BIOE 198MI Biomedical Data Analysis. Spring Semester 2019.
Lab8: Introduction to data types and conversion

Modern programming languages include techniques for creating and converting between different data types. There are basic definitions of data types used in all programming languages and some specific to certain languages. This lab is an overview of a few of the data types used in MATLAB.

The earliest programming employed "machine languages" made of streams of 0's and 1's. Today we enter and process data in data types that include numeric arrays (integer, floating point, and logical arrays), character arrays (character and string arrays), cell arrays, and others, but each has a binary form. That is, each data type is converted into **binary digits (bits)** consisting of zeros and ones that digital computers need to process data. In this lab, we consider four variable types: **Integers, floating-point numbers, characters and logical or Boolean variable.** For example, we can convert between integers and their binary forms as illustrated in the following examples,

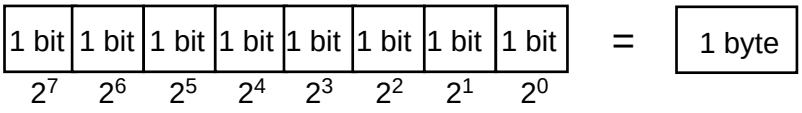
0 → 0000 0000	5 → 0000 0101	99 → 0110 0011	199 → 1100 0111
1 → 0000 0001	20 → 0001 0100	100 → 0110 0100	200 → 1100 1000
2 → 0000 0010	30 → 0001 1110	120 → 0111 1000	255 → 1111 1111
3 → 0000 0011	40 → 0010 1000	150 → 1001 0110	256 → 0000 0001 0000 0000
4 → 0000 0100	50 → 0011 0010	180 → 1011 0100	257 → 0000 0001 0000 0001

In MATLAB you can use `dec2bin` or `bin2dec` to convert back and forth between a decimal integer and its binary equivalent:

`dec2bin(120) = 01111000` and `bin2dec('1111000') = 120`

Note how you need to enter the argument for `bin2dec` as a string! Also notice, in the table of 20 examples above, the first 18 require only one byte. The last two examples require two bytes. **One byte is a string of 8 bits. It is unit of storage capable of representing $2^8 = 256$ distinct values.**

Unsigned Integers



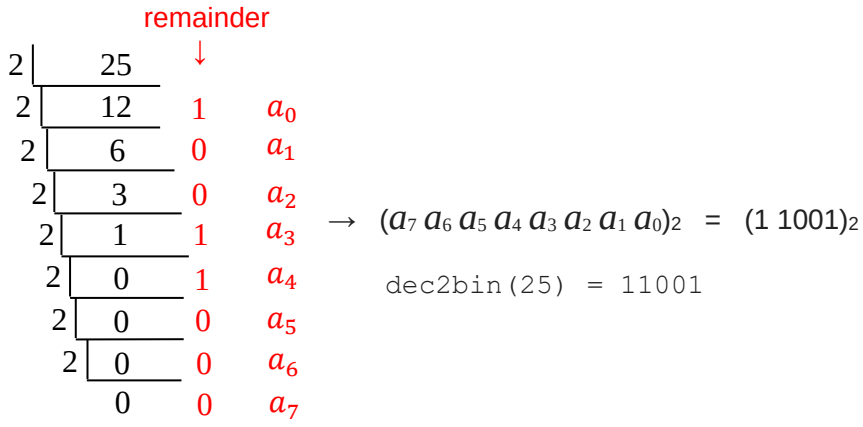
most significant bit → least significant bit

