Matlab Solutions

## Solutions to Homework 8[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

### Problem sparse\_array\_out

function success = sparse\_array\_out (A, filename) fid = fopen (filename,'w+'); success = fid>=0; if ~success error ('Error opening file %s\n',filename) end [r,c,v] = find(A); % locations and values of non-zero elements [nr,nc] = size(A); % dimensions of A nze = length(v); % number of non-zero elements fwrite (fid, [nr,nc,nze], 'uint32'); % global data for k = 1:nze % for each non-zero element ... fwrite (fid, [r(k),c(k)], 'uint32'); % row-column index fwrite (fid, v(k), 'double'); % value end fclose(fid); end

### Problem sparse\_array\_in

function A = sparse\_array\_in (filename) A = []; fid = fopen(filename,'r'); if fid<0 error ('Error opening file %s\n',filename) end x = fread(fid,3,'uint32'); % global data (nr, nc, nze) A = zeros(x(1),x(2)); % initialize new nr x nc matrix for k=1:x(3) % for each non-zero element ... z = fread(fid,2,'uint32'); % row, column A(z(1),z(2)) = fread(fid,1,'double'); % value end fclose(fid); end

### Probel letter\_counter

function n = letter\_counter(fname) fid = fopen(fname,'r'); if fid < 0 n = -1; else x = fread(fid,inf,'char'); % read entire file x = x(isletter(char(x))); % pick the letters n = length(x); % count them fclose(fid); end end

### Probel letter\_counter (alternative solution)

A shorter variant

function n = letter\_counter(fname) n = -1; fid = fopen(fname,'r'); if fid >= 0 n = sum(isletter(fread(fid,inf,'char=>char'))); fclose(fid); end end

### Problem saddle

function s = saddle(M) [r c] = size(M); s = []; if r > 1 cols = min(M); % find the min value in each column else cols = M; % vector is a special case, min would give a single value end if c > 1 rows = max(M'); % find the max value in each row else rows = M; % vector is a special case, max would give a single value end for ii = 1:c % visit each column for jj = 1:r % and each row, that is, each element of M if M(jj,ii) == cols(ii) && M(jj,ii) == rows(jj) % if both conditions hold s = [s; jj ii]; % saddle point! Let's add it! end end end end

### Problem prime\_pairs

function p = prime\_pairs(n) if isprime(2+n) % many times the answer is 2 p = 2; elseif rem(n,2) % if not, and n is odd, no such prime exists p = -1; else for p = primes(1e5) % check all primes smaller than 100,000 if isprime(p+n) % if p+n is prime return; % found it! Return immediately end end p = -1; % none found (btw, we never get here) end end % It turns out that for n-s smaller than 100,000 that are even, there is % always a pretty small such prime. In fact, the largest is 227. % So we could use primes(300) instead of primes(1e5) to make this even % faster. Also, the for-loop would be slow, if we did not check for even n-s, % since it would need to go through all primes smaller than 100,000 to % realize that no solution exists. So, handling the first two cases (p is 2 % and n is odd) separately makes the function very efficient.

### Problem prime\_pairs (alternative solution)

No loop at all. This illustrates yet again that there is always a MATLAB built-in function for almost anything reasonable...

function p = prime\_pairs(n) allp = primes(1e5+n); % Get all primes up to max value + n p = intersect(allp,allp+n); % Get which values are prime when n is added if isempty(p) % Check to see if there are any such values p = -1; else p = p(1)-n; % If so, subtract off the n to get the smaller value of the prime pair end end % Elegant solution, but because it does not check for odd n and because it % always handles the entire vector of primes even though the answer, if it % exists, is small, it is about 4x slower than the for-loop version above % even though the built-in function intersect is very fast.

### Problem bowl

function score = bowl(balls) index = 0; % index into balls first = 1; % multiply next ball second = 1; % multiply ball after next score = 0; % cummulative sum if sum(balls > 10 | balls < 0) > 0 % single hit must be between 0 and 10 inclusive score = -1; % error! return; end for ii = 1:10 % first ten frames index = index + 1; % take next ball if index > length(balls) % not enough balls score = -1; % error! return; end score = score + first \* balls(index); % count score including extra from previous strike or spare first = second; % move multiplier value from second to first second = 1; % reset multiplier for the ball after next to 1 if balls(index) == 10 % strike first = first + 1; % so next counts extra second = 2; % and so is the one after next continue; % go to next frame, there is no second ball in this one end index = index + 1; % take next ball if index > length(balls) % not enough balls score = -1; % error return; end score = score + first \* balls(index); % count score including extra from previous strike first = second; % move multiplier value from second to first second = 1; % reset multpilier for the ball after next to 1 if balls(index) + balls(index -1) == 10 % spare first = first + 1; % so next counts extra elseif balls(index) + balls(index -1) > 10 % cannot score higher than 10 in a frame score = -1; % error! return; end end for ii = [first second] % max 2 extra balls if needed if ii < 2 % no extra ball here break; % we are done end index = index + 1; % take next ball if index > length(balls) % not enough balls score = -1; % error! return; end score = score + (ii-1) \* balls(index); % extra balls: count them one less than a normal ball end if index < length(balls) % additional ball in the input score = -1; % error! end end

