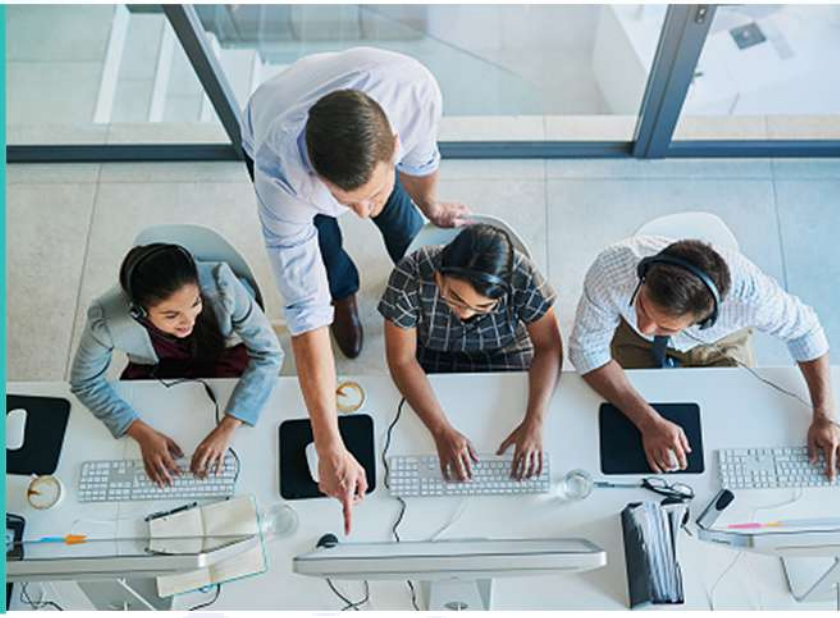
# Andhra Pradesh State Skill Development Corporation



# SciLab

## Programming in Scilab Functions

# Functions in Scilab

## Functions

Functions are used to calling the functions. Which is also used for reducing the code lines.
In this section, we present Scilab functions. We analyze the way to define a new function and the method to load it into Scilab.

Syntax:

```
function f=fun(variable)
 ------
 -------//operations
 endfunction
```

## Overview

Gathering various steps into a reusable function is one of the most common tasks of a Scilab developer.

The most simple calling sequence of a function is the following:

outvar = myfunction ( invar ) where the following list presents the various variables used in the syntax:

- myfunction is the name of the function,
- invar is the name of the input arguments,
- outvar is the name of the output arguments.

The values of the input arguments are not modified by the function, while the values of the output arguments are actually modified by the function. We have in fact already met several functions in this document. The sin function, in the y=sin(x) statement, takes the input argument x and returns the result in the output argument y. In Scilab vocabulary, the input arguments are called the right-hand side and the output arguments are called the left-hand side. Functions can have an arbitrary number of input and output arguments so that the complete syntax for a function that has a fixed number of arguments is the following:

[o1 , ... , on ] = myfunction ( i1 , ... , in )

The input and output arguments are separated by commas ”,”. Notice that the input arguments are surrounded by opening and closing braces, while the output arguments are surrounded by opening and closing square braces.

## Defining a function

To define a new function, we use the function and endfunction Scilab keywords. In the following example, we define the function myfunction, which takes the input argument x, multiplies it by 2, and returns the value in the output argument y.

```
function y = myfunction ( x )
y = 2 * x
endfunction
```

There are at least three possibilities to define the previous function in Scilab.

• The first solution is to type the script directly into the console in an interactive mode. Notice that, once the ”function y = myfunction ( x )” statement has been written and the enter key is typed in, Scilab creates a new line in the console, waiting for the body of the function. When the ”endfunction” statement is typed in the console, Scilab returns back to its normal edition mode.

• Another solution is available when the source code of the function is provided in a file. This is the most common case since functions are generally quite long and complicated. We can simply copy and paste the function definition into the console. When the function 38 definition is short (typically, a dozen lines of source code), this way is very convenient. With the editor, this is very easy, thanks to the Load into Scilab feature.

• We can also use the exec function. Let us consider a Windows system where the previous function is written in the file ”C:\myscripts\examples-functions.sce”. The following session gives an example of the use of exec to load the previous function.

```
--> exec ( "C :\ myscripts \ examples - functions . sce " )
 --> function y = myfunction ( x )
--> y = 2 * x
 --> endfunction
```

The exec function executes the content of the file as if it were written interactively in the console and displays the various Scilab statements, line after line. The file may contain a lot of source code so that the output may be very long and useless. In these situations, we add the semicolon character ”;” at the end of the line. This is what is performed by the Execute file into the Scilab feature of the editor.

```
--> exec ( "C :\ myscripts \ examples - functions . sce " );
```

Once a function is defined, it can be used as if it was any other Scilab function.

```
--> exec ( "C :\ myscripts \ examples - functions . sce " );
-->y = myfunction ( 3 )
 y  =
6.
```

Notice that the previous function sets the value of the output argument y, with the statement y=2*x. This is mandatory. In order to see it, we define in the following script a function that sets the variable z, but not the output argument y.

```
function y = myfunction ( x )
 z = 2 * x
endfunction
```

Lets see some more examples on functions.

Example1:

```
function s=sum(a, b, c)
s=a+b+c
endfunction
```

```
Output:-->exec('C:\Users\DELL\a.sci', -1)
 --> sum(2,3,5)
       10.
```

Example2:

```
function [x, y,z]=sum(a, b, c)
        x=a+b+c
        y=a-b-c
        z=a*b*c
endfunction
```

```
Output:
      --> exec('C:\Users\DELL\a.sci', -1)
      --> [x,y,z]=sum(2,3,5)
           z = 30.
           y = -6.
           x = 10
```