To convert from binary to decimal, use the following method

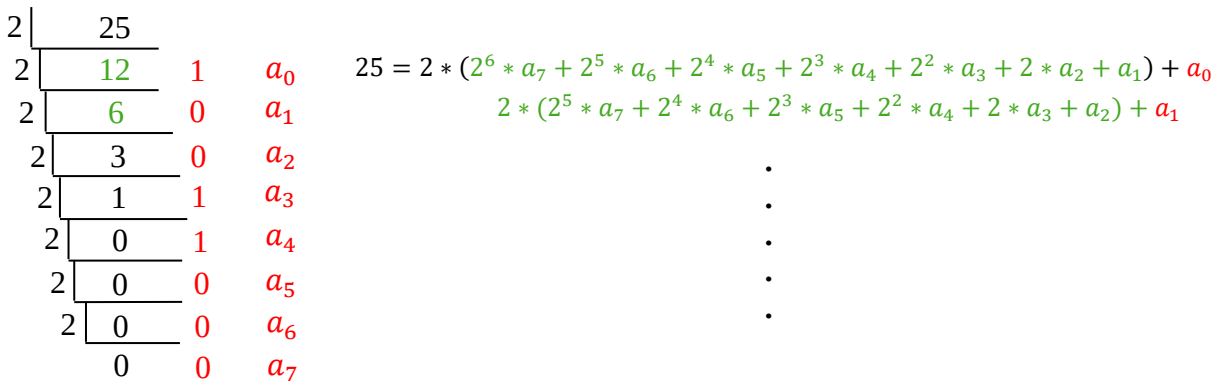
$(0101\ 0101)_2 \rightarrow 0x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 64+16+4+1 = (85)_{10}$

Exercise: what are the decimal equivalents of binary numbers $(1011\ 0110)_2$, $(0110\ 1011)_2$?

To convert from decimal to binary number, we do the following:
To convert $(25)_{10} \rightarrow (?)_2$ use the following algorithm,



Here is how this algorithm works...



Exercise: transform $(35)_{10}$ to a binary number.

We found that a byte can represent nonnegative integers from 0 – 255, which is a total of 256 values. We can convert an array of values to an unsigned 8-bit integer using: `uint8`.

For example, for

```
a = [3 -3 3.6 4/2 7/3 300];
b=uint8(a)
```

Exercise: Can you interpret the results of executing the code above? Explain each element of `b`.

`uint16`, `uint32` and `uint64` are extensions of `uint8` to include, respectively, 2 bytes, 4 bytes, and 8 bytes. What are the ranges of each?

for uint8 = 0 – $(2^8$-1) or 0 – 255. for uint16 = 0 – $(2^{16}$-1) or 0 – 65535.

for uint32 = 0 – $(2^{32}$-1) or 0 – 4.2950e+09. for uint64 = 0 – $(2^{64}$-1) or 0 – 1.8447e+19.

Signed Integers

MATLAB uses `int8`, `int16`, `int32` and `int64` to store signed integers. The left-most bit of data now becomes the *sign bit*, where 1 denotes a negative number and 0 denotes a positive number.

Because the left-most bit indicates the sign of the integer, the range of integers represented is -128 to 127, which remains 256 unique values. For example,

1	1	0	1	0	0	0	1
1 - N 0 - P	2^6	2^5	2^4	2^3	2^2	2^1	2^0

= - 81

$(0000\ 0000)_2 = (0)_{10}$ $(0000\ 0001)_2 = (1)_{10}$ $(1000\ 0001)_2 = (-1)_{10}$ $(1111\ 1111)_2 = (-127)_{10}$

$(0111\ 1111)_2 = (127)_{10}$ $(1000\ 0000)_2 = (-128)_{10}$ = **definition**. Use `bin2dec` on the rightmost 7 bits.

Exercise: Explain each element of `b` and `c` for the following:

```
a = [3 -3 3.6 4/2 7/3 300];
b=int8(a)
c=int16(a)
```

Floating-point representation of real numbers (whole and decimal values)

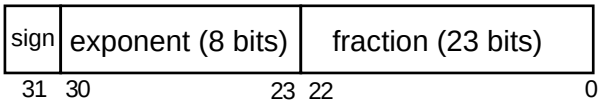
MATLAB represents floating-point numbers in either double-precision or single-precision format.

