



Introduction to Mathematical Software and Programming

Session 4 (w/c 10 March 2025)



Test 1 Reminder

Scheduled during **Lab Session 5** (next week)

20 minutes

10% of overall module marks

Format & Topic

- Write you solutions/codes on the test paper
- You can use MATLAB on UNNC computer during the test
- **No access to other files/notes/devices/Internet**
- Topics from Session 1 to Session 4

Test-1 Sample Problems are available on Moodle

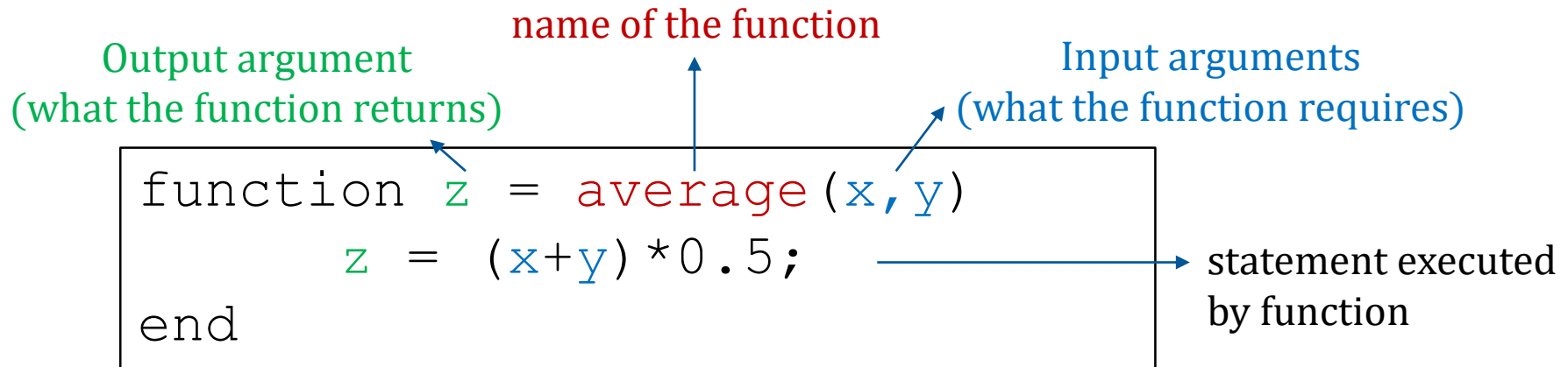
Function File (User-defined function)

- Function file is another type of MATLAB file (M-file) with following structure.

```
function[output1,output2,...] = FunctionName (input1,input2,...)  
  
Statements % main body of function  
  
end
```

- A function file is saved as **FunctionName.m** in the current folder.
- MATLAB allows **user-defined functions** that can be called(used) in the same way as built-in functions.
- Unlike script file that relies on input() or fprintf() for interactions with users, in function files, **input/output arguments** can be directly declared in the function.

Example



- Function can be called in MATLAB **command window** or **within a script file** by using its name with specified input values

Command Window:

```
>> a = average(5, 6)
```

Assign the value 5.5000 to
variable a in Workspace

```
a = 5.5000
```

What happens in the function's private workspace:

$x=5, y=6$ Receive values from function's input arguments

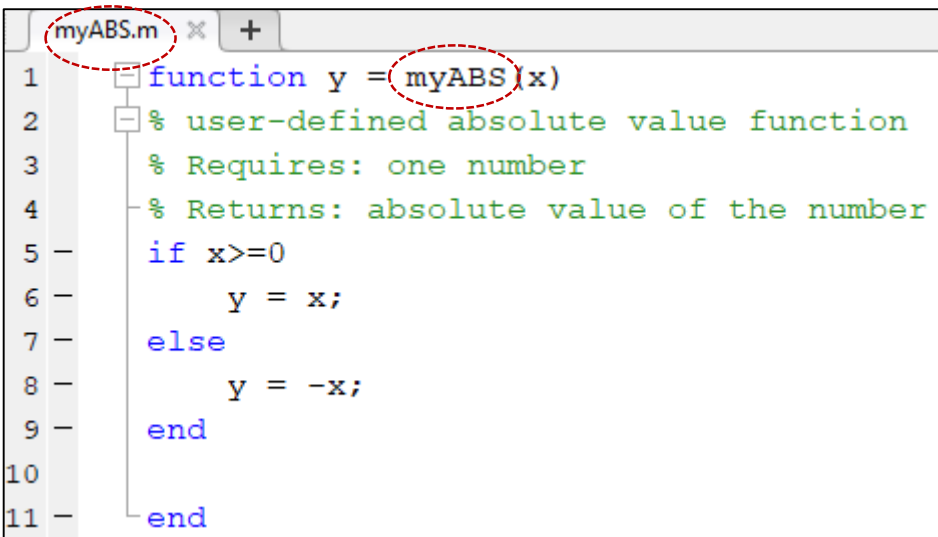
$z = (5+6)*0.5 = 5.5000$ Evaluate the statement