### Problem maxsubsum

traditional brute-force solution with four nested loops

function [x y rr cc s] = maxsubsum(A) [row col] = size(A); % initialize result to the 1-by-1 subarray at the top left corner of A x = 1; % top left corner of subarray y = 1; % top left corner of subarray rr = 1; % height of subarray cc = 1; % width of subarray s = A(1,1); % sum for r = 1:row % height of subarray for c = 1:col % width of subarray for ii = 1:row-r+1 % start position row for jj = 1:col-c+1 % start position col tmp = sum(sum(A(ii:ii+r-1,jj:jj+c-1))); % sum up candidate if tmp > s % if larger than current max s = tmp; % set the new values x = ii; y = jj; cc = c; rr = r; end end end end end end

### Problem maxsubsum (alternative solution)

Using Kadane's algorithm. Kadane's algorithm finds the contiguous subvector with the max sum within a vector using a single loop. For a detailed explanation, google "Kadane's algorithm maximum subarray problem." It is much faster than the previous solution because it needs only three nested loops. Try both with a 100x100 matrix and you'll see the difference :)

function [fx1 fy1 rr cc mx] = maxsubsum(A) [row col] = size(A); mx = A(1,1)-1; for ii = 1:row tmp = zeros(1,col); for jj = ii:row tmp = tmp + A(jj,:); [y1 y2 cur] = kadane(tmp); if cur > mx mx = cur; fx1 = ii; rr = jj-ii+1; fy1 = y1; cc = y2-y1+1; end end end end function [x1, x2, mx] = kadane(v) mx = v(1); x1 = 1; x2 = 1; cx1 = 1; cur = 0; for ii = 1:length(v) cur = cur+v(ii); if(cur > mx) mx = cur; x2 = ii; x1 = cx1; end if cur < 0 cur = 0; cx1 = ii + 1; end end end

### Problem queen check

It uses the fact that a diagonal either starts in the first column or ends in the last column (or both). Only sum and max built-in functions are used.

function ok = queen\_check(board) n = 8; ok = true; v = board(:); % create a vector in col major order w = v(end:-1:1); % reverse order, so last col becomes first col for ii = 1:n tests = [ sum(board(:,ii)) % row #ii sum(board(ii,:)) % col #ii sum(v(ii:n+1:(n-ii+1)\*n)) % diagonal starting in the first column going down sum(v(ii:n-1:ii\*n-1)) % diagonal starting in the first column going up sum(w(ii:n+1:(n-ii+1)\*n)) % diagonal starting in the last column going up sum(w(ii:n-1:ii\*n-1)) % diagonal starting in the last column going down ]; if max(tests) > 1 % these should be all 0 or 1 ok = false; % otherwise return false return; end end end

### Problem queen check (alternative solution)

Surprise, surprise: MATLAB has a built-in function called diag and flip

function safe = queen\_check (B) inC = sum(B); % sum of queens in each column inR = sum(B,2)'; % sum of queens in each row F = flip(B); % flipped board for antidiagonals for k=-6:6 inD(k+8) = sum(diag(B,k)); % sum of queens in each diagonal inE(k+8) = sum(diag(F,k)); % sum of queens in each antidiagonal end safe = max([inR inC inD inE])<=1; % queen counts at most one end

### Problem roman2

Nice and short solution

function A = roman2 (R) % This function initially assumes the supplied input is valid. If it is not valid, % the result, when converted back to Roman, will differ from the original input. Roman = 'IVXLC'; Arabic = {1 5 10 50 100}; LastValue = 0; % V is value, LastValue is last V A = uint16(0); for k = length(R):-1:1 % scan backward from last character P = strfind(Roman,R(k)); % search list of valid Roman characters if isempty(P) % if invalid V = 0; % value is zero else % else V = Arabic{P}; % value is Arabic equivalent end if V<LastValue % if subtractive situation A = A-V; % subtract this value else % else A = A+V; % add this value end % (in either case, V=0 did nothing) LastValue = V; % update last value used end if A>=400 || ~strcmp(R,A2R(A)) % if out of range or result does A = uint16(0); % not generate original string end % send back zero end % convert Arabic to Roman function R = A2R (A) % Remove subtraction by including secondary moduli. Roman = {'I' 'IV' 'V' 'IX' 'X' 'XL' 'L' 'XC' 'C'}; Arabic = {1 4 5 9 10 40 50 90 100}; R = ''; k = 9; while k>0 % remove larger moduli first if A>=Arabic{k} % if value is at least current modulus A = A-Arabic{k}; % remove modulus from value R = [R Roman{k}]; % append Roman character else % else k = k-1; % consider next smaller modulus end end end

### Problem roman2 (alternative implementation)

Uses a Finite State Machine (FSM). For a detailed description, [download this PDF document](https://d396qusza40orc.cloudfront.net/matlab/homeworks/FSM-Desc.pdf" \t "_blank).