The default data type for MATLAB is `double` precision floating point. Each data point requires 64-bits (8 bytes) for storeage. A `single` precision floating-point number requires only 32-bits (4 bytes). You can convert an array of data to single precision using `b=single(a)` or double precision using `c=double(a)`.

Exercise: A 4K (UltraHD) television screen has 3840 x 2160 pixels. While neither dimension is 4000, this is still quite a lot of information. (a) Calculate the amount of memory required to store one frame if the pixel values are `uint8` and `double`. (b) If the monitor displays images at 30 frames per second what is the minimum data delivery rate needed to stream live TV?

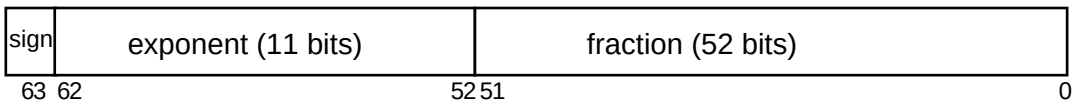
If a number lies approximately between -3.4 x 10^{38} to 3.4 x 10^{38}, we can use either `single` or `double` to store it. If the number exceeds those limits, we must use `double`.

`single`



Value it represents is $(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1 + \sum_{i=1}^{23} b_{23-i}2^{-i})$

`double`



Value it represents is $(-1)^{b_{63}} \times 2^{(b_{62}b_{61}\ldots b_{52})_2 - 1023} \times (1 + \sum_{i=1}^{52} b_{52-i}2^{-i})$

ASCII (characters)

The American Standard Code for Information Interchange (ASCII) is a character-encoding standard for electronic communication. Each keyboard character is assigned a numerical value between 0 and 255 (decimal) and so each character typed occupies a byte of memory.

Originally, the ASCII code contained 128 elements (7-bits), as based on the English alphabet. The code was extended to 8-bits to include special characters. Below is the table for the first 128 elements. In the table below, we find a character has a decimal representation. Recall that a decimal integer is also represented by a binary number.

Decimal	Char	Decimal	Char	Decimal	Char	Decimal	Char
0	[NULL]	32	[SPACE]	64	@	96	`
1	[START OF HEADING]	33	!	65	A	97	a
2	[START OF TEXT]	34	"	66	B	98	b
3	[END OF TEXT]	35	#	67	C	99	c
4	[END OF TRANSMISSION]	36	$	68	D	100	d
5	[ENQUIRY]	37	%	69	E	101	e
6	[ACKNOWLEDGE]	38	&	70	F	102	f
7	[BELL]	39	'	71	G	103	g
8	[BACKSPACE]	40	(	72	H	104	h
9	[HORIZONTAL TAB]	41	)	73	I	105	i
10	[LINE FEED]	42	*	74	J	106	j
11	[VERTICAL TAB]	43	+	75	K	107	k
12	[FORM FEED]	44	,	76	L	108	l
13	[CARRIAGE RETURN]	45	-	77	M	109	m
14	[SHIFT OUT]	46	.	78	N	110	n
15	[SHIFT IN]	47	/	79	O	111	o
16	[DATA LINK ESCAPE]	48	0	80	P	112	p
17	[DEVICE CONTROL 1]	49	1	81	Q	113	q
18	[DEVICE CONTROL 2]	50	2	82	R	114	r
19	[DEVICE CONTROL 3]	51	3	83	S	115	s
20	[DEVICE CONTROL 4]	52	4	84	T	116	t
21	[NEGATIVE ACKNOWLEDGE]	53	5	85	U	117	u
22	[SYNCHRONOUS IDLE]	54	6	86	V	118	v
23	[ENG OF TRANS. BLOCK]	55	7	87	W	119	w
24	[CANCEL]	56	8	88	X	120	x
25	[END OF MEDIUM]	57	9	89	Y	121	y
26	[SUBSTITUTE]	58	:	90	Z	122	z
27	[ESCAPE]	59	;	91	[	123	{
28	[FILE SEPARATOR]	60	<	92	\	124	|
29	[GROUP SEPARATOR]	61	=	93	]	125	}
30	[RECORD SEPARATOR]	62	>	94	^	126	~
31	[UNIT SEPARATOR]	63	?	95	_	127	[DEL]