Return the computing result 5.5000

Note: each function file has its (private) workspace, so values of x, y, z here won't be saved in the (common) Workspace.

Rules of creating functions

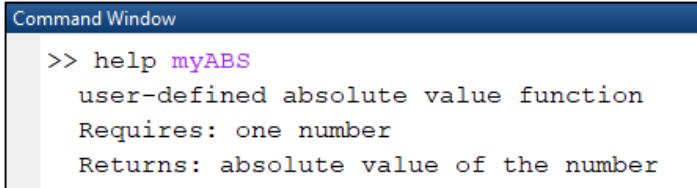
MATLAB functions must be defined in separate files and **function name** must match with the **file name**.



```
1 function y = myABS(x)
2 % user-defined absolute value function
3 % Requires: one number
4 % Returns: absolute value of the number
5 if x >= 0
6     y = x;
7 else
8     y = -x;
9 end
10
11 end
```

It is a good habit to include function's description as comments at the start (similar to algorithm header in Sem-1)

Users who are using your function file can check such information using the help command.



```
Command Window
>> help myABS
user-defined absolute value function
Requires: one number
Returns: absolute value of the number
```

While creating function/script files or variables, you should NOT use names of built-in commands/functions. In addition, no spaces should be added between words in the names (you may use an underscore _ for connecting words).

Examples: **myABS()** (user-defined function) **abs()** (MATLAB built-in function/command)

mySum, or **my_sum** (user-defined variables) **sum()** (MATLAB built-in function/command)

Making function calls

If there are n multiple **output arguments** in the function, we should use a **vector variable** of **length n** for receiving all values computed by the function.

```
max_min.m  x  +
1  function [x,y] = max_min(a,b)
2  % Find max/min values of two
3  % Requires: two numbers
4  % Returns: [max,min]
5  if a>=b
6      x = a;
7      y = b;
8  else
9      x = b;
10     y = a;
11 end
12 end
```

```
Command Window
>> [p,q] = max_min(3,5)

p =

     5

q =

     3
```

Another example: the built-in function `size()` returns both the row and column numbers of an input array. Therefore, we can make a function call of it like:

```
[m,n] = size(A)
```



Algorithm to Function

Algorithm: rootNum(a,b,c)

