1 Mind the gap

```
function p = prime_gap(n,d)
% Returns the prime numbers separated by a gap d and smaller than n.
% p is an array which contains pairs of primes in order.

x = primes(n);      % get a list of primes smaller than n
m = length(x) - 1;
p = [];

for i = 1:m
   if (x(i+1)-x(i)) == d  % check distance for consecutive pairs
      p = [p, x(i), x(i+1)];
   end
end
```

2 Frequency of letters

```
function n = letter_freq(s)
% Returns the number of occurences of letters in a strings. The index i
% of the result array n, contains the occurence of the character
% char('a'+i-1) The length of the output array should be equal to
% 'z'-'a'+1 (=26).

n = zeros(1,'z'-'a'+1); % creates an array filled with zeros

s = lower(s);
for c = s
   if c <= 'z' && c >= 'a',
      n(c-'a'+1) = n(c-'a'+1) + 1;
   end
end
```

3 Polynomials

3.a Evaluation

```
function y = poly_eval(p, x)
% Evaluates the polynomial at x. The polynomial coefficients are
% provided as an array p, in increasing order of the power of x.

y = 0;

for i = 1:length(p)
   y = y + x^(i-1) * p(i);
end
```

3.b Multiple Points

```
function y = poly_eval_array(p, x)
% Evaluate a polynomial at multiple values provided by array x.
% The result, y, is another array.
%
%   Example: If we want to evaluate p(x) = 3x^2 + 4x + 5 at x = [2 3] we
%       should call the function as follows:
%
%    y = poly_eval_array([5 4 3], [2 3])  % result should be [25 44]
%

n = length(x);      % number of points to evaluate
y = zeros(1,n);     % result array
for i = 1:n
   y(i) = poly_eval(p, x(i));
end
```

3.c Multiplication

```
function p = poly_multiply(p1, p2)
% Multiply two polynomials represented with their coefficients and
% returns the coefficients of the product polynomial.

n1 = length(p1);
n2 = length(p2);
n  = n1 + n2 - 1;

p = zeros(1,n);

for i = 1:n1
   for j = 1:n2
      p(i+j-1) = p(i+j-1) + p1(i)*p2(j);
   end
end
```

4 Astroid

```
% Fill in this script to accomplish the following tasks.
% Task0: Write a couple of comment lines to describe the purpose of
% the script

% In this script we will estimate the area of an astroid curve
% using a numerical experiment. After generating uniformly
% distributed random points inside a square region, we count
% the number of points falling inside the astroid curve. The ratio
% of number of points inside to the total number can be used to
% compute the area.
```

```
% Task1: Ask the user to provide an input for N
N = input('Enter the number of points: ');  % Take user input

% Task2: Generate N random numbers between (-1,1) and store in array x
x = 2*rand(N,1)-1;   % Generate random x coordinates

% Task3: Generate N random numbers between (-1,1) and store in array y
y = 2*rand(N,1)-1;   % Generate random y coordinates

% Task4: Create a loop and count the number of points inside the astroid
n = 0;                      % Count the number of points
for j=1:N
   if abs(x(j))^(2/3) + abs(y(j))^(2/3) < 1   % NOTICE abs!! Why?
      n = n + 1;
   end
end

% Task5: Compute the estimated area of the astroid
area = 4*n/N;                % Compute the area

% Task6: The area of the astroid is 3*pi/8. Compute the percentage error.
err = abs(area-3*pi/8)/(3*pi/8);   % and the relative error

fprintf('Percentage error: %6.2f\n', err*100);
```

5 Soccer Ball Toss

5.a Ray Triangle Intersection

```
function hit = ray_intersects_triangle(v0, vA, vB, vC)

% There are standards ways to solve this problem, here is
% an intuitive one.

hit = 0; % assume it doesn't hit the plane
v0 = v0/norm(v0); % compute the unit ray vector

AB = vB-vA; % vector A to B
AC = vC-vA; % vector A to C

n = cross(AB, AC); % find normal to the plane defined by ABC
n = n/norm(n); % unit normal

% check if the ray is parallel to the plane
if abs(n'*v0)<eps, return ; end

beta = (n*vA')/(n*v0');
if beta < 0, return; end % check if it hits in reverse

vP = beta * v0;
PA = vA-vP;
PB = vB-vP;
```

```
PC = vC-vP;

% if P falls outside of the triangle, then the
% sum of the areas of the triangles PAB, PAC, PBC
% must be greater than that of ABC
PAB = norm(cross(PA,PB))/2;
PAC = norm(cross(PA,PC))/2;
PBC = norm(cross(PB,PC))/2;
ABC = norm(cross(AB,AC))/2;

if (PAB + PAC + PBC- ABC)/ABC> 1e-10, return;
else hit = 1; return; end
```

5.b Probability Estimates

```
load('triangulation.mat');

N = 10000;

tic

HB = 0;
HW = 0;

for n = 1:N
   ray = rand_isotropic(3);
   hit = 0;

   c = 0;
   while ~hit && c < 1080
      vA = htria(c+1:c+3);
      vB = htria(c+4:c+6);
      vC = htria(c+7:c+9);
      hit = ray_intersects_triangle(ray, vA, vB, vC);
      c = c + 9;
   end

   if hit,
       HW = HW + 1;
   else

   c = 0;
   while ~hit && c < 540
      vA = ptria(c+1:c+3);
      vB = ptria(c+4:c+6);
      vC = ptria(c+7:c+9);
      hit = ray_intersects_triangle(ray, vA, vB, vC);
      c = c + 9;
   end
      if hit, HB = HB + 1; else error('uppps!'); end
   end
```

```
end

toc
assert(HW+HB == N);

PW = HW/(HW+HB)
PB = HB/(HW+HB)
```