Control flow

Control flow

A *control flow* is the order in which individual statements, instructions or function calls of the code are executed

To control which statements should be executed, and in which order, *control structures* are predefined in keywords

The main control structures are the structures of

- Conditions (if then else)
- Loops (count-controlled, condition-controlled, ...)

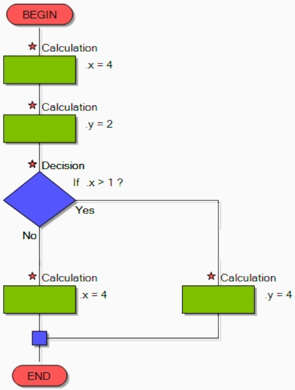

Figure 1: Chart of a control flow

Condition statements

Single condition

The instructions given within a if-statement *block* are only considered if the statement is fulfilled

```
if (condition statement which outputs a boolean)
    (matLab commands)
end
```

Otherwise, the instruction block is simply disregarded

Example:

```
>> x = 2
>> if x == 5  % The block will be executed only if x=5
>>   y=1;     % otherwise the instructions are ignored
>> end
```

Condition from multiple sub-statements

Multiple conditions can be expressed in a statement giving a single boolean output, formed by several sub-statements connected through the operators `|`, `&` and `~`

```
>> x=-3; y=5;
>> if ~(x == 5 & y >= 2) | x<-2  % Executed only if
      y=1;                       % not (x=5 and y>=2)
   end                           % or if x<-2
```

- Know your logical equivalences to simplify the test statement
- Watch out the precedence order of the sub-statements
- The operators `||` and `&&` are the short-cutting ones

Nested conditions

Conditions can be *nested*, meaning that several condition blocks may be stacked one under another

```
>> if x >= 5        % Executed only if x>=5
       y = x+2
       if y == 9;   % Executed only if y=9
          x = 2
       end
   end
```

Note: If the condition statement ruling the nested sub-block does not depend on the instructions of the outer block, it is usually preferable to use a single line multiple condition

Default case

One can specify a *default* case to execute instructions in case the wished statement is not fulfilled, with the keyword `else`

```
>> if x == 5   % Executed if x=5
      y=1;
   else        % Executed in all the
      y=3;     % other cases
   end
```

Damn, no condition is fulfilled...
What to do?

Multiple conditions (1/2)

A sequence of instructions blocks that should be executed upon ordered tests statements is defined using `if`, `elseif` and `else`

```
>> if x == 5   % Executed if x=5
       y=1;
   elseif x==6 % If the previous block was not entered,
       z=2;    % enter this one if x=5
   else
       y=3;    % Enter here if none of the above case
   end         % was successful
```

- The `else` statement is not mandatory
- Watch out blocks that are never considered

```
>> x=6; if x>=5; y=1; elseif x==6; y=0; end
```

Multiple conditions (2/2)

If the multiple conditions are such that

- they act on the same variable
- the tests' natures are the same, focusing on the output's value

a convenient way to write down the conditions is to use the keyword `switch`. The runtime execution will also be faster

```
>> if x+2==1
>>    y = 2
>> elseif (x^2==2)
>>    y = x+2+4
>> else
>>    error("Oops")
>> end
```

```
>> switch(x+2)
>>    case 1
>>       y = 2
>>    case 2
>>       y = x+2+4
>>    otherwise
>>       error("Oops")
>> end
```

Condition statements

Statements involving predefined variables

When the obtained results are close to machine precision or do not have a floating point representation on 8 bytes, one can test them against predefined variables

```
>> eps
>> 1+eps == 1
>> 1+3*eps/4 == 1
>> 1+eps/2 == 1
>> 1-eps/4 == 1
```


```
>> nan == Inf
>> 1/0 == NaN
>> 10^308 == Inf
>> 10^309 == Inf
```

Mathematical specificities or errors are also handled similarly

```
>> 0 == -0
```

```
>> 1/0
```

```
>> 0/0
```

Statements involving predefined variables

When the obtained results are close to machine precision or do not have a floating point representation on 8 bytes, one can test them against predefined variables


```
>> eps
>> 1+eps == 1
>> 1+3*eps/4 == 1
>> 1+eps/2 == 1
>> 1-eps/4 == 1
```