Exercise: Use the ASCII values above or the MATLAB function `char` to decode the message stored as `uint8` values: [72 101 108 108 111 32 87 111 114 108 100 33]

Boolean (logical) operators:

Logical operations like AND and OR generate only 1 or 0 outputs indicating true or false conditions. There are three logical operators in MATLAB: the unary NOT (~), the binary AND (&&), and the inclusive OR (||) operators. Let's examine

X	Y	X && Y	X || Y	~X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Note that for `&&` and `||` the variables X and Y must be scalars. For `X&&Y`, MATLAB examines X first and if it is 0 it returns a 0 (false) output without looking at Y. This can be fast.

Exercise: Predict the outcomes of the following.

```
a=true;     %this sets scalar variable a to logical true which is logical 1
b=false;    %this sets scalar variable b to logical false which is logical 0

a&&b        % 0
(~a)&&b     % 0
(~a)||b     % 0
a||(~b)     % 1
a==false    % 0
a=='true'   % 0 0 0 0 get it?
```

However, if `a` and `b` are vectors or matrices of the same size, use `&` and `|` in place of `&&` and `||`.

Exercise: Predict the outputs of `a&b` and `a|b` and `(~a)&b` where,

```
a=[1 2 3 'c';0 2.333 10^3 55]; b=[0 4 0 'd';',' '6' 2 0];
```

Relational operators compare two variables and return a logical 0 or 1, indicating true or false.

Symbols	Descriptions
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

x	y	x<y	x<=y	x>y	x>=y	x==y	x~=y
4	5	1	1	0	0	0	1

Logical operators are often found in iterative comparative statements.

Exercise: Predict the output values for c and d.

```
clear all;
a = [1 2];
b = [1 2];
c = [0 0];
d = [0 0];

for i = 1:2
    if a(i)>=2 && b(i)<=2
       c(i) = a(i)*b(i);
    end

    if a(i)>=2 || b(i)<=2
       d(i) = a(i)*b(i);
    end
end
```

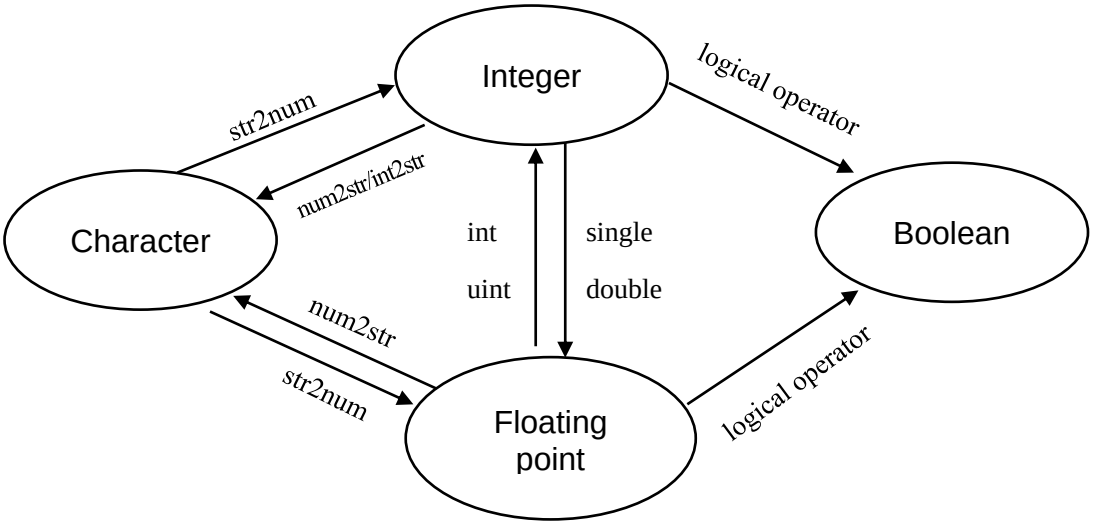
Logical indexing is a technique for selecting elements from an array that satisfy specific criteria. The elements are selected by forming an index array whose elements are either 0 or 1.

```
clear all;
x = [1 2 3 4 5 6 7 8 9 10];
y = [3 1 5 6 8 2 9 4 7  0];

a = x>3;              % a = 0   0   0   1   1   1   1   1   1   1
b = x<8;              % b = 1   1   1   1   1   1   1   0   0   0
c = x(x>3|x<8);       % c = 1   2   3   4   5   6   7   8   9   10
d = x(x>3&x<8);       % d = 4   5   6   7
e = y(x<=3);          % e = 3   1   5
f = x(y~=3);          % f = 2   3   4   5   6   7   8   9   10
```

