6.2.5 User-defined functions

In addition to many internal pre-programmed functions, ILWIS gives the user the opportunity to create new functions. They may be used in all ILWIS calculators such as: MapCalc, TabCalc or in the pocket line calculator. Especially when you need to execute certain calculations which require a lot of typing work several times, user-defined functions may be time saving. A user-defined function is an expression, which may contain any combination of operators, functions, variables and parameters resulting in one output value.

Creating and editing a user-defined function
To create a user-defined function:

- Choose File, Create, Create Function in the Main window, or
- Double-click New Function in the Operation-list

A dialog box appears in which you can fill in the name of your new function (starting with a character from A to Z and not exceeding 8 characters). Then, you enter an expression and optionally you give a short description of the function in the description box. The different parameters in your function may be characters (*a,b,c* etc.) but it is recommended to give them logic names. Clear names make the application of user-defined functions much easier.

☞ The parameters as well as the output map or column are assigned the value domain in ILWIS. If this is not the intention you have to change that manually in the Edit Function dialog box
☞ Parameter domain types which may be used are: Value, String, Coord, Color, Bool
☞ Return domain types which may be used are: any domain available as well as Coord.

- Click OK.

The Edit Function dialog box appears on the screen, showing the newly created function. If necessary you can edit the function in this dialog box until it satisfies your purposes.

To edit a user-defined function

- Open the Edit menu from the Main window, select Edit Object; in the appearing dialog box select Function from the drop-down list and select the function which you want to edit, or
- Press the right mouse button on a function in the Catalog and select Open from the context-sensitive menu.

Applying a user-defined function
You have to apply your function on the command line of the Main window or a table window.

The syntax to apply a user-defined function is:

```
Output = Function(a,b,c...etc.)
```

where:

Output	is the output map name (when applied on the command line of the Main window) or output column name (when applied on the command line of the table window)
Function	is the name of the user-defined function
a,b,c...etc.	are the parameters such as values, map names or table column names, used in the function; they should be typed in the same order as they appear in the function definition

Examples of a user-defined functions

Example 1

To define a function SUM to sum two values e.g. bands of an image:

- Double-click the New Function item in the Operation-list of the Main window. The Create Function dialog box appears on the screen.
- Type SUM for the function name and type `a + b` in the expression box.

  This function is defined as follows:

```
Function sum(Value a,Value b) : Value
Begin
 Return a + b;
End;
```

  which means:
  The values a and b are created and the output of the function will also be a value. The calculation itself is defined as: return the sum of a and b.

- You apply your function on the command line of the Main window or a table window by typing:

  `Result1 = SUM (band1, band2)`

  where `Result1` is the output object (value domain) and `band1` and `band2` are the parameters replacing the a and b in the function. This example showed the functionality of a user-defined function. But in fact this was so simple that in such a case it is more convenient to type immediately on the command line:

  `Result = band1 + band2`

Example 2

Another, and a little bit more complex example is the calculation of the seasonal rainfall and express it as the relative amount of total rainfall. In this example, a table named `rainfall` is used. It shows the amount of rainfall per month for several rainfall stations. A new column `total` was already calculated, showing the total rainfall per year for each rainfall station.

- Double-click the New Function item in the Operation-list of the Main window. The Create Function dialog box appears on the screen.

- Type `Season` for the function name and type in the expression box:

  ```
  100 * (month1 + month2 + month3) / totrain
  ```

  To indicate the parameters, it is advised to use logic names instead of just characters such as in the former example. It is recommended to do so because clear names make the application of user-defined functions much easier.

- Optionally you may fill in the description box. Type: `seasonal rainfall as relative amount of total rainfall`.
- Click the OK button after which the Edit Function dialog box appears on the screen. You can see that the function is defined as follows:

```
Function season(Value month1,Value month2,Value month3,Value
totrain) : Value
Begin
  Return 100 * (month1 + month2 + month3) / totrain;
End;
```

  which means:
  parameters `month1, month2,month3` and `totrain` are values and the output of the function will also be a value.

In the Edit Function dialog box, you may edit your function until it satisfies your purposes.

In this example the output value domain is changed into domain `Perc` (Percentage) as follows:

- Highlight the word **Value** with the mouse pointer which indicates the output domain (the one after the colon **:**) and change it to **Perc** (which is a system domain). Now the definition of the function is:

```
Function season(Value month1,Value month2,Value month3,Value
totrain) : Perc
Begin
  Return 100 * (month1 + month2 + month3) / totrain;
End;
```

- When you apply this function for the first three months of the year, type on the command line of the table window:

  ```
  Result2 = SEASON (january, february, march, total)
  ```

`Result2` stores the relative amount of rainfall which fell in the months January, February and March compared to the total rainfall in that year.