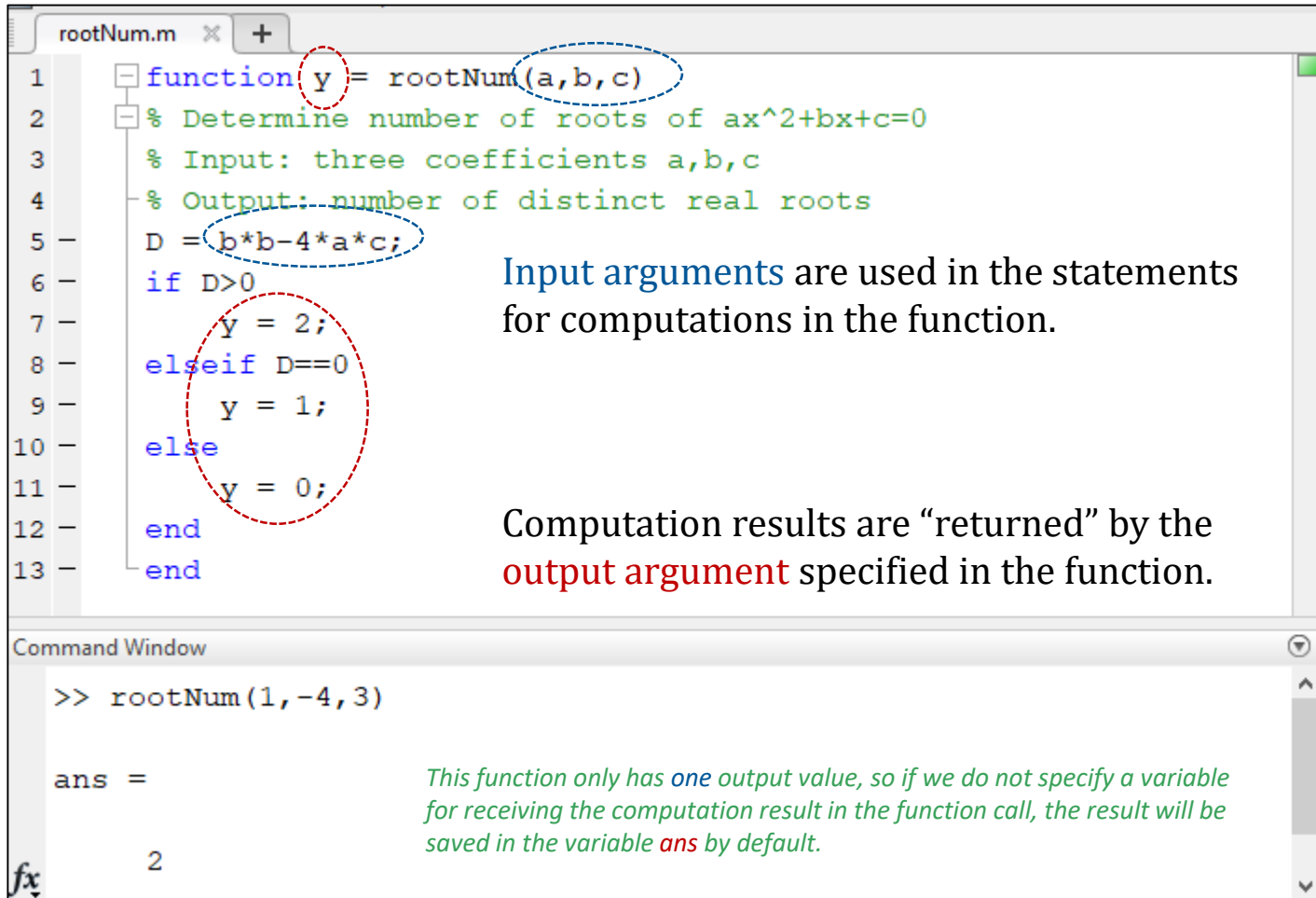
Requires: Three real numbers a, b, c ; a is not equal to 0.

Returns: Numbers of real roots of quadratic equation $ax^2 + bx + c = 0$

```
1. let D = b*b-4*a*c // assign the discriminant value to variable D
2. if D > 0
3.     return 2 // two distinct real roots
4. elseif D == 0
5.     return 1 // one distinct real roots (two identical real roots)
6. else
7.     return 0 // no real roots
8. endif
```

(Sem-1, Seminar 1)

Algorithm to Function



```
rootNum.m
1 function y = rootNum(a,b,c)
2 % Determine number of roots of ax^2+bx+c=0
3 % Input: three coefficients a,b,c
4 % Output: number of distinct real roots
5 D = b*b-4*a*c;
6 if D>0
7     y = 2;
8 elseif D==0
9     y = 1;
10 else
11     y = 0;
12 end
13 end
```

Input arguments are used in the statements for computations in the function.

