then calculate

if isreal(m) && p > 0

    y = (1/(sqrt(2\*pi)\*p)) \* exp(-(1/2)\*((x-m)/p)^2);

else

    disp(['Make sure your values meet the parameters.'])

end

% Checking my answer with the built-in MATLAB function

normpdf(1,0,2)

```

2)

```bash

% Initializing the variables for egg-cooking calculation

% Small egg mass

MS = 47;

% Large egg mass

ML = 67;

% Density

p = 1.038;

% Specific heat

c = 3.7;

% Thermal conductivity

K = 0.0054;

% Temperature of water

T\_w = 100;

% Temperature of the yolk

T\_y = 70;

% Initial egg temperature

T\_0 = input('Enter the initial temperature of the egg in Celsius: ');

% Asking user to give egg type

Egg = input('Is the egg small or large? ', 's');

% Creating the if statement to determine how long to cook egg

if Egg == 'small' || Egg == 'Small'

    t = ((MS^(2/3)\*c\*p^(1/3))/(K\*pi^2\*((4\*pi)/3)^(2/3)))\*log(0.76\*((T\_0-T\_w)/(T\_y-T\_w)));

elseif Egg == 'large' || Egg == 'Large'

    t = ((ML^(2/3)\*c\*p^(1/3))/(K\*pi^2\*((4\*pi)/3)^(2/3)))\*log(0.76\*((T\_0-T\_w)/(T\_y-T\_w)));

else

    disp('Please check for errors in your entries.');

end

% Displaying the time

disp(['It takes ' num2str(t) ' seconds to cook the egg.'])

```

3)

```bash

function outstruct = getCart(inputPolarStruct)

% asking the user to give the r and phi values for the structure

inputPolarStruct.r = input('Enter the value for r: ');

inputPolarStruct.phi = input('Enter the value for phi in radians: ');

% making sure the entries have the required field then finding x and y

if isfield(inputPolarStruct, 'r') && isfield(inputPolarStruct, 'phi')

    x = inputPolarStruct.r \* cos(inputPolarStruct.phi);

    y = inputPolarStruct.r \* sin(inputPolarStruct.phi);

else

    disp('Please check your values for r and phi.')

end

% making the output a structure with fields x and y

outstruct.x = x;

outstruct.y = y;

end

function outstruct = getPolar(inputCartesianStruct)

% asking the user to give the x and y values for the structure

inputCartesianStruct.x = input('Enter the value for x: ');

inputCartesianStruct.y = input('Enter the value for y: ');

% making sure the entries have the required field then finding r and phi

if isfield(inputCartesianStruct, 'x') && isfield(inputCartesianStruct,'y')

    r = sqrt(inputCartesianStruct.x.^2 + inputCartesianStruct.y.^2);

    phi = atan(inputCartesianStruct.y./inputCartesianStruct.x);

else

    disp('Check your values for x and y.')

end

% making the output a structure with fields r and phi

outstruct.r = r;

outstruct.phi = phi;

end

```

4)

```bash

function totalSize = sizeDirectory(path)

% using the built-in function, dir, to get information about variable, path

s = dir(path);

% using built-in function, sum to sum the array of bytes

totalSize = sum([s.bytes]);

% displaying the total bytes in the directory path

disp(['The total number of bytes is ' num2str(totalSize)])

end

```

5)

```bash

function fib()

% asking the user to enter the non-negative integer

n = input('Enter a non-negative integer or ''stop'' if you are done. ','s');

% nested if blocks call the getFib() function under the right conditions

if strcmp(n, 'stop') || strcmp(n, 'Stop')

    return

else

    n = str2double(n);

    if isreal(n) && n == round(n) && n >= 0

        disp(['The ' num2str(n) ' term of the Fibnacci sequence is '...

            num2str(getFib(n))]);

        fib()

    else

        disp('Make sure you entered a non-negative integer.')

