```
function particles = accretion(P, N, s, w, e)
% ACCRETION - Biased random walk of falling and accreting particles
% particles = accretion(P,N,s,w,e) takes N many particles beginning at a
% starting point y = 99, x = [1,99] determined by P, and 'drops' them
% until they hit the bottom, or accrete onto another fallen particle. This
% fall is a biased random walk. A particle will move; south (-y) at
% probability s, west (-x) at probability w, and east (+x) at
% probability e. The west and east boundaries loop, x = 0 equivalent to
% x = 99, x = 100 equivalent to x = 1.
% The function outputs particles, a 2xN array of final positions of all
% particles.
% Assumes P == 1 or 'rand'. Assumes s + w + e == 1.

%% Initialising

domain = 99; % Set by the question, but it's useful to have here
p = cumsum([s,w,e]); % Cumulative sums of the probabilities
boundary = false([domain,domain+1]); % Records the final position
%^% NOTE!! For boundary; dim=1 is the x-coordinate, dim=2 is y + 1
boundary(:,1) = 1; % Adding a collision boundary along y = 0 (y+1=1)
% boundary is the collision mask, particles will check against this to see
% if a position they intend to move into is already occupied. There is
% collision set along the bottom of the domain (so particles do not fall
% endlessly), and once a particle reaches its final position, it will be
% added to the collision mask
particles = zeros([2,N], 'int8'); % x=row1, y=row2; dim=2 is particle no.
particles(2,:) = domain; % Setting y = 99 for all particles
% particles records the position of each N particles, in terms of x and y
% coordinates. As such, by the end of the function, particles contains the
% final positions.

if P == 1 % Applying start conditions for x
    particles(1,:) = 50; % All particles begin at x = 50
elseif isequal(P,'rand')
    particles(1,:) = randi([1,domain], 1,N); % x = N random int from 1-99
else
    error("Input error. P can only be 1 or 'rand'.")
end

%% Running simulation

for i = 1:N % Looping over each particle
    % Initialising while loop
    rest = false; % There are valid moves for particle i
    south = false; west = false; east = false; % Particle hasn't moved yet!

    safety_switch = 0;
    while rest == false % Looping until particle i can no longer move.
        %-------SAFETY SWITCH---------------]
        safety_switch = safety_switch + 1; % ]
        if safety_switch > 10^6            % ]
```

```
        safety_switch = 0, break       % ]
    end %---SAFETY SWITCH---------------]
    % Because while loops can get out of hand; this will break the loop
    % if it iterates too many times (i.e., if it's not actually
    % progressing towards the end of the loop), and will output
    % safety_switch = 0 since it ends in "," not ";"

    u = rand; % Determining a direction
    if u <= p(1) || (west && east)
    % MOVING SOUTH
    % If there are collisions to the east and west, ignore the
    % randomiser and check south.
        target = particles(:,i) + int8([0; -1]);
        south = true;
    elseif (u <= p(2) && ~west) || east
    % MOVING WEST
    % If there is a collision west, ignore the randomiser and skip. If
    % there is a collision east (and you already skipped south), ignore
    % the randomiser and check west.
        target = particles(:,i) + int8([-1; 0]);
        west = true;
    elseif u <= p(3) || west
    % MOVING EAST
    % If there is a collision to the west, ignore the randomiser and
    % check east.
        target = particles(:,i) + int8([1; 0]);
        east = true;
    end

    wrap_left2right = domain * int8(target(1) < 1);
    wrap_right2left = domain * int8(target(1) > domain);
    target(1) = target(1) + wrap_left2right - wrap_right2left;
    % If x <= 0, +domain (to loop back around to x <= domain)
    % If x > domain, -domain (to loop back around to x >= 1)
    % We're only applying this to x because;
    %   1. There is no +y movement
    %   2. We want y = 0 to exist so it triggers the boundary there
    collision = boundary(target(1), target(2)+1);

    if collision && south % Collides with something south
        boundary(particles(1,i), particles(2,i)+1) = true;
        rest = true; % There are no valid moves for particle i
    elseif ~collision % No collisions
        % If there is a collision, particle position is not updated and
        % that direction will be set =true at the start of the loop.
        south = false;
        west = false;
        east = false;
        particles(:,i) = target; % Free to move!
    end
end
```

```
end
end
```