function num = roman2(rom) % State machine-based implementation % the variable states contains the value of each state % the index into this vector is the ID of the given state states = [0 1 1 1 3 8 5 1 1 1 10 10 10 30 80 50 10 10 10 100 100 100]; % each row of trans contains one state transition % 1st col: current state; 2nd col: input char; 3rd col: next state trans = [ 1 'I' 2; 1 'X' 11; 1 'C' 20; 1 'L' 16; 1 'V' 7; 2 'I' 3; 2 'V' 5; 2 'X' 6; 3 'I' 4; 7 'I' 8; 8 'I' 9; 9 'I' 10; 11 'X' 12; 11 'V' 7; 11 'I' 2; 11 'L' 14; 11 'C' 15; 12 'X' 13; 12 'V' 7; 12 'I' 2; 13 'V' 7; 13 'I' 2; 14 'V' 7; 14 'I' 2; 15 'V' 7; 15 'I' 2; 16 'V' 7; 16 'I' 2; 16 'X' 17; 17 'V' 7; 17 'I' 2; 17 'X' 18; 18 'V' 7; 18 'I' 2; 18 'X' 19; 19 'V' 7; 19 'I' 2; 20 'V' 7; 20 'I' 2; 20 'C' 21; 20 'X' 11; 20 'L' 16; 21 'V' 7; 21 'I' 2; 21 'C' 22; 21 'X' 11; 21 'L' 16; 22 'V' 7; 22 'I' 2; 22 'X' 11; 22 'L' 16; ]; state = 1; % initial state: 1 num = 0; % initial value: 0 for ii = 1:length(rom) % take input from left state = next\_state(state, rom(ii), trans); % find next state if state == -1 % no such transition num = 0; % illegal roman number break; % get out end num = num + states(state); % otherwise, increase value end num = uint16(num); end function state = next\_state(state,ch,trans) for ii = 1:size(trans,1) % check each legal transition if trans(ii,1) == state && trans(ii,2) == ch % for current state and input char state = trans(ii,3); % return next state return; end end state = -1; % no transition found end

### Problem bell

function x = bell(n) % Check input (integer >= 1) if (n ~= floor(n)) || (n < 1) x = []; elseif (n == 1) % Special case of n = 1 x = 1; else % Make matrix of zeros x = zeros(n); % Fill in top-left corner for 2-by-2 x(1:2,1:2) = [1 2;1 0]; % Loop over remaining "lines" for k = 3:n % 1st element of the line k is the last element of line k-1 x(k,1) = x(1,k-1); % Loop over the remaining elements for j = 2:k % jth element is sum of j-1 element of current line plus % j-1 element of previous line x(k-j+1,j) = x(k-j+1,j-1) + x(k-j+2,j-1); end end end end

**Solutions to Homework 7**[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

**Problem integerize**

traditional solution with a single if-elseif-statement

function name = integerize(A) mx = max(A(:)); name = 'NONE'; if mx <= intmax('uint8') name = 'uint8'; elseif mx <= intmax('uint16') name = 'uint16'; elseif mx <= intmax('uint32') name = 'uint32'; elseif mx < intmax('uint64') name = 'uint64'; end end

**Problem integerize (alternative solution)**

using a cell vector of strings and a for-loop instead

function cl = integerize(A) cls = {'uint8'; 'uint16'; 'uint32'; 'uint64'}; cl = 'NONE'; mx = max(A(:)); for ii = 1:length(cls) if intmax(cls{ii}) >= mx cl = cls{ii}; break; end end end

**Problem integerize (alternative solution)**

using a cell vector of strings and vector indexing instead

function iclass = integerize(A) c = {'uint8','uint16','uint32','uint64','NONE'}; % Array of maximum values for each class x = 2.^[8,16,32,64] - 1; % Index into names, based on size of largest element of A iclass = c{sum(max(A(:))>x)+1}; end

**Problem May2015**

function sub\_May2015 days = ['Thu'; 'Fri'; 'Sat'; 'Sun'; 'Mon'; 'Tue'; 'Wed' ]; for ii = 1:31 m(ii).month = 'May'; m(ii).date = ii; m(ii).day = days(rem(ii,7)+1,:); % +1 is needed because 0 is an invalid index end end

**Problem June2015**

traditional solution with a for-loop

function m = June2015 days = [ 'Sun'; 'Mon'; 'Tue'; 'Wed'; 'Thu'; 'Fri'; 'Sat']; for ii = 1:30 m{ii,1} = 'June'; m{ii,2} = ii; m{ii,3} = days(rem(ii,7)+1,:); end end

**Problem June2015 (alternative solution)**

using MATLAB built-in functions instead

function x = June2015 % Make vector of dates for June 2015 t = datetime(2015,6,1:30); % Make a cell array from the components of t x = cat(1,month(t,'name'),num2cell(day(t)),day(t,'shortname'))'; end

