

Assignment A Conversion of grades

One advantage of the new Danish 7-point grading scale compared to the old 00-13 grading scale is that it is easily converted to the European Credit Transfer System (ECTS). The following table provides conversion between the three different scales.

13-scale	7-point	ECTS
13	12	A
11		
10	10	B
9	7	C
8		
7	4	D
6	02	E
5	00	Fx
03		
00	-3	F

■ Problem definition

Create a function named `convertGrade` that takes a grade given in the 7-point or 13-scale as numeric input (use the numbers in brackets) and the scale (7-point or 13-scale) as a string, and returns the grade converted to the ECTS scale as a string (written exactly as in the table above.)

■ Solution template

```
function ECTSGrade = convertGrade(grade, scale)
% Insert your code
```

Input

grade Grade given on the 7-point scale or 13-scale (whole number).
scale Scale to convert from (string): either '13-scale' or '7-point'.

Output

ECTSGrade Grade on the ECTS scale (string).

■ Example

If the grade is 7 and the scale to convert from is 13-scale, the string D must be returned.

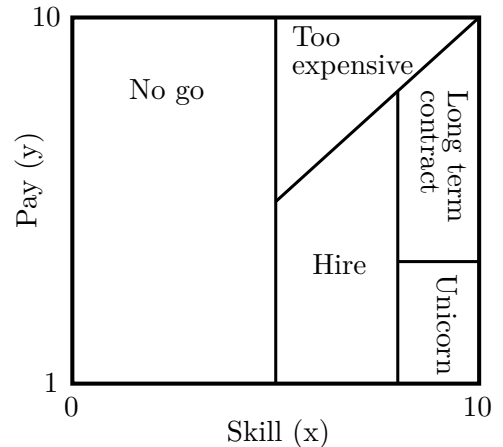
A ■

```
function ECTSGrade = convertGrade(grade, scale)
    T = [0 3 5 6 7 8 9 10 11 13];
    S = [-3 0 2 4 7 10 12];
    if(grade == -3)
        ECTSGrade = 'F';
    elseif(grade == 0 && strcmp(scale, '13-scale'))
        ECTSGrade = 'F';
    elseif(grade == 0 && strcmp(scale, '7-point'))
        ECTSGrade = 'Fx';
    elseif(grade == 2)
        ECTSGrade = 'E';
    elseif(grade == 3 || grade == 5)
        ECTSGrade = 'Fx';
    elseif(grade == 4)
        ECTSGrade = 'D';
    elseif(grade == 6)
        ECTSGrade = 'E';
    elseif(grade == 7 && strcmp(scale, '13-scale'))
        ECTSGrade = 'D';
    elseif(grade == 7 && strcmp(scale, '7-point'))
        ECTSGrade = 'C';
    elseif(grade == 8 || grade == 9)
        ECTSGrade = 'C';
    elseif(grade == 10)
        ECTSGrade = 'B';
    elseif(grade >= 11 && grade <= 13)
        ECTSGrade = 'A';
    else
        disp('Error');
    end
end
```

Assignment B When to hire an employee

An employer has made a chart of when to hire applicants to his firm. He uses two parameters to decide how to handle the applicant; how skilled the applicant is (skill) and how high a salary the applicant wants (pay). Skill is raised on a scale from 0 to 10 and Pay is raised on a scale from 1 to 10, since no applicants want to work for free. The employer do not want to hire any applicants who are less than 5 skilled, so this is the "No go" zone. He makes a Skill-Pay line and applicants above this line is considered "Too expensive". Applicants below the Skill-Pay line and between 5 and 8 skilled ends up in the "Hire" zone. Applicants above 8 skilled, below the skill-pay line and above 4 pay should be considered for a "Long term contract". Applicants which are more than 8 skilled and who want less than 4 pay are considered "Unicorn", since such applicants simply do not exist. The following table lists the applicant zones:

Applicant zone	Skill (x)	Pay (y)
No go	$x < 5$	
Too expensive	$x \geq 5$	$y > 0.9x + 1$
Hire	$5 \leq x < 8$	$y \leq 0.9x + 1$
Long term contract	$x \geq 8$	$4 < y \leq 0.9x + 1$
Unicorn	$x \geq 8$	$y \leq 4$



■ Problem definition

Create a function named `hireApplicant` that takes the skill and pay as numeric input and returns the applicantZone as a string (written exactly as in the table above).

■ Solution template

```
function applicantZone = hireApplicant(skill, pay)
% Insert your code
```

Input

skill Skill given on a scale from 0 to 10 (decimal number).
pay Pay given on a scale from 1 to 10 (decimal number).

Output

applicantZone Applicant zone (string).

■ Example

If the skill is 8 and the pay is 7, the string `Long term contract` must be returned.

```
function applicantZone = hireApplicant(skill, pay)
    if(skill < 5)
        applicantZone = 'No go';
    elseif(skill >= 5 && pay > (skill*0.9 +1))
        applicantZone = 'Too expensive';
    elseif(skill >= 5 && skill < 8 && pay <= (skill*0.9 +1))
        applicantZone = 'Hire';
    elseif(skill >= 8 && pay > 4 && pay <= (skill*0.9 +1))
        applicantZone = 'Long term contract';
    elseif(skill >= 8 && pay <= 4)
        applicantZone = 'Unicorn';
    end
end
```

Assignment C Average hearing loss

Hearing loss is usually evaluated using an audiogram where the hearing loss is measured for different pure tones as the deviation from 0 dB. You are working with a data set where hearing loss has been measured for 7 different frequencies and for a number of individuals. The data for each individual is saved as the rows in a matrix M . The matrix therefore contains 7 columns, one for each measured frequency. You want to calculate the average hearing loss across the individuals for each frequency, however without including individuals with a severe hearing loss. A hearing loss of more than 70 dB is considered severe.

