```
>> Lecture 4
>>
>>              -- Functions
>>
```

Motivation

- A large and complicated problem would be conquered by solving its subproblems.
- So the first step is problem decomposition, that is, separating tasks into smaller self-contained units.
- This is also beneficial to code reuse without copying the codes.
- Note that bugs propagate across the program when you copy and paste the codes.

Function

- A function is a piece of program code that accepts input arguments from the caller, and then returns output arguments to the caller.
- In MATLAB, the syntax of functions is similar to math functions,

$$y = f(x),$$

where x is the input and y is the output.

User-Defined Functions

- We can define a new function as follows:

```
function [outputVar] = function_name(inputVar)
    % What to do.
end
```

- This function should be saved in a file with the function name!
- Note that the input/output variables can be optional.

Example: Addition of Two Numbers

```
function z = myAdd(x, y)
    % Input: x, y (any two numbers).
    % Output: z (sum of x and y).
    z = x + y;
end
```

- It seems bloody trivial.
- The truth is that the plus operator is actually the function **plus**.[1]
- Also true for all the operators like $+$.

[1] See https://www.mathworks.com/help/matlab/ref/plus.html.

Variable-length Input Argument List[2] (Optional)

- We can know the number of input arguments for the function executed by **nargin**.
- **varargin** is an input variable in a function definition statement that enables the function to accept any number of input arguments.
  - It must be declared as the last input argument and collects all the inputs from that point onwards.
- The variable **varargout** is a special word similar to **varargin** but for outputs.

[2]See `https://www.mathworks.com/help/matlab/ref/varargin.html`.

Example

```
function ret = myAdd(varargin)

    switch nargin
        case 0
            disp("No input.");
        case 1
            ret = varargin{1};
        case {2, 3}
            ret = sum([varargin{:}]);
        otherwise
            error("Too many inputs.");
    end

end
```

Variable Scope

- Variables in a function are known as local variables, existing only for the function.
- These variables are wiped out when the function finishes its task.
- You may trace the data flow in the program by using the debugger.[3]
    - Let's set some breakpoints!!!

[3] See https://www.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html.

Example

```
clear; clc;

x = 0;
for i = 1 : 5
    addOne(x);
    disp(x); % output ?
end
```

```
function addOne(x)
    x = x + 1;
end
```

Function Handles & Anonymous Functions

- Anonymous functions are used once and not written in the standard form of functions, for example,

```
f = @(x) x .^ 2 + 1 % f is a function handle.
```

- However, they contain only single statement.
- Besides, we use function handles[4] to handle functions.
- This is also called lambda expressions.
- You can also assign an existing function to a handle, for example,

```
g = @sin
```

[4]You may refer to https://en.wikipedia.org/wiki/Function_pointer. The truth is that every function name is an alias of the function address!

More Examples[5,6,7]

```
function y = parabolicFunGen(a, b, c)
    y = @(x) a * x .^ 2 + b * x + c;
end
```

```
function y = getSlope(f, x0)
    eps = 1e-9;
    y = (f(x0 + eps) - f(x0)) / eps;
end
```

```
function y = differentiate(f)
    eps = 1e-9;
    y = @(x) (f(x + eps) - f(x)) / eps;
end
```

[5] Thanks to a lively class discussion (MATLAB244) on August 22, 2014.
[6] Contribution by Ms. Queenie Chang (MAT25108) on March 18, 2015.
[7] Thanks to a lively class discussion (MATLAB260) on September 16, 2015.

Vectorization (Revisited)

- We can apply a function to each element of array by **arrayfun**.[8]

```
B = arrayfun(@(x) 2 * x, A) % Equivalent to 2 * A.
```

- **cellfun** is similar to **arrayfun** but applied to cells.[9]

```
>> data = {"NTU", "CSIE", [], "MATLAB"};
>> isempty(data) % Output 0.
>> cellfun(@isempty, data) % Output 0 0 1 0.
```

[8] See https://www.mathworks.com/help/matlab/ref/arrayfun.html.

[9] See https://www.mathworks.com/help/matlab/ref/cellfun.html.

Error and Error Handling

- You can issue/throw an **error** if you do not allow the callee for some situations.

```
if bad_condition
    error("So wrong."); % Interrupt the normal flow.
end
```

- As an app programmer, you should use a try-catch statement to handle errors.

```
try
    % Normal operations.
catch
    % Handler operations.
end
```

Example: Combinations

- For all nonnegative integers $n \geq k$, $\binom{n}{k}$ is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

- Note that **factorial**(n) returns $n!$.

```
clear; clc;

n = input("n = ? ");
k = input("k = ? ");
y = factorial(n) / (factorial(k) * factorial(n - k))
disp('End of program.');
```

- Try $n = 2, k = 5$.
- However, **factorial**(-3) is not allowed!
- The program is not designed to handle this error, so it is interrupted in Line 5 and does not reach the end of program.
- Add error handling to the program:

```
1  clear; clc;
2
3  n = input("n = ? ");
4  k = input("k = ? ");
5  try
6      y = factorial(n) / (factorial(k) * ...
           factorial(n - k))
7  catch e % capture the thrown exception
8      disp("Error: " + e.message); % show the message
9  end
10 disp("End of program.");
```

```
>> Lecture 5
>>
>>          -- Special Topic: Text Processing
>>
```

(Most) Common Codec: ASCII[11]

- Everything in the computer is encoded in binary.
- ASCII is a character-encoding scheme originally based on the English alphabet that encodes 128 specified characters into the 7-bit binary integers (see the next page).
- Unicode[10] became a standard for the modern systems from 2007.
  - Unicode is backward compatible with ASCII because ASCII is a subset of Unicode.

[10] See Unicode 8.0 Character Code Charts.

[11] Codec: coder-decoder; ASCII: American Standard Code for Information Interchange, also see `http://zh.wikipedia.org/wiki/ASCII`.

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Characters and Strings (Revisited)

- Before R2017a, a text is a sequence of characters, just like numeric arrays.
  - For example, 'ntu'.
- Most built-in functions can be applied to string arrays.

```
clear; clc;

s1 = 'ntu'; s2 = 'csie';
s = {s1, s2};
upper(s) % output: {'NTU', 'CSIE'}
```

- Since R2017a, you can create a string by enclosing a piece of text in double quotes.[12]
  - For example, "ntu".
- You can find a big difference between characters and strings in this example:

```
clear; clc;

s1 = 'ntu'; s2 = 'NTU';
s1 + s2 % output: 188 200 202

s3 = string(s1); s4 = string(s2);
s3 + s4 % output: "ntuNTU"
```

[12] See https://www.mathworks.com/help/matlab/ref/string.html.

Selected Text Operations[13]

sprintf	Format data into string.
strcat	Concatenate strings horizontally.
contains	Determine if pattern is in string.
count	Count occurrences of pattern in string.
endsWith	Determine if string ends with pattern.
startsWith	Determine if string starts with pattern.
strfind	Find one string within another.
replace	Find and replace substrings in string array.
split	Split strings in string array.
strjoin	Join text in array.
lower	Convert string to lowercase.
upper	Convert string to uppercase.
reverse	Reverse order of characters in string.

[13] See `https://www.mathworks.com/help/matlab/characters-and-strings.html`.