**Problem codeit**

traditional solution, looking at one char at a time

function out = codeit(in) for ii = 1:length(in) if in(ii) <= 'z' && in(ii) >= 'a' % lower case out(ii) = 'a' + 'z' - in(ii); % encrypt it elseif in(ii) <= 'Z' && in(ii) >= 'A' % upper case out(ii) = 'A' + 'Z' - in(ii); % encrypt it else % everything else out(ii) = in(ii); % no change needed end end end

**Problem codeit (alternative solution)**

using logical indexing instead, the input and the output arguments are the same

function txt = codeit (txt) U = txt > 64 & txt < 91; % identify uppercase L = txt > 96 & txt < 123; % identify lowercase txt(U) = char(155-txt(U)); % encrypt uppercase txt(L) = char(219-txt(L)); % encrypt lowercase end

**Problem dial**

translating the actual requirements straight to code works, but it is pretty long and somewhat awkward

function ph = dial(str) code = {'ABC'; 'DEF'; 'GHI'; 'JKL'; 'MNO'; 'PRS'; 'TUV'; 'WXY'}; ph = str; % set the output to the input for ii = 1:length(str) c = str(ii); % the current char from the input if c == ' ' || c == '-' || c == '(' || c == ')' ph(ii) = ' '; % these characters need to turn into spaces continue; elseif (c >= '0' && c <= '9') || c == '#' || c == '\*' continue; % these need to remain unchanged else n = -1; for jj = 1:length(code) if ~isempty(strfind(code{jj},c)) % looking for legal letters n = jj + 1; % Found it! ABC on the dial maps to 2 not 1, hence the +1 break; end end if n == -1 % if we did not find a valid letter ph = []; % need to return [] return; end ph(ii) = char('0' + n); % otherwise, add the char for the right number end end end

**Problem dial (alternative solution)**

no loop and a single if-statement

function ph = dial(str) % the variable code has the characters' required mapping starting from space, ending with Y % x is for illegal input (e.g., see how Q maps to x in-between 7-s) code = ' xx#xxxx \*xx xx0123456789xxxxxxx2223334445556667x77888999'; ph = []; % default return value in case of illegal input n = str-' '+1; % creates a vector of indexes into code from the input characters % the first two sum()-s check for out-of-range input (smaller than space or larger than Y ) % the third sum() checks for any input char mapping to x, that is, illegal input if ~((sum(n <= 0) + sum(n > length(code))) || sum(code(n) == 'x')) ph = code(n); % a single assignment does the actual transformation of the input string end end

**Problem replace**

using a for-loop and logical indexing

function strs = replace(strs,c1,c2); for ii=1:length(strs) % for each string in the cell vector strs{ii}(strs{ii} == c1) = c2; % replace all c1-s with c2-s at once end end

**Problem roman**

problem size is small, so it is easier to simply enumerate all 20 numbers

function num = roman(rom) romans = { 'I' 'II' 'III' 'IV' 'V' 'VI' 'VII' 'VIII' 'IX' 'X' ... 'XI' 'XII' 'XIII' 'XIV' 'XV' 'XVI' 'XVII' 'XVIII' 'XIX' 'XX'}; num = uint8(0); for ii = 1:20 if strcmp(rom,romans{ii}) num = uint8(ii); break end end end

**Problem roman (alternative solution)**

using find() instead of a loop

function ar = roman(str) allstr = {'I','II','III','IV','V','VI','VII','VIII','IX','X',... 'XI','XII','XIII','XIV','XV','XVI','XVII','XVIII','XIX','XX'}; ar = find(strcmp(allstr,str)); % find() returns the indexes of non-zero elements if isempty(ar) % if no match, input is bad ar = 0; % no need to convert to uint8 yet end ar = uint8(ar); % convert to uint8 end

**Problem censor**

function out = censor(strs,str) out = {}; % creates the output from scratch for ii = 1:length(strs) % for each string in the cell vector if isempty(strfind(strs{ii},str)) % if the substring is not found out = [out strs{ii}]; % the current string goes into the output end end end

**Solutions to Homework 6**[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

**Problem neighbor**

function w = neighbor(v) w = []; if min(size(v)) == 1 % must be a vector for ii = 1:length(v)-1 % if length is less than 2, loop won't do anything w(ii) = abs(v(ii+1) - v(ii)); end end end

**Problem neighbor (alternative solution)**

no explicit loop

function w = neighbor(v) if length(v) < 2 || min(size(v)) ~= 1 % must be a vector of at least two elements w = []; else w = abs(v(1:end-1)-v(2:end)); % take the difference of two subvectors end % of length (n-1) end

**Problem replace\_me**

builds up the output one element at a time

function w = replace\_me(v,a,b,c) if nargin < 3 b = 0; end if nargin < 4 c = b; end w = []; for k = 1:length(v); if v(k) == a % if a is found, w = [w,b,c]; % we insert b and c at the end of the current w else % otherwise, w = [w,v(k)]; % we insert the original element of v end end end