Computation results are “returned” by the **output argument** specified in the function.

```
Command Window
>> rootNum(1,-4,3)

ans =

     2
```

*This function only has one output value, so if we do not specify a variable for receiving the computation result in the function call, the result will be saved in the variable **ans** by default.*

Script to Function

```
fibonacci.m  x  +
1  % First n Fibonacci number script file
2  n = input('Enter a positive integer:\n');
3  F = ones(1,n);
4  for i=3:n
5      F(i) = F(i-1)+F(i-2);
6  end
7  fprintf('First %d Fibonacci numbers are:\n',n)
8  disp(F)
```

Script file can be modified into function file by declaring **input/output arguments** in the first line of a function file.

Depending on what kind of values we want to compute using the function, we can specify different output variables in the function.

```
fibonacci.m  x  +
1  function F = fibonacci(n)
2  % First n fibonacci number function file
3  % Input: a positive integer
4  % Output: a vector of first n Fibonacci number
5  F = ones(1,n);
6  for i=3:n
7      F(i) = F(i-1)+F(i-2);
8  end
9  end
```

This example function has **one input argument (a number)** and **one output (a vector)**

Recursive Function

Algorithm: $F(n)$

Requires: a positive integer n

Returns: the n -th number in the Fibonacci sequence

```
1. if  $n==1 \parallel n==2$ 
2.     return 1 // base case
3. else
4.     return  $F(n-1)+F(n-2)$  // recursive step
5. endif
```

Two base cases (combined as one)

Two recursive calls onto function F itself at each recursive step

(Sem-1, Lecture 4)

```
fib.m  x  +
1  function y = fib(n)
2  % Recursive function for the n-th Fibonacci number
3  % Input: a positive integer
4  % Output: the n-th Fibonacci number
5  if n==1 || n==2
6      y = 1;
7  else
8      y = fib(n-1)+fib(n-2);
9  end
```

Note:

A recursive function calls itself in its body structure (same as recursive algorithm in Sem-1)

Function example

Algorithm: **factorial**(n)
Requires: a non-negative integer n
Returns: n! the factorial of n

```
1. if n==0
2.   return 1
3. else
4.   return n*factorial(n-1)
5. endif
```

(Sem-1, Lecture 3)

base case

recursive formula
recursive step
recursive call

using recursion:

```
myFactorial.m
1 function y = myFactorial(n)
2   % Compute n! using recursion
3   if n==0
4     y = 1;
5   else
6     y = n*myFactorial(n-1);
7   end
8 end
```

using iteration(loops):

```
Factorial.m
1 function y = Factorial(n)
2   % Compute n! using iteration
3   y = 1;
4   for i = 1:n
5     y = y*i;
6   end
7 end
```

Line 3: We need to design suitable initial value for y, because it is used in the statement (Line 5 on the right) for computation.

Performance comparison

```

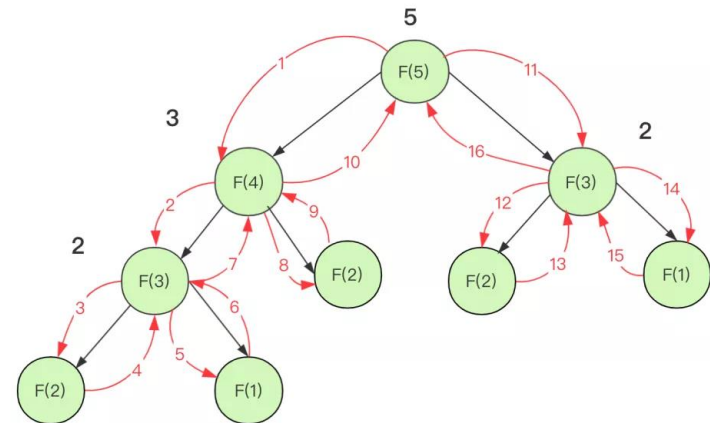
test.m  x  +
1      % script for comparing running time of
2      % iteration and recursion
3      % using the Fibonacci example
4      clear;clc
5      n = 20;
6      tic % start counting running time
7      fibo(n) % iteration
8      toc % end counting time and display running time
9      tic
10     fib(n) % recursion
11     toc
  
```

We can write script files for testing functions.

tic and **toc** will display the elapsed time for executing the codes within them.

Note:

In general, when the problem size is large, programs using *iteration* (moving forward) are *faster* than programs using *recursion* (tracing back).





