OPEN SOURCE ENGINEERING

openeering

A SCILAB PROFESSIONAL PARTNER

powered by ENGIN SOFT

SCILAB AS A CALCULATOR

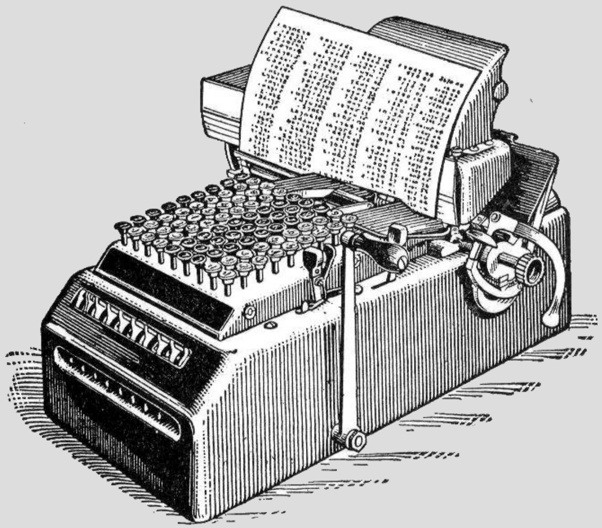
The purpose of this tutorial is to get started using Scilab as a basic calculator by discovering some predefined data types and functions.

Level

www.openeering.com

Step 1: The purpose of this tutorial

In the tutorial “First steps with Scilab” we have introduced to the user the Scilab environment and its features and here the aim is to make him/her comfortable with Scilab basic operations.



Step 2: Roadmap

In this tutorial, after looking over Scilab basic predefined data types and functions available in the environment, we will see the usage of variables, how to define a new variable and some operations on numbers.

We will apply the acquired competencies for the resolution of a quadratic equation of which we know the solution.

Descriptions	Steps
Basic commands and operations	3-8
Predefined variables	9
Arithmetic and formats	10-12
Variables	13-16
Functions	17
Example	18
Conclusions and remarks	19-20

Step 3: Scilab as a basic calculator

Scilab can be directly used to evaluate mathematical expressions.

`ans` is the default variable that stores the result of the last mathematical expression (operation). `ans` can be used as a normal variable.

```
0.4 + 4/2
```

```
ans =
    2.4
```

Step 4: Comments

A sequence of two consecutive slashes `//` out of a string definition marks the beginning of a comment. The slashes as well as all the following characters up to the end of the lines are not interpreted.

```
// This is a comment
// Let's divide the previous value by two
0.4 + 4/2
ans/2
```

```
ans =
    2.4
ans =
    1.2
```

Step 5: Basic mathematical operators

Basic mathematical operators:

- addition and subtraction: `+`, `-`
- multiplication and division: `*`, `/`
- power: `^`
- parentheses: `()`

```
(0.4 + 4)/(3-4^0.5) // A comment after the command
```

```
ans =
    4.4
```

Step 6: The Scilab operator “,”

The Scilab operator , can be used to separate expressions in the same row.

```
// Two expressions
1*2 , 1.1 + 1.3
```

```
ans =
    2.
ans =
    2.4
```

Step 7: The Scilab operator “...”

The Scilab operator ... can be used to split an expression in more than one row.

```
// The expression is toooooooo long
1 + 1/2 + 1/3 + ...
1/4 + 1/5 + ...
1/6
```

```
ans =
    2.45
```

Step 8: The Scilab operator “;”

The Scilab operator ; is used to suppress the output, which will not be displayed in the Console.

```
// An expression
1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6;
// The result is stored in the ans variable
ans
```

```
ans =
    2.45
```

The command ; can also be used to separate expressions (in general statements, i.e. Scilab commands) in the same row.

Step 9: Predefined variables

In Scilab, several constants and functions already exist and their names begin with a percent character `%`.

