7 Programming

Scilab is a language that provides structured instructions for writing complex programs. We first formalize some notions of syntax that we have already encountered and used.

7.1 Variables and assignment

Different values (or arrays of values) can be stored in variables, a variable being identified by a name. A variable name consists of a letter possibly followed by other letters and/or digits without any space.

In Scilab, a variable is defined *interactively* by assigning it the result of an expression.

<table>
<thead>
<tr>
<th>Algorithmic notation</th>
<th>Scilab syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable ← expression</td>
<td>variable = expression</td>
</tr>
</tbody>
</table>

7.2 Test

◊ The statement *if* allows to execute a sequence of instructions if and only if an expression is true.

<table>
<thead>
<tr>
<th>Algorithmic notation</th>
<th>Scilab syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>if boolean_expression then instructions end_if</td>
<td>if boolean_expression then instructions end</td>
</tr>
<tr>
<td>if boolean_expression then instructions else instructions end_if</td>
<td>if boolean_expression then instructions else instructions end</td>
</tr>
</tbody>
</table>

◊ We can also perform several tests one after the other.

Multiple tests
Algorithmic notation | Scilab syntax
---|---
`if boolean_expression then
| instructions
else
| if boolean_expression then
| instructions
| instructions
else
| if boolean_expression then
| instructions
| instructions
end_if
| end

Example: Create the following script with name `ex_test.sce`

```scilab
n = input("Enter an integer n : ")
disp(n)
if n>2 then
disp("n is greater than 2")
end
if n==0 then
disp("n is zero")
elseif n>0 then
disp("n is strictly positive")
else
disp("n is strictly negative")
end
```

Then run this script `ex_test.sce` several times, with different input values for `n`.

We recall the usual comparison operators and Boolean operators are the followings.

<table>
<thead>
<tr>
<th>Comparison operators</th>
<th>Boolean operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>EQUAL TO</td>
</tr>
<tr>
<td>~=</td>
<td>DIFFERENT FROM</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>DIFFERENT FROM</td>
</tr>
<tr>
<td>&lt;</td>
<td>STRICTLY SMALLER THAN</td>
</tr>
<tr>
<td>&lt;=</td>
<td>SMALLER OR EQUAL TO</td>
</tr>
<tr>
<td>&gt;</td>
<td>STRICTLY GREATER THAN</td>
</tr>
<tr>
<td>&gt;=</td>
<td>GREATER OR EQUAL TO</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td>OR</td>
</tr>
<tr>
<td>~</td>
<td>NO</td>
</tr>
</tbody>
</table>

7.3 Conditional loop

The statement `while` is used to repeat a sequence of instructions as long as a Boolean expression is true.
### Example:
Create the following script with name `ex_loop1.sce`

```scilab
// We define two integers a and b such that 0 < b < a
a = 66; b = 48;
mprintf("a = %d , b = %d \n", a, b)
// we save the initial values of a and b (in aa and bb)
aa = a; bb = b;
while b>0
    r = b;
    b = pmodulo(a,b); // remainder of the eucl. div. of a by b
    a = r;
end
mprintf("the GCD of %d and %d is %d\n", aa , bb , a)
```

Then run the script `ex_loop1.sce`.
Determine the GCD of the two integers \( a = 715 \) and \( b = 546 \)

- **break**. — You can exit from a loop with the statement `break`

```scilab
k = 0;
while 1 == 1
  k = k+1;
  disp(k)
  if k > 20 then
    break
  end
end
```

### 7.4 Unconditional loop

- The statement `for` is used to repeat a sequence of instructions for a set of values.

**Unconditional loop**

<table>
<thead>
<tr>
<th>Algorithmic notation</th>
<th>Scilab syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for variable in array_of_values do instructions end_for</code></td>
<td><code>for variable = array_of_values instructions end</code></td>
</tr>
</tbody>
</table>

- It is commonly used with the following syntax.

<table>
<thead>
<tr>
<th>Algorithmic notation</th>
<th>Scilab syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for variable from v_min to v_max do instructions end_for</code></td>
<td><code>for variable = v_min:v_max instructions end</code></td>
</tr>
</tbody>
</table>
and instructions are runned for \( \text{variable} = v_{\text{min}}, \text{variable} = v_{\text{min}} + 1, \ldots \),
up to \( \text{variable} = v_{\text{max}} \)

*Example* : Create the following script with name `ex_loop2.sce`

```scilab
disp("Example 1")
for i = 1:10
    disp(i)
end

disp("Example 2")
tab_v = [2.4 7.4 8 -3.1 9.5 0.1];
sum_t = 0;
min_t = tab_v(1);
for i = 1:length(tab_v)
    v = tab_v(i)
    disp(v)
    sum_t = sum_t + v;
    if v < min_t then
        min_t = v
    end
end
mprintf("The sum of elements of tab_v is %f\n", sum_t);
mprintf("The min of elements of tab_v is %f\n", min_t);
```

Then run this script `ex_loop2.sce`.

- **break**. — You can exit from a loop with the statement `break`

```scilab
for k = 1:1000
    disp(k)
    if k > 20 then
        break
    end
end
```

### 7.5 Recursive functions

Recursion in programming is a method where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to iteration).