Script file vs. Function file

Script file	Function file
For organizing a sequence of statements; can be executed by “Run” button or by calling the file name	For achieving a particular task; must be called by its name with designated input/output
All variables are global variables, saved in the Common Workspace after execution	Local variables are used independently for each function
Usually used for the execution of multiple statements (e.g. testing codes)	Can be called by other script/function files for the specific computation it is designed for
Recursive structure cannot be used	Recursive functions can be designed

Note:

- *Do not use the built-in command `input()` in functions, use the input arguments instead.*
- *We may use the built-in command `fprintf()` or `disp()` to display error messages in function file when needed.*



Iteration vs. Recursion

Iteration	Recursion
is when a loop repeatedly executes the set of instructions like For Loop and While Loop	is when a function calls itself within its code, thus repeatedly executing the instructions present inside it
terminates when loop condition fails	Recursion terminates when the base case is met
uses less memory; faster	uses more memory; slower
increases the size of the code	reduces the size of the code
can be applied to script and function files	can only be applied to functions

Note:

You need to carefully design the condition and variable updates in While Loop, and the base case in recursion, to avoid infinite loops or recursions.

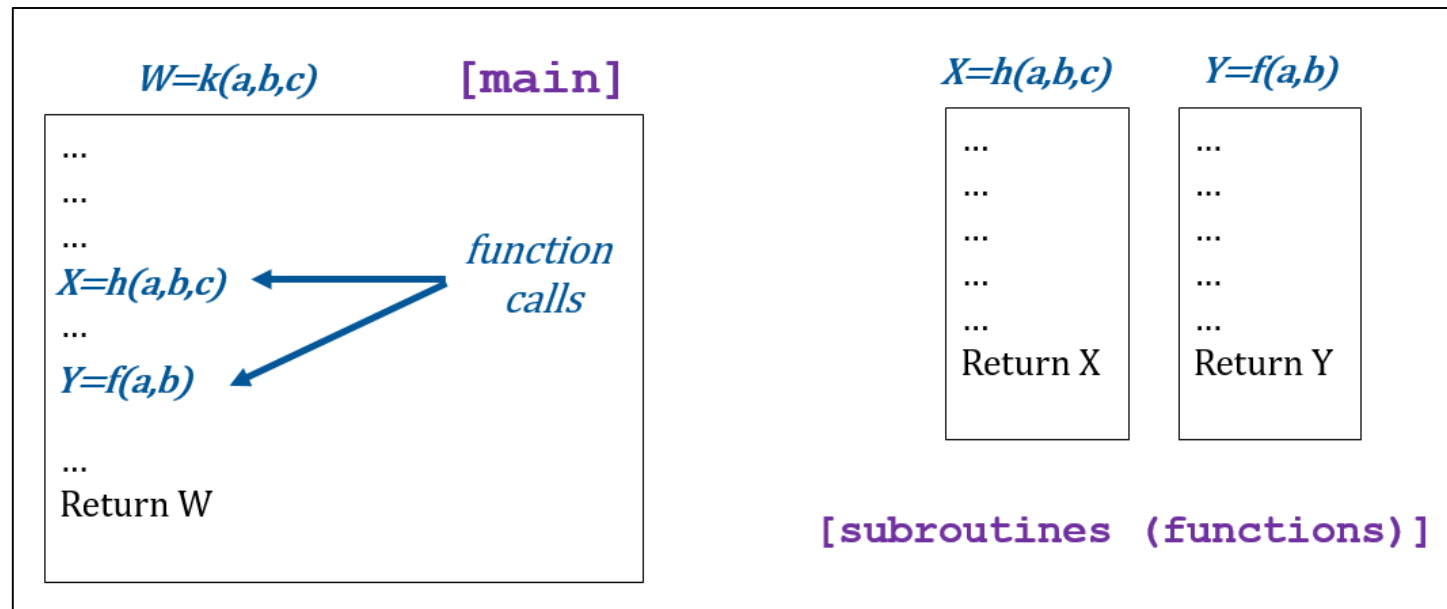
You can manually terminate the program by pressing "Ctrl"+"C" on the keyboard.

Calling sub-function

Each function file will do its own computation task independently.