■ Problem definition

Create a function named `averagedB` which takes a matrix as input, removes rows that contain values above 70 dB, and returns a vector containing the average of each column for the reduced matrix.

■ Solution template

```
function averageM = averagedB(M)
% Insert your code
```

Input

M Input matrix

Output

`averageM` Output vector

■ Example

Consider the following input matrix:

$$M = \begin{bmatrix} 25 & 25 & 30 & 40 & 40 & 40 & 45 \\ 50 & 55 & 55 & 65 & 65 & 70 & 75 \\ 75 & 70 & 70 & 70 & 50 & 50 & 55 \\ 25 & 30 & 35 & 40 & 50 & 55 & 60 \end{bmatrix}$$

Since row 2 and 3 contain a value above 70 dB the rows must be removed.

$$M = \begin{bmatrix} 25 & 25 & 30 & 40 & 40 & 40 & 45 \\ \del{50} & \del{55} & \del{55} & \del{65} & \del{65} & \del{70} & \del{75} \\ \del{75} & \del{70} & \del{70} & \del{70} & \del{50} & \del{50} & \del{55} \\ 25 & 30 & 35 & 40 & 50 & 55 & 60 \end{bmatrix}$$

The average of each column are calculated for the reduced matrix to produce the vector below, which should be the output of the function:

$$averageM = \begin{bmatrix} 25.0 & 27.5 & 32.5 & 40.0 & 45.0 & 47.5 & 52.5 \end{bmatrix}$$

```
function averageM = averagedB(M)
    %eliminate rows with elements > 70
    M = M(~any(M > 70,2),:);
    %mean for each coloumn
    [s,~] = size(M);
    if (s == 1)
        averageM = M;
    else
        averageM = mean(M);
    end
end
```

Assignment D Building LEGO bricks

You want to build the tallest tower of lego bricks such that the volume of the tower does not exceed 1000cm^3 . It is assumed that the volume of a lego brick can be calculated by:

$$V = h \cdot l \cdot w$$

where h , w and l are the height, width, and length of the lego brick, respectively.

■ Problem definition

Create a function named `buildLego` that takes as input the height h , width w , and length l of a lego brick and computes the maximum number of lego bricks that can be used for your tower if the volume can not exceed 1000cm^3 .

■ Solution template

```
function bricks = buildLego(h, l, w)
% Insert your code
```

Input

<code>h</code>	Height of the lego brick (positive decimal number)
<code>l</code>	Length of the lego brick (positive decimal number)
<code>w</code>	Width of the lego brick (positive decimal number)

Output

<code>bricks</code>	Number of lego bricks (integer)
---------------------	---------------------------------

■ Example

Consider $h = 1$, $l = 8$, and $b = 1.6$. For different values of *bricks* the volume can be computed as shown below:

<i>bricks</i>	75	76	77	78	79	80
<i>volume</i>	960.00	972.80	985.60	998.40	1011.2	1024.0

The largest number of lego bricks for which the volumen does not exceed 1000cm^3 is $bricks = 78$. Thus, the function should return the value 78.

D

```
function bricks = buildLego(h, l, w)
    Vb = h*l*w;
    V = 0;
    bricks = 0;
    while (V <= 1000)
        V = V + Vb;
        bricks = bricks + 1;
    end
    bricks = bricks - 1;
end
```


Assignment E Sudoku check

Sudoku is a number puzzle, where the objective is to fill a 9-by-9 grid with numbers between 1 and 9 so that each row, each column, and each 3-by-3 block contains all the numbers from 1 through 9. Each row, each column, and each 3-by-3 block must therefore sum to 45.

■ Problem definition

Create a function named `sudokuCheck` that takes as input a 9-by-9 matrix, `sudokuBoard`, and checks whether the sum of each row, each column, and each 3-by-3 block is 45. The function must output the variable `check` that contains the overall number of rows, columns, and blocks that do not sum to 45.

■ Solution template

```
function check = sudokuCheck(sudokuBoard)
% Insert your code
```

Input

`sudokuBoard` 9-by-9 matrix (numbers from 1 to 9)

Output

`check` Result from check (integer)

■ Example

Consider the sudoku board below. The sum is calculated for each row, column, and 3-by-3 block (brown and red numbers).

5	3	4	6	7	8	9	1	2	45
6	7	2	1	9	5	3	5	8	46
1	9	8	3	4	2	4	6	7	44
8	5	9	7	6	1	4	2	3	45
4	2	6	8	5	3	7	9	1	45
7	1	3	9	2	4	8	5	6	45
9	6	1	5	3	7	2	8	4	45
2	8	7	4	1	9	6	3	5	45
3	4	5	2	8	6	1	7	9	45
45	45	45	45	45	45	44	46	45	

The sum of row 2 and 3 and column 7 and 8 is different from 45, and the program must therefore return the integer 4.

```
function check = sudokuCheck(sudokuBoard)
    M = sudokuBoard;
    tot = 0;
    %row
    R = sum(M);
    %coloumns
    C = sum(M,2);
    %blocks
    B(1) = sum(sum(M(1:3,1:3)));
    B(2) = sum(sum(M(1:3,4:6)));
    B(3) = sum(sum(M(1:3,7:9)));
    B(4) = sum(sum(M(4:6,1:3)));
    B(5) = sum(sum(M(4:6,4:6)));
    B(6) = sum(sum(M(4:6,7:9)));
    B(7) = sum(sum(M(7:9,1:3)));
    B(8) = sum(sum(M(7:9,4:6)));
    B(9) = sum(sum(M(7:9,7:9)));

    check = sum(R ~= 45) + sum(C ~= 45) + sum(B ~= 45);
end
```