For example, three of the main variables with a mathematical meaning are

- `%e`, which is the Euler's constant *e*
- `%pi`, which is the mathematical constant π
- `%i`, which is the imaginary number *i*

In the example on the right we have displayed the value of π and its sinus through the use of the Scilab sinus function `sin`. We should obtain $\sin \pi = 0$, but we get a really close to zero value because of the machine rounding error.

```
%pi // pi = 3.1415....
sin(%pi)
```

```
ans =
    3.1415927
ans =
    1.225D-16
```

Step 10: Complex arithmetic

Also complex arithmetic is available. `%i`, is the imaginary unit *i*

On the right we get the imaginary unit also computing the square root of -1 and the Euler relation returns a really close to zero value because of the machine rounding error.

```
%i // imaginary unit
sqrt(-1)
exp(%i*%pi)+1 // The famous Euler relation
```

```
ans =
    i
ans =
    i
ans =
    1.225D-16i
```

Step 11: Extended arithmetic

In Scilab, the “not a number” value `Nan` comes from a mathematically undefined operation such as 0/0 and the corresponding variable is `%nan`, while `Inf` stands for “infinity” and the corresponding variable is `%inf`.

```
ieee(2) // set floating point exceptions for Inf and
Nan
1/0
0/0, %inf*%inf, %inf*%nan
ieee(0) // unset floating point exceptions for Inf and
Nan
1/0
0/0
```

The command `ieee()` returns the current floating point exception mode.

0 → floating point exception produces an error

1 → floating point exception produces a warning

2 → floating point exception produces `Inf` or `Nan`

```
ans =
    Inf
ans =
    Nan
ans =
    Nan
ans =
    Nan


    !--error 27
Division by zero...

    !--error 27
Division by zero...
```

The command `ieee(mod)` sets the current floating point exception mode. The initial mode value is 0.

Step 12: Change the visualization format

All computations are done in **double precision arithmetic**, although the visualization format may be limited.

Using the command `format` the option `'e'` sets the e-format, while `'v'` sets the variable one. We can also choose the number of digits to visualize.

```
format('v',20); %pi // Change visualization
format('e',20); %pi // Change visualization
format("v",10); %pi // Restore original visualization
```

```
ans =
    3.14159265358979312
ans =
    3.1415926535898D+00
ans =
    3.1415927
```

Step 13: Defining new variables

Syntax:

`name of the variable = expression`

where `expression` can involve other variables.

Some constraints:

- Variable names can be up to 24 characters long
- Variable names are case sensitive (variable A is different from a)
- The first letter must be an alphabetic character (a-A, z-Z) or the underscore character (_)
- Names must not contain blanks and special characters

The `disp` command is used to display data to the console.

```
// Define variables a and b
a = 4/3;
b = 3/4;
// Define variable c as expression of a and b
c = a*b;
// Display the result
disp(c)
```

```
    1.
```

Step 14: String variables

String variables are delimited by quotes characters of type `"` or `'`.

The command `string` converts a number into a string.

```
// Two strings
a = 'Hello';
b = 'World';
// String concatenation
c = a + " " + b + "!" ;
disp(c);
// Concatenation of a string with a number
d = "Length of " + a + " is " + string(length(a))
```

```
Hello World!

d =
  Length of Hello is 5
```

Step 15: Boolean variables

Boolean variables are used to store true (`%t` or `%T`) or false data (`%f` or `%F`) typically obtained from logical expressions.