Convert from one data type to another:

MATLAB contains a selection of functions for converting one data type into another.



Exercise: Find outputs for the following statements and interpret the results.

```
clear all
a = num2str(int8(128))              127
b = int8(str2num('-129'))          -128
whos
Name      Size              Bytes  Class
Attributes

  a         1x3                 6  char
  b         1x1                 1  int8

c = int2str(3.499999999)            3
d = int2str(3.5)                    4

e = double('1') + double('1')      98
f = double(1)   + double(1)         2

A = 123.456;
g = num2str(A)                      123.456
h = num2str(A,'%.2e')               1.23e+02
k = num2str(A,'%.2f')               123.46

x = 1; y = 1; z = 0;
o = x & y & z;                      0
```

Summary:

Categories of data types	Data types
int8 int16 int32 int64 uint8 uint16 uint32 uint64	Integer
single double	Floating point
char	Character/string
logical	Boolean

Sparse Matrix:

Thus far, we have always used dense matrices. That is, we store every elements of the matrix. However, when matrices become large, we can run out of memory to store it. Fortunately, it is not always necessary to store every element. A sparse representation can be used when many elements are zero.

```
%% sparse matrix
dense_matrix=[3 1 0 0 0;
              0 0 0 0 0;
              2 4 1 0 0;
              1 1 0 0 0;
              0 0 0 0 0;];
sparse_matrix = sparse(dense_matrix)
dense_matrix1 = full(sparse_matrix)
whos dense_matrix       % 200 bytes
whos sparse_matrix      % 160 bytes
```

```
%% create a sparse matrix
i=[1 1 3 3 3 4 4];
j=[1 2 1 2 3 1 2];
v=[3 1 2 4 1 1 1];
sparse_matrix2=sparse(i,j,v,5,5);
V=full(sparse_matrix2)
```

The description of the MATLAB function `sparse` explains the structure as follows:

```
S = sparse(i,j,s,m,n,nzmax) uses vectors i, j, and s to generate an
    m-by-n sparse matrix such that S(i(k),j(k)) = s(k), with space
    allocated for nzmax nonzeros.
```

Image Data:

2-D images are arrays of picture elements (pixels). Image pixels can be displayed in binary, grayscale, or color.

All pixels in a **binary image** are either 0 (black) or 1 (white). Binary images may also be labeled black-and-white monochrome or bitmap images since, in principle, each pixel only requires **1 bit of memory**. In practice, a logical data pixel occupies **one byte** of memory.

Pixels in a **grayscale image** describe signal intensity information, e.g., the amount of light captured by a detector as a function of position. Typically, each pixel displays one of 256 shades of gray. In that case, pixels can be `uint8` requiring **1 byte of memory** each.