We can call user-defined function in a script file, as well as in a function file.

In a function/script file for solving a complex problem, we may need to call multiple functions (as sub-functions).



(Sem-1, Lecture 3)

Program (Luhn Algorithm)

Sub-algorithm

Main algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

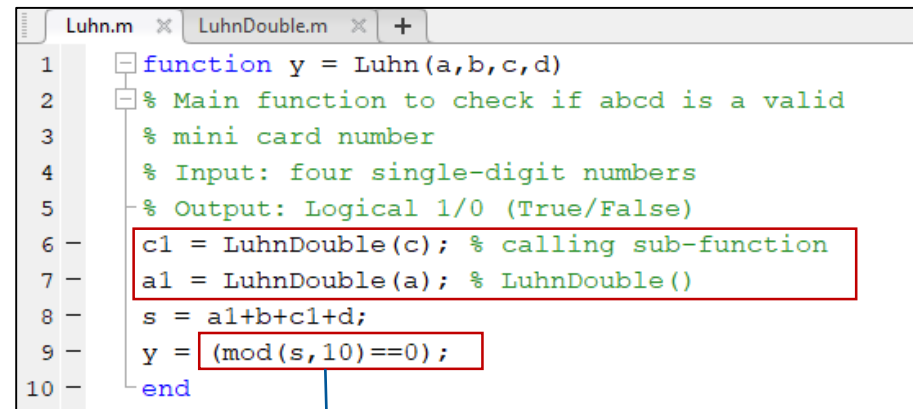
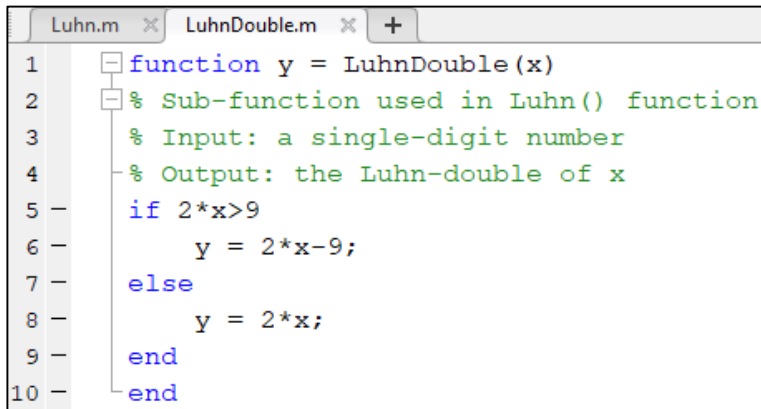
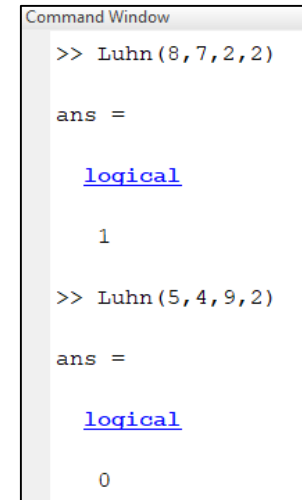
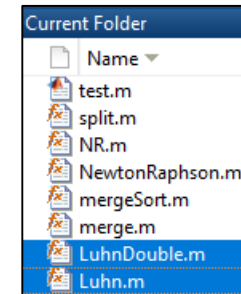
```
1. if 2*x>9
2.   return 2*x-9
3. else
4.   return 2*x
5. endif
```

calling
sub-algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c' = LuhnDouble(c)
2. let a' = LuhnDouble(a)
3. let s = a' + b + c' + d
4. return (s mod 10) == 0
```

(Sem-1, Lecture 3)



The value of this logical statement is returned by y as the computation result of this function. You can replace Line 9 by a IF structure as well.



Self-study

- Check Session 3 Solution Set available on Moodle
- Complete lab worksheet 4
- Complete homework exercise sheet 4

For environmental science and/or transferred students: review the following knowledge points from Sem-1 CELEN086 (Session 1 to Session 4) or from Internet resources.

- *Recursion and recursive algorithms*
- *Luhn Algorithm*
- *Greatest Common Divisor*
- *Euclidean Algorithm*