The comparison operators are:

- `<` : Less than
- `<=` : Less than or equal to
- `==` : Equal to
- `>=` : Greater than or equal to
- `>` : Greater than

```
// Example of a true expression
res = 1>0
// Example of a false expression
res = 1<0
```

```
res =
    T
res =
    F
```

Step 16: Main advantages using Scilab

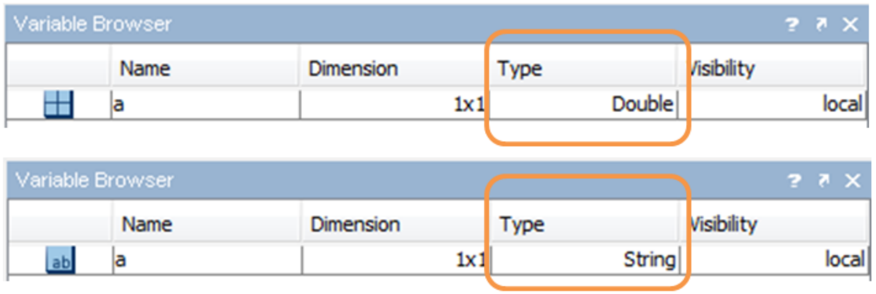
When working with variables in Scilab we have two advantages:

- Scilab does not require any kind of declaration or sizing
- The assignment operation coincides with the definition

In the example on the right we have not declared the type and the size of `a`: we just assigned the value 1 to the new variable.

Moreover, we have overwritten the value 1 of type double contained in `a` with the string `Hello!` by simply assigning the string to the variable.

In the Variable Browser we can see that the type of `a` changes outright:




```
// a contains a number
a = 1;
disp(a)
// a is now a string
a = 'Hello!';
disp(a)
```

```
1.

Hello!
```

Step 17: Scilab functions

Many built-in functions are already available, as you can see in the table on the right. Type in the Console the command `help` followed by the name of a function to get the description, the syntax and some examples of usage of that function.

Field	Commands
Trigonometry	`sin, cos, tan, asin, acos, atan, sinh, cosh, ...`
Log - exp – power	`exp, log, log10, sqrt, ...`
Floating point	`floor, ceil, round, format, ieee, ...`
Complex	`real, imag, isreal, ...`

In the examples on the right you can see different ways to set input and output arguments.

```
// Examples of input arguments
rand
sin(%pi)
max(1,2)
max(1,2,5,4,2)
```

```
// Examples of output arguments
a = rand
v = max(1,2,5,4,2)
[v,k] = max(1,2,5,4,2)
```

Step 18: Example (quadratic equation)

The well-known solutions of a quadratic equation

$$ax^2 + bx + c = 0$$

are

$$x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$$

where $\Delta = b^2 - 4ac$. If $\Delta < 0$ solutions are imaginary.

We assess the implementation on the following input data:

Coefficient	Value
a	+3.0
b	-2.0
c	-1.0/3.0

where the solutions are

$$x_{1,2} = \frac{1 \pm \sqrt{2}}{3}$$

On the right you can find the implementation and the validation of the numerical solutions with respect to the exact solutions.

```
// Define input data
a = 3; b = -2; c = -1/3;
// Compute delta
Delta = b^2-4*a*c;
// Compute solutions
x1 = (-b+sqrt(Delta))/(2*a);
x2 = (-b-sqrt(Delta))/(2*a);
// Display the solutions
disp(x1); disp(x2);
```

```
    0.8047379

  - 0.1380712
```

```
// Exact solutions
x1e = (1+sqrt(2))/3
x2e = (1-sqrt(2))/3
// Compute differences between solutions
diff_x1 = abs(x1-x1e)
diff_x2 = abs(x2-x2e)
```

```
x1e =
    0.8047379

x2e =
  - 0.1380712

diff_x1 =
        0.

diff_x2 =
        0.
```

Step 19: Concluding remarks and References

In this tutorial we have introduced to the user Scilab as a basic calculator, in order to make him/her comfortable with Scilab basic operations.

1. Scilab Web Page: www.scilab.org.
2. Openeering: www.openeering.com.

Step 20: Software content

To report a bug or suggest some improvement please contact Openeering team at the web site www.openeering.com.

Thank you for your attention,

Anna Bassi, Manolo Venturin

```
----------------------
SCILAB AS A CALCULATOR
----------------------


--------------
Main directory
--------------
license.txt                : the license file
example_calculator.sce     : examples in this tutorial
```