Example of **grayscale** and **binary** images generated in MATLAB.

```
close all; clear all
x=1:100;y=zeros(100,100);   %first row in 100x100 image
for j=1:100                 %generate the other 99 rows
    y(j,:)=j*x;
end
q=log(y);z=q>7;             %take log and convert double to logical
yy=uint8(q);                %yy converts double to uint8
subplot(2,2,1);imagesc(y);colormap(gray);axis square
title('Linear Double')
subplot(2,2,2);imagesc(q);axis square
title('Log Double')
subplot(2,2,3);imagesc(yy);axis square
title('Log uint8 (10 levels)')
subplot(2,2,4);imagesc(z);axis square
title('Log bitmap (2 levels)')
figure;subplot(2,1,1);plot(q(1,:))
title('Row 1'); ylabel('ln(y)')
subplot(2,1,2);plot(q(100,:))
title('Row 100');ylabel('ln(y)')
```

Notice: `min(min(y)) = 1 and log(1) = 0. max(max(y))=10000 and log(10000)=9.21.` So `uint8` conversion generates 10 gray levels. Also, we can assign a `colormap` with color, but that is a false-color image.

Each pixels in a basic **color image** is composed of three primary colors, red, green, and blue (RGB). A pixel is a 1x3 array of `uint8` values requiring **3 bytes of memory per pixel**. For example, a bright pure red pixel has its red channel at maximum and its green and blue channels set at zero, (255,0,0). Similarly, a pure green pixel is stored as (0,255,0) and a pure blue channel as (0,0,255).

To convert **RGB to grayscale**, we can display equal part of each color via

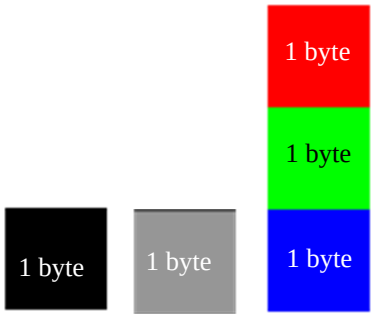
$$Gray = (R + G + B)/3$$

or weight colors differently,

$$Gray = 0.2989 * R + 0.5870 * G + 0.1140 * B$$

Assignment:

This assignment is designed to explore the appearance of color and grayscale images presented with various bit depths. The number of bits used to present each pixel determines the **contrast resolution** of the image. An RGB color image where each color channel is given as `uint8` has 24-bits of information. That is, 2^{24} = 16,777,216 different colors are possible. This image has a greater bit depth than if the each channel had 4-bits, giving 2^{12}=4,096 different colors. To reduce from 4 bits to 3 bits first convert each channel to `double`, divide the 4-bit image data by 2, round the result, convert back to `uint8`, and multiply by 2.



On the left, we represent a **binary image** as a 1-byte logical data type, having two values of 0 or 1. In the middle, we represent a **grayscale image** as a 1-byte `uint8` type with 256 gray levels. On the right, we represent a **color image** and an array of three `uint8` values occupying three bytes of storage.

1. Download **Lab8image.png** from the website. Read and display the image using

```
I=imread('Lab8image.png');
imshow(I)
```

2. Type `whos` and interpret the data you have. Create a grayscale image from the original data using the formula above.

3. Separately histogram the 8-bit R, G, and B channels in one 3x1 subplot for the original 8-bit channel data.

4. Convert the original color image data into a 4-bit image that you display at full scale.

5. Separately histogram the R, G, and B channels in one 3x1 subplot for the 4-bit channel data.

6. Convert the original color image data into a 1-bit image to be displayed at full scale. Do not use the logical operator for this!

7. Separately histogram the R, G, and B channels in one 3x1 subplot for the 1-bit channel data.

8. Presents the results as one 2x2 subplot displayed the four images (labeled). Also present three 3x1 subplots with the three sets of histograms. Four plots in total should be included in the Results.

Rubric:

1. Introduction (1 point).

2. Methods (2 points).

3. Four result figures clearly labeled and explained (4 points)

4. Discussion describing how a reduction in color-image bit depth influences the appearance of the image (3 points)