文本

```
>> nan == Inf
>> 1/0 == NaN
>> 10^308 == Inf
>> 10^309 == Inf
```

→ It is linked to `realmin`

Mathematical specificities or errors are also handled similarly

```
>> 0 == -0
```

```
>> 1/0
```

```
>> 0/0
```

Loops

Loops are instructions that repeat a same block of instruction upon an evolving variable. Each step of the loop is called an *iteration*

```
TypeOfLoop (IterationControl)
     Block of instructions that should be
     repeated, possibly depending on the
     iterated variable's value
EndOfLoop
```

- Useful to carry out the same command multiple times
- Possible to create nested loops (though usually not advised)
- Different types of control on the iterated variable depending on the aim of the loop: `for` and `while` loops

For loops

A *for* loop is a loop that iterates a predefined number of times. The iteration is controlled by the definition of a given list of iterates (in a vector format)

```
for iterate=start:end
   (Instructions)
end
```

```
>> for k=1:10
     disp(k^2)
   end
```

```
>> for k=[4.0,2.1]
     disp(k^2)
   end
```


```
>> for k=1
     disp(k^2)
   end
```

For loops

A *for* loop is a loop that iterates a predefined number of times. The iteration is controlled by the definition of a given list of iterates (in a vector format)

```
for iterate=start:end
   (Instructions)
end
```

```
>> for k=1:10
     disp(k^2)
   end
```

```
>> for k=[4.0,2.1]
     disp(k^2)
   end
```


```
>> for k=1
     disp(k^2)
   end
```

→ 1

While loops

A *while* loop executes the code's block until a specified condition is fulfilled. Only the change in the values of the variables used within the loop can be used to define a *stopping criterion*

```
>> a=0;
>> while a<5
     a=a+1;
   end
```

```
>> a=6;
>> while a>5
     a=a+1;
   end
```

```
>> a=0;b=1;
>> while a<b
     a=a+1;
     b=0.5*a;
   end
```

Note:

- The variables involved in the stopping criterion should be *initialized* before the loop
- `ctrl + c` interrupts a script

Control keywords

On the top of the iteration instruction that defines the loop, it is possible to control the loop flow from the instruction block itself by the keywords

`break`: interrupts the iteration and jumps to below the loop

`continue`: jumps to the instruction block's star. In a for-loop, the iterated value is updated to the next one

```
>> for k=1:10
     disp(k);
     continue; % Skips below
     a=1/0;    % Not done
   end
```

```
>> k=0;
>> while k<10
     disp(k);
     break; % Breaks
     a=1/0; % Not done
   end
```

Variables scope

In Matlab®, the variables defined or updated inside a condition statement or a loop are accessible from outside the code's block

```
>> clear x
>> for k=1:3
      x=2*k;
   end

>> disp(x)
>> disp(k)
```

The value retrieved outside the loop corresponds to:

- the last value assigned within the loop
- the last value of the iterated variable

Control flow

Best practice

- *Indent* the code's blocks that are subject to conditions or loops, and keep the *indentation level* consistent

- Always write a safety condition that stops a while loop
- Try not to call the iterated variable(s) as i, j
- Avoid nested loops

Exercises

Exercises: Condition statements & Loops

Exercise

1. Write a script that contains each of the keywords
   - if
   - for
   - while
   - continue
   - break
   - a call to a self implemented function

   and check its right execution through the debugger.

Exercises: Condition statements & Loops

Exercise

2. Write a function that takes a positive integer n and computes the members of the Fibonacci sequence $1, 1, 2, 3, 5, 8, ..$
   a) once without using loops as a recursive function $f(n) = f(n-1) + f(n-2) \quad \forall n > 2, \quad f(1) = f(2) = 1.$
   b) once using loops

   Do not forget to write a documentation for the implemented functions

Exercises: Condition statements & Loops

Exercise

3. Check numerically that:

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

   a) Write a function that takes a continuous function f and two values (boundary) a, b, such that $f(a) \cdot f(b) < 0$. The function will return the x for which $f(x) = 0$. This is done through the bisection method on the interval $[a, b]$.
   b) Check the function on `sin` to compute `pi`.

Exercises: Condition statements & Loops

Exercise

4. We are interested in finding numerically the zeros of a function.
   a) Write a function, that takes as input a function f, its derivative f' and an initial value x_0. This function will give as output a zero of f found with the Newton method, that reads

   $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

   Be careful, $f'(x) \neq 0$, so, choose properly x_0.
   b) Check your implemented function using the function `sin` to compute `pi`.