MATLAB PROGRAMMING: EXPRESSIONS, FUNCTIONS, CONDITIONALS AND LOOPS

COMPUTER SESSION A1

BACKGROUND

An algorithm is a step-by-step description of how to compute something. Computer programming is the implementation of algorithms on computers.

At the lowest level, a computer can perform only a few fixed, basic operations, such as adding two numbers and storing the result in a memory location, or jumping to another part of the program if the value of a memory location is 0. Such simple operations are enough to perform any computational task, but the programs become large and difficult to understand. To simplify programming, *computer languages* have been created, which can interpret higher lever constructs such as expressions and functions into the basic operations of the computer.

For this to be possible, a computer language must be very formal, every statement must be unambiguous. This means computer languages have a strict syntax where `a = 5` might mean something different than `5 = a`.

Computer languages are organized into elements such as expressions, functions and loops which can be combined to form a computer program. We will use the Matlab programming language in these courses, but the elements we will learn are common to almost all programming languages, there are only small differences in syntax.

The elements we will cover in this session are:

(1) Expressions
An expression is a collection of values, operators and variables that evaluates to a value. A single value or variable is an expression:

```
8

ans =
```

```
     8

a * (5 - 2)
```

If a has not been assigned any value, then the expression is invalid. If a has been assigned the value 5, then the expression evaluates to 15.

```
a = 5;
a * (5 - 2)

ans =

    15
```

(2) Functions

A function is a subprogram which takes values as input and returns values as output. A function behaves just like its mathematical counterpart. Functions are also very useful for structuring a big program into smaller self-contained units.

In Matlab we need to define functions in separate files, where the filename is the name of the function. To define the function $f(x) = x + 2$ we would create a file called `f.m` and write into it:

```
function y = f(x)
  % Represents the simple function f(x) = x + 2
  y = x + 2;
```

We can then evaluate the function or use it in expressions:

```
f(3)

ans =

     5

8 + f(3)

ans =

    13
```

(3) Conditionals

A *logical expression* is an expression that evaluates to a logical value - “true” or “false”. In Matlab, logical values are represented by numbers, where 0 represents “false” and all other values represent “true”.

We can test if 1 is greater than 2:

```
(1 > 2)

ans =

     0
```

The contents of a conditional statement (or selection statement) is executed only if a given logical expression is true, otherwise it is skipped over, and the next statement is executed:

```
if(1 > 2)
  5 * 7
end
```

The expression `5 * 7` inside will not be evaluated, because the logical expression $(1 > 2)$ is false. If we change it to be true:

```
if(2 > 1)
  5 * 7
end

ans =

    35
```

then the expression inside is evaluated.

(4) Loops

A loop repeats executing its content several times, either a fixed number of times by going through a list of values or until some condition has been met by evaluating a logical expression.

The *for loop* goes through a list of values:

```
for i = [1 2 3]
  i
end

i =
```

```
     1


i =

     2


i =

     3
```

Matlab has a colon operator which makes it easy to generate a list of values in sequence:

```
1:3

ans =

     1     2     3

for i = 1:3
  i
end

i =

     1


i =

     2


i =

     3
```

The *while loop* loops while the condition is true:

```
i = 1;
while(i < 5)
  i
```

```
    i = i + 1;
end

i =

     1


i =

     2


i =

     3


i =

     4
```

Before today's computer session, make sure that you understand and can answer the following questions.

Question 1

What is the difference between:

```
a = 5;
```

and

```
5 = a;
```

in Matlab?

Question 2

Does it make a difference if there's a semicolon at the end of a line?

Question 3

How do you create and edit a Matlab file (m-file)?

PREPARATIONS

Keep the relevant literature (books, lecture notes) in front of you. Have a pen and paper in front of you, it is often best to sketch a program or function on paper and to go through what will happen before starting to implement it.

PROBLEMS

Problem 1.

A simple function. We create a (very trival) matlab function, that just multiplies a given x by 2, $y = 2*x$. We give this function the name `twice(x)`. We define the function in a Matlab function m-file with the name `twice.m`. We thus create (with `>> edit` in Matlab) a file with the following content

```
function y = twice(x)
  % This function computes and returns 2x for any given x.
  y = 2 * x;
```

and give the file (with Save as ..) the name `twice.m`.

Comments:

(1) The file must start with the key word function, here followed by `y = twice(x)`, where x is the (independent) input variable, y is the (dependent) output variable, and twice is the name of your function.
(2) On the following line(s) you should write some text that explains what the function is doing. This text is displayed if you type help twice at your matlab prompt. You then get useful info about the

function without actually using it. Try this! When you use the function in a matlab computation, all text following % is ignored.