Example 3
You can also use ID or class domains in user-defined functions.
In this example a table is used with information on parcels. Imagine that you want to build extra houses, but only on parcels which are already in residential use and only when there is enough space to build. If the parcels satisfy certain conditions, specified in the user-defined function, they are assigned "suitable" else "not suitable".

- Double-click the New Function item in the Operation-list of the Main window. The Create Function dialog box appears on the screen.
- Type `Suitable` for the function name and type in the expression box:

```
iff ((landuse = "residential") and (area/nrhouses > 10000), "suitable", "not suitable")
```

Because ILWIS assigns by default the `system Value` domain to all input parameter of the function and to the output of the function, an error message appears to warn you that ILWIS does not recognize the quotation mark " before the word `residential`. This needs to be changed in the Edit Function dialog box.

- Click OK in the Create Function dialog box; then the Edit Function dialog box appears on the screen showing:

```
Function suitable(Value landuse,Value area,Value nrhouses) : Value
Begin
  Return  iff ((landuse = "residential") and (area/nrhouses > 10000)
, "suitable", "not suitable");
End;
```

- Change **Value** landuse to **String** landuse
- Change : **Value** to : **String**

Now the function looks like:

```
Function suitable(String landuse,Value areasize,Value nrhouses) :
String
Begin
  Return  iff ((landuse = "residential") and (areasize/nrhouses >
10000)  , "suitable", "not suitable");
End;
```

- Click OK in the Edit Function dialog box.
- Now the user-defined function can be applied by typing on the command line of the `Parcel` table window:

```
Suit = suitable (Landuse, Area, Units)
```

Parcel	Landuse	Area	Units	Suit
100	Residential	58200	5	suitable
124	Residential	16600	4	not suitable
157	Commercial	82540	12	not suitable
162	Residential	65925	4	suitable
181	Industrial	8365	3	not suitable
202	Institutional	82470	5	not suitable
225	Residential	19830	7	not suitable
269	Commercial	52945	9	not suitable
288	Institutional	29400	6	not suitable
295	Residential	7235	4	not suitable

Only parcels having a residential landuse as well as an area per unit over 10000 square meters are assigned "`suitable`".

User-defined functions in slope instability hazard analysis
This example shows an advanced user-defined function.

The hazard degree of slope instability can be expressed by the Safety Factor (F). This is the ratio between the forces that make the slope fail and those that prevent the slope from failing. F-values larger than 1 indicate stable conditions, and F-values smaller than 1 instable conditions. At F=1 the slope is at the point of failure. Many different models exist for the calculation of Safety Factors. Here one of the simplest models is used.

Performing the model for each independent pixel, will result in a hazard map of safety factors. The safety factor is calculated according the following formula (Brunsden and Prior, 1979) :

$$F = \frac{c' + (\gamma - m\gamma_w) z \cos^2\beta \tan\phi'}{\gamma z \sin\beta \cos\beta}$$

Below, the meaning of the symbols is explained briefly completed with average values from laboratory analysis

c'	= effective cohesion (Pa= N/m^2)	= 10000 Pa
γ	= unit weight of soil (N/m^3)	= 11000 N/m^3
m	= relative water depth	= z_w/w (dimensionless)
γ_w	= unit weight of water (N/m^3)	= 10000 N/m^3
z	= depth of failure surface below the surface (m)	= map ASHT
z_w	= height of water table above failure surface (m)	
β	= slope surface inclination (˚)	= map SLOPE
Φ	= effective angle of shearing resistance (/)	= 30(˚)
tan(ϕ')	= tangent of the eff. angle of shearing resistance	= 0.58

For more detailed information on this formula refer to chapter 5 of the Applications Guide.

In this example the slope stability analysis is made by using only two parameter maps: `Asht` (thickness of volcanic ashes) and `Slope` (slope angles in degrees). The depth of the possible failure plane is taken at the contact of the volcanic ashes and the underlying material. The consequence of that is that safety factors will not be calculated for the entire area, but only for the areas where there is volcanic ash at the surface.

The only unknown parameter yet is the depth of the water table. In the formula this is expressed as the value *m*, which is the relation between the depth of the water table and the depth of the failure surface.

ILWIS does not recognize symbols such as: c' or Φ . Therefore, give a clear name to these symbols in the user-defined function. Later, while applying the user-defined function you fill in the value of the parameter represented by those symbols.