**Problem replace\_me (alternative solution)**

only changes the output vector when an instance of a is found

function w = replace\_me(v,a,b,c) if nargin < 3 b = 0; end if nargin < 4 c = b; end w = v; % make w the same as v wi = 1; % wi is used to index into w for vi = 1:length(v) if v(vi) == a w = [w(1:wi-1) b c w(wi+1:end)]; % insert b and c at position wi wi = wi + 1; % increment wi end wi = wi + 1; % wi is incremented in either case end end

**Problem halfsum**

using nested loops

function s = halfsum(A) [row col] = size(A); s = 0; for ii = 1:row for jj = ii:col % the column index only starts at the current row index s = s + A(ii,jj); end end end

**Problem halfsum (alternative solution)**

using a single loop and sum

function s = halfsum(A) [nr,~] = size(A); s = 0; for r = 1:nr % for each row s = s + sum(A(r,r:end)); % sum adds up the elements right of the diagonal (inclusive) end % in the current row end

**Problem large\_elements**

function found = large\_element(A) [row col] = size(A); found = []; for ii = 1:row for jj = 1:col if A(ii,jj) > ii + jj % if the element is larger than the sum of its indexes found = [found; ii jj]; % add a new row to the output matrix end end end end

**problem one\_per\_n**

using while-loop

function n = one\_per\_n(x) n = 0; sum = 0; while sum < x && n <= 10000 n = n + 1; sum = sum + 1/n; end if n > 10000 n = -1; end end

**problem one\_per\_n (alternative solution)**

using for-loop

function n = one\_per\_n(x) s = 0; for n = 1:1e4 s = s + 1/n; if s >= x return; end end n = -1; end

**Problem approximate\_pi**

function [a,k] = approximate\_pi(delta) k = 0; f = sqrt(12); % compute sqrt(12) only once a = f; % the value of a for k == 0 while abs(pi-a) > delta % while we are further away than delta k = k + 1; % increment k a = a + f\*(-3)^(-k)/(2\*k+1); % add increment to current value of a end end

**Problem separate\_by\_two**

using division and rounding

function [even,odd] = separate\_by\_two(A) even = A(fix(A/2) == A/2)'; % if A is even, rounding does not do anything to A/2 odd = A(fix(A/2) ~= A/2)'; % if A is odd, it gets rid of the .5 part, so they won't be equal end % note that this will put non-integers into odd

**Problem separate\_by\_two (alternative solution)**

using mod (or rem)

function [even, odd] = separate\_by\_two(A) even = A(mod(A,2) == 0)'; % mod gives 0 if even odd = A(mod(A,2) == 1)'; % mod gives 1 if odd end % note that this one will not put non-integers in any of the outputs

**Problem separate\_by\_two (alternative solution)**

using mod (or rem)

function [even,odd] = separate\_by\_two(A) mod2 = logical(mod(A,2)); even = A(~mod2)'; % modulo 2 is zero for even numbers (logical false), so we need to negate it odd = A(mod2)'; % modulo 2 is non-zero for odd numbers, that is, logical true end % note that this will put non-integers into odd

**Problem divvy**

function A = divvy (A,k) L = (mod(A,k) ~= 0); % creates a logical matrix based on divisibility by k A(L) = k \* A(L); % changes only the non-divisible elements of A by multiplying them by k end % uses A as both input and output, so we only need to modify some elements of A

**Problem divvy (alternative solution)**

single line solution

function I = divvy(I,k) I(mod(I,k) ~= 0) = I(mod(I,k) ~= 0) \* k; end % same solution as above, but it repeats the modulo computation

**Problem square\_wave**

using a for-loop

function sq = square\_wave(n) t = 0 : 4\*pi/1000 : 4\*pi; % setup vector according to the specs sq = zeros(1,length(t)); % initialize output to 0 for ii = 1:2:2\*n % run for first n odd numbers (2k-1) sq = sq + cos(ii\*t-pi/2)/ii; % add the next cosine term end end

**Problem square\_wave (alternative solution)**

tricky code with no explicit loops

function s = square\_wave(n) t = 0 : 4\*pi/1000 : 4\*pi; % setup vector according to the specs idx = (2\*(1:n)' - 1); % make column vector of fist n odd numbers (2k-1) % idx\*t makes a matrix; each row is (2k-1)\*t, for a given k % idx\*ones(size(t)) also makes a matrix; each element of row k is just (2k-1) % sum down the columns s = sum(sin(idx\*t) ./ (idx\*ones(size(t))),1); end % the second argument to sum is needed in case n is 1 % remember that sum(x) sums x along columns unless x is a row vector!

**Problem my\_prime**

using a for-loop

function a = myprime(n) a = false; if n > 1 % 1 is by definition not prime for ii = 2:sqrt(n) % see explanation below if ~mod(n,ii) return; end end a = true; end end % x is prime if it is NOT divisible by all integers from 2 to sqrt(x) % because factors have to come in pairs -- one bigger than sqrt(x) and % one smaller (or both equal)

**Problem my\_prime (alternative solution)**

with no explicit loops

function prim = myprime(p) v = 2:sqrt(p); v = v(rem(p,v) == 0); % if p is prime, none of the remainders can be 0 prim = ~length(v) && (p ~= 1); % so if v has any elements, p is not prime end % 1 is handled by the (p ~= 1) condition

**Solutions to Homework 5**[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

Here are the "official" solutions. Note that there are multiple ways to solve any non-trivial problem, so these are just representative examples.