(3) You may put in extra lines and blanks to get better readability if you like.

Now test by

```
help twice
```

The info text `This function ..` should then be displayed.

Then

```
twice(7)
```

which should result in `ans = 14`, or

```
y = twice(8)
```

which should result in `y = 16`, or

```
x = 8; y = twice(x)
```

which should give the same result.

Functions of more than one variable. Write a matlab function that computes the area of a triangle with base b and height h, of the form

```
function a = triangle_area(base, height)
% comments
a = .. (for you to complete!)
```

saved under the name `triangle_area.m`. Test with

```
a = triangle_area(3,7)
```

Write a matlab function that computes the resulting capital c of a given initial investment c0, a given annual interest rate r, and after a given number of years n.

Functions with more than one output argument. Define a function that computes both the difference and the ratio of $a = 1 + 2 + 3 + .. + N$ and $b = N^2/2$, of the form

```
function [difference, ratio] = Leibschnizel(N)
% Computes the difference and ratio of a = 1 + 2 + 3 + .. + N
% and b = N^2 / 2
a = sum(1:N);
b = N^2 / 2;
difference = ..
```

Note that the output variables appear in a (comma separated) list/vector. Note also that we surpress output from within the function by ending the assignments by ;. The default output is then just the resulting values of the output variables, here difference and ratio.

Note also that if at the matlab prompt you type just `Leibschnizel(N)`, you only get the first output variable of Leibschnizel, that is the difference variable, and it will be put in the answer "box". To get both output variables you must specify two "boxes" in the matlab workspace for the Leibschnizel function to deliver the result, like `[d, r]=Leibschnizel(N)`. Now Leibschnizel will put its first output variable in d and its second in r. Test!

Vector input/output. The input/output arguments of a function can also be lists/vectors. For example, write a matlab function that computes the individual costs for newspapers, movies and cds, respectively, and also the corresponding total cost. Hint: use two input list/vector variables, one for the number of articles, and one for the corresponding current price per item, and two output variables, one list/vector with the resulting individual price for each category, and one scalar (list with just one element) for the total cost, of the following form, with input arguments `n = [n1 n2 n3]` and `p = [p1 p2 p3]`, where n1 is the number of newspapers you plan to buy, and p1 the price per newspaper, etc.

```
function [individual_costs, total_cost] = budget(n, p)
% .. Write a descriptive comment of what the function does
  individual_costs = .. (see Matlab intro I for the .* multiplication)
  total_cost = ..
```

Set up a budget using your new tool!

Composite functions. In addition to your `twice(x)` matlab function, write another one: `square_and_add_1_to(x)` which squares a given input argument and then adds one. Then consider the composite functions `twice(square_and_add_1_to(3))` and `square_and_add_1_to(twice(3))`. Do they yield the same result? Should they?

Problem 2.

Simple sum. Write a function `simplesum(n)` which sums $1+2+...+n$ using a for loop. Verify manually that it produces correct results by testing a few simple cases.

Factorial. The factorial ("fakultet" in Swedish) function (denoted by an exclamation mark after an expression) is defined for integers like so:

$$n! = \begin{cases} 1 & n = 0 \\ 1 * 2 * ... * n & n > 0 \end{cases}$$

Write a function `fact(n)` which implements the factorial function using a for or while loop and conditionals to test for the different cases (n can be either 0 or bigger than 0).

Note that a conditional statement can also include an `else`-clause:

```
result = 0;

if(n == 0)
  disp('n is 0');
  result = 1;
else
  disp('n is not 0');
```

```
  result = 2;
end
```

Before you implement your factorial function, think about whether you want to use `if-else` or perhaps several `if` testing for the different cases. Sketch your function on paper and go through the function for different cases of n.

Matlab has already implemented the factorial function as `factorial(n)`. Test that you get the same results for the different cases. Write and test one case first (the loop perhaps) to make it simpler.

Approximating e. The natual number e can be approximated using the following series:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + ...$$

Write a function `e_approx(n)` which sums n terms of the above series. Use your `fact(n)` function to compute the factorials.

e can be computed in Matlab by evaluating `exp(1)`, which computes e^1.

After you have finished, imagine how your implementation of `e_approx(n)` would look if you were not allowed to use the `fact(n)` function. Would it be easier or harder to understand the code?

SOLUTIONS

Make sure that you really try to solve each problem before looking at the solutions. Have you really tried to solve the problem or should you try again before looking at the solution?

The solutions are available on the web page of this session under *Solutions to problems*.

ABOUT

This Computer Session is part of the Body and Soul educational program. More information can be found at http://www.phi.chalmers.se/bodysoul/

This Computer Session is maintained by Johan Jansson (johanjan@math.chalmers.se).