You can define the user-defined function as follows:

To create the user-defined function

- Double-click the New Function item in the Operation-list of the Main window. A dialog box appears on the screen in which you fill in the name of the function. In this example fill in `Safety`.
- Then fill in the expression. Type the following expression:

```
effectivecohesion + (soilweight - relwaterdepth *
waterweight) * failuredepth *
SQ(COS(DEGRAD(mapslope))) * tanangleres / soilweight *
failuredepth * SIN(DEGRAD(mapslope)) *
COS(DEGRAD(mapslope))
```

- Finally you may optionally fill in the description of the function. In this example fill in: `Safety factor.`
- Click OK.

The Edit Function box appears on the screen and shows the function definition:

```
Function safety (Value effectivecohesion,Value soilweight,Value
relwaterdepth,Value waterweight,Value failuredepth,Value
mapslope,Value tanangleres) : Value
Begin
  Return effectivecohesion + (soilweight - relwaterdepth *
waterweight) * failuredepth * SQ(COS(DEGRAD(mapslope))) *
tanangleres / soilweight * failuredepth * SIN(DEGRAD(mapslope)) *
COS(DEGRAD(mapslope));
End;
```

- Click OK.

Application of the user-defined function

Firstly calculate the safety factor for the volcanic ashes under the assumption that the soil is completely dry. In that case the parameter m (watfail in the expression and representing the relation between the depth of the water table and the depth of the failure surface) equals zero.

Type on the command line of the Main window:

```
fdry = Safety (10000, 11000, 0, asht, slope, 0.58)
```

This means apply the function `Safety` using the values and maps which are specified between the brackets and store the result in map `fdry`. The map shows the safety factors for extremely dry conditions. As you can imagine that a situation with a completely dry situation does not occur in many tropical regions such as for instance Manizales in Colombia.

You can easily calculate a next scenario that which will evaluate for instance a condition in which the slopes are completely saturated. This is also not a very realistic situation, but it will give us the most pessimistic estimation of slope stability.

When the soil is saturated, the m factor from formula [1] is equal to 1 which means that the water table is at the surface. There is also another factor which is different for saturated conditions:
γ = unit weight of soil (N/m^3) = 16000 N/m^3

Now you simply apply the user-defined function again, but in this case using the new values.

```
fsat = Safety (10000, 16000, 1, asht, slope, 0.58)
```

This means: apply the function `Safety` using the values and maps which are specified between the brackets and store the result in map `fsat`. The map shows the safety factors for extremely wet conditions.

As soon as you have created your user-defined function you are able to perform many similar calculations only by using different values for your parameters.

6.2.6 Pocket Line Calculator

The Pocket Line Calculator is an ILWIS tool which enables the user to make quick calculations. All operators and functions described in the chapters Map Calculation and Table Calculation are available. On the command line of the Main window you type your calculation starting with a ? which is in ILWIS the syntax for using the Pocket Line Calculator.
You may use the Pocket Line Calculator as well on the command line of a table window. This enables you to retrieve quickly some information on data in your table without creating new columns.

The pocket line calculator is especially convenient for:
- Quick calculations without the need to start other programs
- Retrieving pixel information of maps (values, classes or ID's, colors, coordinates etc.) without displaying them on the screen.
- Trying out difficult calculations before performing them on maps or table columns.

Examples of the Pocket Line Calculator

`? 7 + 8`	returns 15
`? 10 DIV 3`	returns 3
`? SIN(PI/2)`	returns 1.0
`? COS(DEGRAD(60))`	returns 0.50

(Note that for functions SIN and COS the input angle should be given in radians. To convert degrees to radians use the DEGRAD function).

```
? MAPCOLOR (MapName, rowexpr, colexpr)
```

returns the color of a pixel in a raster map using the default representation. The pixel is defined by an expression resulting in a row number and an expression resulting in a column number.

```
? MAPVALUE (Map1, COORD(100000, 100000)
```

returns the Value, Class or ID of `Map1` on the location with the specified coordinates.

```
? IFF (MAPVALUE (DEM, (COORD (100000,100000)) > 2500,
  "high", "low")
```

returns "high" if the value of the `DEM` on the specified location is higher than 2500; when the altitude is less than 2500 it returns "low".

Examples of the Pocket Line Calculator on the command line of the Table window

Many functions specifically used in TabCalc may be used in the Pocket Line Calculator as well. Note that the following functions require columns having a Value domain.

`? AVG(ColName)`	returns the average of column `ColName`
`? STDEV(ColName)`	returns the standard deviation of column ColName
`? MAX(ColName)`	returns the maximum of column ColName
`? CORR(Col`*a*`, Col`*b*`)`	returns the correlation between the columns Col*a* and Col*b*

These functions may also be used in combination with operators and functions mentioned before.

```
? IFF(Landuse = "residential",(MIN(ColumnName) +
  MAX(ColumnName) / 2)), 0)
```

☞ More complicated expressions or expressions which will be used several times may be stored as a user-defined function or in an ASCII-file. Refer for user-defined functions to the description in the sections MapCalc and TabCalc. ASCII-files can be made in any text editor. Application of such a file requires ?@*FileName* on the command line.