**Problem generationXYZ**

function gen = generationXYZ(year) if year < 1966 gen = 'O'; elseif year < 1981 gen = 'X'; elseif year < 2000 gen = 'Y'; elseif year < 2013 gen = 'Z'; else gen = 'K'; end end

**Problem generationXYZ (alternative solution)**

Using no if-statements

function gen = generationXYZ(yr) opts = {'O','X','Y','Z','K'}; % Create cell array of options idx = 1 + sum(yr >= [1966,1981,2000,2013]); % Calculate index by comparing year to edge values gen = opts{idx}; end

**Problem letter\_grade**

function G = letter\_grade(score) if score >= 91 G = 'A'; elseif score >= 81 G = 'B'; elseif score >= 71 G = 'C'; elseif score >= 61 G = 'D'; else G = 'F'; end end

**Problem sort3**

Using no built-in functions

function v = sort3(a, b, c) if a <= b v = [a b]; else v = [b a]; end if c >= v(2) % a and b in v are ordered. Where to insert c? v = [v c]; % at the end elseif c <= v(1) v = [c v]; % at the beginning else v = [v(1) c v(2)]; % in the middle end end

**Problem sort3**

Using no if-statements.

function v = sort3 (a,b,c) v = [a b c]; % unordered v = [min(v(1),v(3)) v(2) max(v(1),v(3))]; % the 1st and 3rd are in order v = [min(v(1),v(2)) max(v(1),v(2)) v(3)]; % move 2nd left if necessary v = [v(1) min(v(2),v(3)) max(v(2),v(3))]; % move 2nd right if necessary end

**Problem classify**

function c = classify(x) [row, col] = size(x); if row == 0 || col == 0 % any dim == 0 -> empty c = -1; elseif row == 1 && col == 1 % both dim == 1 -> scalar c = 0; elseif row == 1 || col == 1 % none of the above, but one dim == 1 -> vector c = 1; else c = 2; end end

**Problem classify (alternative solution)**

function c = classify(x) mindim = min(size(x)); maxdim = max(size(x)); if mindim == 0 % if one dim == 0, it must be empty c = -1 elseif maxdim == 1 % otherwise, both dim == 1 (since max == 1) -> scalar c = 0; elseif mindim == 1 % otherwise, if the smaller dim == 1 -> vector c = 1; else c = 2; end end

**Problem classify (alternative solution)**

Using no if-statements

function y = classify (x) d = size(x); p = prod(d); % multiplies the two dims y = -1 +(p>=1) +(p>1) +(min(d)>1) % each added condition increases the answer by one end % Note that the first two solutions are longer but easier to read and understand than this one.

**Problem older**

function a = older(y1,m1,d1,y2,m2,d2) a = 1; if y1 == y2 && m1 == m2 && d1 == d2 a = 0; elseif (y1 > y2) || (y1 == y2 && m1 > m2) || (y1 == y2 && m1 == m2 && d1 > d2) a = -1; end end

**Problem older (alternative solution)**

Using no if-statements

function a = older(y1,m1,d1,y2,m2,d2) a1 = y1 \* 366 + m1 \* 31 + d1; % does not have to be exact date in days... a2 = y2 \* 366 + m2 \* 31 + d2; % it simply makes a1 and a2 comparable a = sign(a2 - a1); % sign() returns -1, 0 or 1, just what is needed end % multiplying by 366 or greater is needed because of leap years

**Problem movies**

function cando = movies(hr1,min1,durmin1,hr2,min2,durmin2) cando = false; endtime = hr1\*60 + min1 + durmin1; % convert times to minutes starttime = hr2\*60 + min2; if endtime <= starttime && endtime + 30 >= starttime % so we can compare them cando = true; end end

**Problem movies (alternative solution)**

Using no if-statement

function cando = movies(h1,m1,d1,h2,m2,d2) end1 = h1\*60 + m1 + d1; st2 = h2\*60 + m2; cando = (end1 <= st2 && end1+30 >= st2); end

**Problem sines**

function [s1 s2 sums] = sines(pts,amp,f1,f2) if nargin < 1, pts = 1000; end if nargin < 2, amp = 1; end if nargin < 3, f1 = 100; end if nargin < 4, f2 = f1\*1.05; end t = 0 : 2\*pi/(pts-1) : 2\*pi; s1 = amp \* sin(f1\*t); s2 = amp \* sin(f2\*t); sums = s1 + s2; end % The sin() function has a full period between 0 and 2\*pi. % To set up the vector t, dividing by (pts-1) is needed % because n points in a line define (n-1) consecutive segments % and not n. For example, two points define a single line segment. % The function call sin(f1\*t) will create exactly f1 full periods % using vector t defined above.