    end

end

end

function y = getFib(n)

% creating the recursive function to calculate the nth term of fib sequence

% note: the checks on the user inputs are in the fib.m function

    if n<2

        y = n;

    else

        y = getFib(n-1) + getFib(n-2);

    end

end

```

6)

```bash

function Area = evalTriangle()

% creating a 3 by 3 cell array and asking the user to fill the array

vertices = cell(3,3);

vertices{1} = input('Enter x1: ');

vertices{2} = input('Enter y1: ');

vertices{3} = input('Enter x2: ');

vertices{4} = input('Enter y2: ');

vertices{5} = input('Enter x3: ');

vertices{6} = input('Enter y3: ');

% calling the getTriangleArea.m function to do the area calculation

Area = getTriangleArea(vertices);

disp(['The area of the the triangle is ', num2str(Area)])

end

function A = getTriangleArea(vertices)

% assigning the coordinates from the vertices input given in evalTriangle.m

x1 = vertices{1};

y1 = vertices{2};

x2 = vertices{3};

y2 = vertices{4};

x3 = vertices{5};

y3 = vertices{6};

% calculating the area of the triangle with the given vertices

A = .5 .\* abs((x2 .\* y3)-(x3 .\* y2)-(x1 .\* y3)+(x3 .\* y1)+...

    (x1 .\* y2)-(x2 .\* y1));

end

```

7)

```bash

function out = PrimeOutput()

% asking the user to give a number to check

n = input('Enter a number greater than 1 to see if it prime: ');

% makes sure the value entered is a number greater than 1

if isnumeric(n) && (n > 1)

    out = isPrime(n,n-1); % calls isPrime w/ intial values n and n-1

    if out == 0

        disp('The value is not prime.')

    else

        disp('The value is prime.')

    end

else

    disp('Please enter a number greater than 1')

end

end

function y = isPrime(n,i)

%{

checking if the divisor got from n-1 down to i;

if mod(n,i) was not equal to zero anywhere from n-1 to i = 1, it can be

concluded the number is prime

%}

if i == 1

    y = true;

    return

elseif mod(n,i) == 0 % if mod(n,i)=0, the number is divisible and not prime

    y = false;

    return

else

    y = isPrime(n,i-1); % calls function again, reducing the divisor by 1

end

end

```

8)

```bash

function functionHandle = genFunc(varargin)

function out = genFuncOut(a,b,c)

% using the values of a,b,c calculated below to generate a function handle

    out = @(x) (a .\* (x .^ 2) + b .\* x + c);

end

% generating the switch case to set a,b,c based on the number of arguments\

% also, checking if the values are real numbers in each case

    switch nargin

        case 0

            disp('No inputs given')

            a = 0;

            b = 0;

            c = 0;

        case 1

            if isnumeric(varargin{1}) && isreal(varargin{1})

            disp('One input given')

            a = varargin {1};

            b = 0;

            c = 0;

            else

                disp('Make sure you have entered real, numeric values.')

            end

        case 2

            if isnumeric(varargin{1}) && isreal(varargin{1}) && ...

                    isnumeric(varargin{2}) && isreal(varargin{2})

            disp('Two inputs given')

            a = varargin{1};

            b = varargin{2};

            c = 0;

            else

                disp('Make sure you have entered real, numeric values.')

            end

        case 3

            if isnumeric(varargin{1}) && isreal(varargin{1}) && ...

                    isnumeric(varargin{2}) && isreal(varargin{2}) && ...

                        isnumeric(varargin{3}) && isreal(varargin{3})

            disp('Three inputs given')

            a = varargin{1};

            b = varargin{2};

            c = varargin{3};

            else

                disp('Make sure you have entered real, numeric values.')

            end

        otherwise

            error('Make sure you did not enter too many input arguments.')

    end

% calling genFuncOut to create function handle with values a,b,c

functionHandle = genFuncOut(a,b,c);

end

```