- A **recursive function** is a function that calls itself

- As an example, to compute the factorial \( \text{fact}(n) \) of an integer \( n \), we can apply the relation

  \[
  \text{if } n == 0 \text{, } \text{fact}(n) = 1, \text{ else } \text{fact}(n) = n \times \text{fact}(n-1)
  \]

*Example* : Create the following script with name `ex_Factorial.sci`. Then compare the two functions.
// Iterative function
function res = MyIterativeFact(n)
    // n is assumed to be an integer >= 0
    res = 1;
    if n>0 then
        for k=1:n
            res = res * k
        end
    end
endfunction

// Recursive function
function res = MyRecursiveFact(n)
    // n is assumed to be an integer >= 0
    if n==0 then
        res = 1;
    else
        res = n * MyRecursiveFact(n-1)
    end
endfunction

Example : In order to better understand the execution of a recursive function, first create the following script with name ex_RecursiveOrder.sce, then run this script with the value N=0

// Order of execution of a recursive function
function recursiveFunction(N)
    printf("Before, N = %d\n", N);
    if N < 4 then
        recursiveFunction(N+1)
    end
    printf("After, N = %d\n", N);
endfunction

The execution scheme of the instruction recursiveFunction(0) can be described as follows (from Wikipedia).
7.6 Exercises

Exercise 15

Write a script Scilab named exoLoop2.sce that complete the previous file ex_loop2.sce so as to determine and display:

- the product (and the sum) of all the elements of the array tab_v,
- the maximum (and the minimum) of elements of the array tab_v.

Exercise 16

Write a Scilab script named exoSequenceSqrt.sce that performs the following:

- ask the user to enter a positive value $a$,
- compute the vector $u$ composed of $n = 10$ values defined as follows:
  \[ u(1) = \frac{a + 1}{2} \quad \text{and for } k \in \{2, 3, \ldots, n\}, \quad u(k) = \frac{a}{2} u(k - 1) + \frac{u(k - 1)}{2} \]
- display the values of the vector $u$ together with the squares of the values of vector $u$.

Can you interpret this result?

Exercise 17

Write a Scilab script named exoSumSquaresCubes.sce that performs the following.

1) Determine the sum of the squares and the cubes of all positive integers less than or equal to a given integer $n$.
2) Control the result of your sums with the following formulas
   \[ S2(n) = \sum_{k=0}^{n} k^2 = \frac{n(n + 1)(2n + 1)}{6}, \quad S3(n) = \sum_{k=0}^{n} k^3 = \left( \frac{n(n + 1)}{2} \right)^2 \]
3) Determine the greatest integer $n$ such that $S3(n) < 50000$. 

32 Introduction to Scilab
Exercise 18

We propose to calculate an approximation of the exponential of the real value \( x = 2 \) by means of its Taylor series

\[
\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots
\]

by using only the four classical operators +, −, *, /.

1) Experiment (and run) the following script that you will save as `exoTaylorSeries.sce`

```scilab
// Approximation of exp(x) by its Taylor series
// unconditional loop
x = 2;
N = 8; // number of terms in the Taylor series
val = 1; // approximation value
power_x = 1;
fact = 1;
for k = 1 : N
    power_x = power_x * x;
    fact = fact * k;
    val = val + power_x / fact;
end
error = abs(exp(x) - val);
disp(error)
```

Modify the value of \( N \) to get an approximation with a precision less than \( 10^{-4} \).

2) In order to control the number of terms of the Taylor series according to the desired precision, we propose to use a conditional loop instead of the unconditional loop.

Complete accordingly the following Scilab script (that you will add in the file `exoTaylorSeries.sce`)

```scilab
// Approximation of exp(x) by its Taylor series
// conditional loop
epsilon = 0.00001;
x = 2;
e2 = exp(x);
val = 1; // approximation value
power_x = 1;
fact = 1;
k = 0; // number of terms
while abs(e2 - val) > epsilon
    ...
end
disp(k)
disp(abs(e2 - val))
```

Run this file for different values of \( x \) and \( \epsilon \).

Exercise 19

Write a script Scilab named `exoEratosthenes.sce` that determine all prime numbers less than or equal to a given integer \( n \) by applying the sieve of Eratosthenes.
Precisely, you have to perform the following steps.
1) Create a vector \texttt{listNb} of integers between 1 and \( n \).
2) For each integer \( k \) between 2 and \( \sqrt{n} \), mark the multiples of \( k \), except \( k \):
   \[
   \texttt{listNb}(2*k) = 0, \texttt{listNb}(3*k) = 0, \ldots
   \]
3) Finally, copy all the non marked integers (the non zero integers) of your vector \texttt{listNb} into a new vector \texttt{primeNb}. Be aware that 1 is not a prime number.

Example with \( n = 500 \):

\[
\begin{align*}
\end{align*}
\]

Exercise 20

Save the following Scilab script as \texttt{exoSort.sce}

```scilab
function k = posMin(vect,k0)
    k = k0;
    m = vect(k0);
    for i = k0+1 : length(vect)
        if vect(i) < m then
            m = vect(i);
            k = i;
        end
    end
endfunction

function tab = MySort(tab0)
    tab = tab0;
    n = length(tab);
    for i = 1 : n-1
        k = posMin(tab,i);
        if k <> i then
            tmp = tab(k);
            tab(k) = tab(i);
            tab(i) = tmp;
        end
    end
endfunction

tab0 = [2 -3 0 -5 -3 1 7 -5 3];
tab = MySort(tab0)
```

1) Experiment this script and determine its purpose.
2) Comment each (main) line of this program.
3) In order to test the efficiency of this program, test the following instructions in the console.

```scilab
help gsort
help rand
```
\[
\begin{align*}
n &= 1000; \quad // \text{then } n = 10000; \\
tab0 &= \text{rand}(1,n); \\
tab1 &= \text{gsort}(\text{tab0},'g','i'); \\
tab2 &= \text{MySort}(\text{tab0});
\end{align*}
\]

You can test with \( n = 10000 \), but be patient :) ... You may block the console!

→ If you block the console: CTRL C and then abort