**Problem moving average**

function a = moving\_average(x) persistent xp; if isempty(xp) xp = x; % first time, the buffer simply contains x elseif length(xp) < 25 xp(end+1) = x; % while fewer than 25 elements, keep adding x to the buffer else xp = [xp(2:end),x]; % replace first (oldest) element by shifting to the left end % and inserting x at the end a = mean(xp); end

**Problem moving average (alternative solution)**

Using no if-statement

function avg = moving\_average (in) persistent buffer; buffer = [in buffer(1:end-(length(buffer) == 25))]; avg = mean(buffer); end % This is an illustration of a short, but tricky solution. However, % a longer, but more readable solution is always preferred, therefore, % the first solution is better! % % This one works by realizing that we do not need to check whether the % buffer is empty or not, since [x buffer] will work either way. % The tricky part is how the length is handled. While the buffer is % shorter than 25, buffer(1:end) is used. Once it reaches 25, it turns % into buffer(1:end-1), exactly what is needed.

[Published with MATLAB® R2014a](http://www.mathworks.com/products/matlab/)

**Solutions to Homework 5**[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

Here are the "official" solutions. Note that there are multiple ways to solve any non-trivial problem, so these are just representative examples.

**Problem generationXYZ**

function gen = generationXYZ(year) if year < 1966 gen = 'O'; elseif year < 1981 gen = 'X'; elseif year < 2000 gen = 'Y'; elseif year < 2013 gen = 'Z'; else gen = 'K'; end end

**Problem generationXYZ (alternative solution)**

Using no if-statements

function gen = generationXYZ(yr) opts = {'O','X','Y','Z','K'}; % Create cell array of options idx = 1 + sum(yr >= [1966,1981,2000,2013]); % Calculate index by comparing year to edge values gen = opts{idx}; end

**Problem letter\_grade**

function G = letter\_grade(score) if score >= 91 G = 'A'; elseif score >= 81 G = 'B'; elseif score >= 71 G = 'C'; elseif score >= 61 G = 'D'; else G = 'F'; end end

**Problem sort3**

Using no built-in functions

function v = sort3(a, b, c) if a <= b v = [a b]; else v = [b a]; end if c >= v(2) % a and b in v are ordered. Where to insert c? v = [v c]; % at the end elseif c <= v(1) v = [c v]; % at the beginning else v = [v(1) c v(2)]; % in the middle end end

**Problem sort3**

Using no if-statements.

function v = sort3 (a,b,c) v = [a b c]; % unordered v = [min(v(1),v(3)) v(2) max(v(1),v(3))]; % the 1st and 3rd are in order v = [min(v(1),v(2)) max(v(1),v(2)) v(3)]; % move 2nd left if necessary v = [v(1) min(v(2),v(3)) max(v(2),v(3))]; % move 2nd right if necessary end

**Problem classify**

function c = classify(x) [row, col] = size(x); if row == 0 || col == 0 % any dim == 0 -> empty c = -1; elseif row == 1 && col == 1 % both dim == 1 -> scalar c = 0; elseif row == 1 || col == 1 % none of the above, but one dim == 1 -> vector c = 1; else c = 2; end end

**Problem classify (alternative solution)**

function c = classify(x) mindim = min(size(x)); maxdim = max(size(x)); if mindim == 0 % if one dim == 0, it must be empty c = -1 elseif maxdim == 1 % otherwise, both dim == 1 (since max == 1) -> scalar c = 0; elseif mindim == 1 % otherwise, if the smaller dim == 1 -> vector c = 1; else c = 2; end end

**Problem classify (alternative solution)**

Using no if-statements

function y = classify (x) d = size(x); p = prod(d); % multiplies the two dims y = -1 +(p>=1) +(p>1) +(min(d)>1) % each added condition increases the answer by one end % Note that the first two solutions are longer but easier to read and understand than this one.

**Problem older**

function a = older(y1,m1,d1,y2,m2,d2) a = 1; if y1 == y2 && m1 == m2 && d1 == d2 a = 0; elseif (y1 > y2) || (y1 == y2 && m1 > m2) || (y1 == y2 && m1 == m2 && d1 > d2) a = -1; end end

**Problem older (alternative solution)**

Using no if-statements

function a = older(y1,m1,d1,y2,m2,d2) a1 = y1 \* 366 + m1 \* 31 + d1; % does not have to be exact date in days... a2 = y2 \* 366 + m2 \* 31 + d2; % it simply makes a1 and a2 comparable a = sign(a2 - a1); % sign() returns -1, 0 or 1, just what is needed end % multiplying by 366 or greater is needed because of leap years

**Problem movies**

function cando = movies(hr1,min1,durmin1,hr2,min2,durmin2) cando = false; endtime = hr1\*60 + min1 + durmin1; % convert times to minutes starttime = hr2\*60 + min2; if endtime <= starttime && endtime + 30 >= starttime % so we can compare them cando = true; end end

**Problem movies (alternative solution)**

Using no if-statement

function cando = movies(h1,m1,d1,h2,m2,d2) end1 = h1\*60 + m1 + d1; st2 = h2\*60 + m2; cando = (end1 <= st2 && end1+30 >= st2); end

