Problem 1.
(a) Write MATLAB code that generates a 4×4 array named `XXA` whose elements are all one except for its diagonals elements that are as follows. $XXA(1,1) = 40$, $XXA(2,2) = 100$, $XXA(3,3) = 100$, and $XXA(4,4) = 1000$.
(b) Write MATLAB code that generates a 5×5 array named `XXB` whose elements are all 2 except for 5 in the middle.

Problem 2.
What does the following piece of code do with respect to a, b? What are the values of a, b at the end of execution?

```
a=100; b=200; t=0;
t= a; a=b; b=t;
```

Problem 3.
What are the values of x, y, z if each one of the three MATLAB Programs is executed?

```
Program1
x=10;
y=20;
z=10\20;

x=2*x;
z=2+x;
y=y+z;
z=1-y;
```

```
Program2
x=10;
y=x;
z=2*x;

z=z/x;
x=y-x;
z=y;
x=z;
```

```
Program3
x=10;
y=20;
z=-1;

z=x;
x=y;
y=z;
```

Problem 4.
How can one print 15 in hexadecimal in MATLAB?

Problem 5.
What happens with `int8(127), int8(128), int8(-128), int8(-129)`.

Problem 6.
Write MATLAB code that prints the elements of an $m \times n$ matrix a in column major form but (i) as a row vector, (ii) as a column vector.
Repeat the same with row-major form.

Problem 7.
Rewrite each of the next four statements using the colon operator.

```
v1 = [10 9 8 7 6 5 4 3 2 1]
v2 = v1([10 7 4 1 2 4 6 8])
v1([2 3 4]) = [10 5 0]
v3 = [1 2 3 1 2 3 1 2 3]
```

Then replace the next 3 statements with one.

```
v1(6) = 10
v1(7) = 20
v1(8) = 30
```

Turn page for solutions

Problem 1.

(a)

```
XXA = ones(4,4); XXA(1,1)=40; XXA(2,2)=100;XXA(3,3)=100;XXA(4,4)=1000;
```

(b)

```
XXB = ones(5,5); XXB=XXB +XXB; XXB(3,3)=5;
```

Problem 2.

It swaps a, b. Note the sequence (t, a, b) of assignments (i.e. the left side of an equal sign $=$) in the three statements of line 2. Note also that the right side of the equal sign $=$ is (a, b, t) i.e. the original sequence (t, a, b) circularly shifted right by one position. Thus a is 200 and b is 100 at the end.

Problem 3.

(a)

```
x=20; y=42, z=-41;
```

(b)

```
x=10; y=10, z=10;
```

(c)

```
x=20; y=10, z=10;
```

Problem 4.

If you do naively, `format hex` and then type in `15` you will get something like the third line below, which is not very clear how close to the expected answer is. On paper the answer for the hexadecimal representation of 15 is `F`, or `0xF` if we do not specify number of bits (or number of hexadecimal digits). If an 8-bit unsigned int representation is requested then `0x0F` is the answer.

Note 1. MATLAB skips `0x` in printing hexadecimal numbers.

```
>> format hex
>> 15                              % 15 is stored as a double
ans = 402e000000000000             % this is the hex representation of double 15.0 not int 15!
>> single(15)                      % this is the hex representation of float  15.0
 ans = 41700000
>> double(15)                      % same as the first
ans = 402e000000000000
>> int8(15)                        % explicitly store 15 as 8-bit integer
 ans = 0f
>> int16(15)                       % or 16-bit integer
 ans = 000f
>> uint8(15)                       % explicitly store 15 as 8-bit usigned integer
 ans = 0f
```

The reason for thas is that 15 is stored in `ans` as a double. Thus `402e000000000000` is the hexadecimal representation in double of `15.0`. We did not discuss the representation of real numbers in double (double precision) or float (single precision) in class. If you are wondering what `15` looks like in `float` type in `single(15)` in MATLAB. Note that `double(15)` is equivalent to just typing `15.` i.e. it provides the default behavior.

Now to get the representation of integer 15 in hexadecimal you need to EXPLICITLY request that 15 be stored as an integer rather than the default `double` floating-point representation. One way to do so is to use one of `int8, int16, int32, int64, uint8, uint16, uint32, uint64` functions to get signed 8-bit, 16-bit, 32-bit, 64-bit and unsigned 8-bit, 16-bit, 32-bit, 64-bit representations of an integer (in decimal).

So `int8(15)` and `int16(15)` give us the familiar `0f` and `000f` results. Note that a hexadecimal digit corresponds to 4 binary digits thus in the former case the 2 hexadecimal digits are for the 8-bit and in the latter case the 4 hex digits are for the 16-bit representation of 15.

There is one more way to get the hex representation using `fprintf`. So `%X %x` print the hexadecimal representation of an integer argument, but `%bx %tx` view the integer arguments in double and single floating-point precision (i.e. as a double or single).

```
>> fprintf('%X %x %bx %tx\n',15,15,15,15);
F f 402e000000000000 41700000
>> fprintf('%X %x %bx %tx\n',int8(15),int8(15),int8(15),int8(15));
F f 402e000000000000 41700000
>>
```

Problem 5.

As explained in the previous problem `int8` uses 8-bit to store an integer that is signed. For an unsigned integer use `uint8` instead. This means that only 256 values can be represented, the negative 128 values `-128 , ... , -1`, the 127 positive values `1 , ... 127` and the zero. Any positive integer greater than 127 will be represented as 127 in 8-bit signed representation in MATLAB. Similarly any negative integer less than -128 will be represented as -128.

```
>> format short
>> int8(128)
 ans = 127
>> int8(127)
 ans = 127
>> int8(-129)
 ans = -128
```

Problem 6.

```
% Let matrix a be m x n
>> a(:)      % prints column major order/form/fill-in of a as a vector (aka column vector)

>> a(1:end)  % prints column major as a row vector

>> a=a';     % if a can have complex numbers, use a = a.' instead
>> a(:)      % prints column major as a column

>> a(1:end)  % prints column major as a row vector

>> a=a';     % or a=a.'
```

Problem 7.

```
v1 =10:-1:1;
v2 = v1([10:-3 1 2:2:8]);
v1(2:4) = 10:-5:0;
v3 =[ 1:3 1:3 1:3];

v1(6:8) = 10:10:30;
```