**Problem sines**

function [s1 s2 sums] = sines(pts,amp,f1,f2) if nargin < 1, pts = 1000; end if nargin < 2, amp = 1; end if nargin < 3, f1 = 100; end if nargin < 4, f2 = f1\*1.05; end t = 0 : 2\*pi/(pts-1) : 2\*pi; s1 = amp \* sin(f1\*t); s2 = amp \* sin(f2\*t); sums = s1 + s2; end % The sin() function has a full period between 0 and 2\*pi. % To set up the vector t, dividing by (pts-1) is needed % because n points in a line define (n-1) consecutive segments % and not n. For example, two points define a single line segment. % The function call sin(f1\*t) will create exactly f1 full periods % using vector t defined above.

**Problem moving average**

function a = moving\_average(x) persistent xp; if isempty(xp) xp = x; % first time, the buffer simply contains x elseif length(xp) < 25 xp(end+1) = x; % while fewer than 25 elements, keep adding x to the buffer else xp = [xp(2:end),x]; % replace first (oldest) element by shifting to the left end % and inserting x at the end a = mean(xp); end

**Problem moving average (alternative solution)**

Using no if-statement

function avg = moving\_average (in) persistent buffer; buffer = [in buffer(1:end-(length(buffer) == 25))]; avg = mean(buffer); end % This is an illustration of a short, but tricky solution. However, % a longer, but more readable solution is always preferred, therefore, % the first solution is better! % % This one works by realizing that we do not need to check whether the % buffer is empty or not, since [x buffer] will work either way. % The tricky part is how the length is handled. While the buffer is % shorter than 25, buffer(1:end) is used. Once it reaches 25, it turns % into buffer(1:end-1), exactly what is needed.

**Solutions to Homework 4**[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

**Problem quadrants:**

**function Q = quadrants(n)**

**a = ones(n);**

**Q = [a 2\*a ; 3\*a 4\*a];**

**end**

**Problem checkerboard:**

**function b = checkerboard(n,m)**

**b = ones(n,m);**

**b(1:2:n,2:2:m) = 0;**

**b(2:2:n,1:2:m) = 0;**

**end**

**Problem randomness:**

**function r = randomness(limit,n,m)**

**r = fix(limit \* rand(n,m)) + 1;**

**end**

**Problem mtable:**

**function [t s] = mtable(n,m)**

**t = (1:n)' \* (1:m);**

**s = sum(t(:));**

**end**

*If we matrix multiply a column vector of length N by a row vector of length M, each element of the resulting N-by-M matrix will be the product of one element from each vector. Therefore, we can create a multiplication table by setting the column vector to 1:N and the row vector to 1:M and using matrix multiplication.*

**Problem identity:**

**function I = identity(n)**

**I = zeros(n);**

**I(1 : n+1 : n^2) = 1;**

**end**

*Here we index into a matrix with a single index and MATLAB handles it as if it was a vector using column-major order. Putting ones at the first position and jumping n+1 every time, will put them exactly in the diagonal.*

**olutions to Homework 3**[Help Center](https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc/" \o "Click here if you're experiencing technical problems or found errors in the course materials." \t "_blank)

**Problem odd\_index:**

**function out = odd\_index(M)**

**out = M(1:2:end, 1:2:end);**

**end**

**Problem int\_col:**

**function v = int\_col(n)**

**v = [n 1:n-1]';**

**end**

*Note that this is just one possible solution. There are many others.*

**Problem rich:**

**function usd = rich(cent)**

**usd = [0.01 0.05 0.10 0.25] \* cent';**

**end**

*We use the fact that matrix multiplication sums up a set of products. Multiplying a row vector with a column vector will result in a scalar. Here it performs the exact calculations we need.*

**Problem light\_time:**

**function [mins km] = light\_time(mile)**

**km = mile \* 1.609;**

**mins =  km / 3e5 / 60;**

**end**

**Problem pitty:**

**function c = pitty(ab)**

**c = sqrt(ab(:,1) .^ 2 + ab(:,2) .^2);**

**end**

**Problem pitty (alternative solution):**

**function c = pitty(ab)**

**c = sqrt(sum(ab' .^ 2))';**

**end**

*Here we use the fact that the function sum works column by column. So, transposing and then squaring every element will put the squares of the corresponding a-s and b-s into columns. The function sum then adds them up, and sqrt computes each element's square root. Finally, we need to transpose the result back into a column vector.*

**Problem bottom\_left:**

**function M = bottom\_left(N,n)**

**M = N(end-n+1:end, 1:n);**

**end**

*We need the last n rows and the first n columns. The only trick here is that we need end-n+1, because end-n:end would get us n+1 indexes and not n as required.*

**Problem mean\_squares:**

**function mm = mean\_squares(nn)**

**mm = mean((1:nn).^2);**

**end**

**Problem hulk:**

**function H = hulk(v)**

**H = [v' (v').^2 (v').^3];**

**end**