Example of applying an ASCII-file

The ASCII-file `Suitable` may look like:

```
IFF ((MAPVALUE (Soil, coord (150000,200000)) = "rock") or
     (MAPVALUE (Slope, coord (150000,200000)) >30),
     "unsuitable", "suitable")
```

To apply this file:

```
?@suitable
```

6.2.7 Scripts

ILWIS enables the user to perform a series of calculations by creating and applying a script. This is comparable to using batch files in ILWIS version 1.4. The application of scripts may be time saving but is something for the more advanced users.
A script may contain expressions for MapCalc, TabCalc, or any other ILWIS operation. Furthermore, some extra commands are possible to show objects, or to perform file management. For more information on script syntax, see Appendix F ILWIS Script language (syntax).

Example:
The following steps describe how a script can be created and applied to calculate a slope map in percentages and in degrees. A contour map and several MapCalc/TabCalc functions.

To calculate a slope maps in percentages and in degrees:

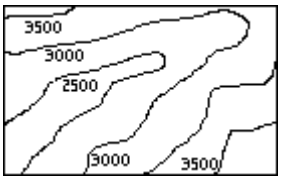
1. Create a script (e.g. Slopes)
2. In the dialog box where the script is defined:
   type, to insert a comment line:
   `//script to calculate slope maps in percentages and in degrees`
3. To use the InterpolContour operation to create a interpolated height map from segment contour lines; type:
   `%2 = MapInterpolContour(%1,geo)`
   Besides the input segment map `1%` a georeference ‘geo’ is needed.
4. To use filter DFDX on the interpolated contour map to calculate height difference in X-direction; type:
   `%3 = MapFilter(%2, dfdx)`
5. To use filter DFDY on the interpolated contour map to calculate height differences in Y-direction; type:
   `%4 = MapFilter(%2, dfdy)`
6. To calculate a slope map from these, type:
   `%6 = ((HYP(%3,%4))/%5)*100`
   HYP is an internal Mapcalc/Tabcalc function; `%3` and `%4` are the output maps from the filtering; `%5` represents the pixel size of the maps; `%6` is the output map name of the map containing slope value in percentages.
7. To convert the percentage values into degrees, type:
   `%7 = RADDEG(ATAN(%6/100))`
   Function ATAN and RADDEG are internal MapCalc/TabCalc functions.
8. To run the script, type on the command line in the Main window:
   `run Slopes contour.seg dem.mpr DX DY 20 Slopepct Slopedeg`

   In script `Slopes`, `%1` is filled out as `contour.seg`, `%2` as `dem.mpr`, `%3` as `DX`, `%4` as `DY`, `%5` as `20`, and `%6` as `SlopePct`, and `%7` as `SlopeDeg`.

Of course, you can also use objects names inside the script instead of parameters `%1` (Contour, segment map), `%2` (DEM, Digital Elevation Mode), etc.

Starting with a `Contour` map, the end result of running this script are two maps: `SlopePct` and `SlopeDeg`. These are slope maps in percentages and in degrees. Also other maps are calculated during the whole procedure, such as the `DEM`, `DX` (height difference in X-direction) and `DY` (height difference in Y-direction). The following maps show the input map and very small parts of the other maps (only 16 pixels) calculated in the script.

Contour



DEM

2976	2971	2976	2987
2971	2960	2973	2979
2960	2957	2960	2976
2957	2957	2960	2977

DX

-8.0	-0.2	9.2	9.1
-9.7	1.1	10.8	9.4
-8.0	-0.4	10.0	17.1
-1.1	0.6	10.3	17.7

DY

6.3	11.3	3.7	4.7
8.3	7.3	8.8	6.4
7.7	0.8	7.3	0.6
1.5	0.0	0.3	-0.7

Negative values in map `DX` mean that the slope is downwards when you go from west to east. A positive value indicates an upwards slope. In map `DY` positive values indicate that when you go from the south to north the slope is upwards and negative values mean that the slope is downwards.

SlopePct

50.9	56.5	49.6	51.2
63.8	36.9	69.7	56.9
55.5	4.5	61.9	85.6
9.3	3.0	51.5	88.6

SlopeDeg

27.0	29.5	26.3	27.1
32.6	20.3	34.9	29.6
29.0	2.6	31.8	40.6
5.3	1.7	27.3	41.5

The slope maps show the steepness of the slopes irrespective of the direction of the slope. Mind that the following slope values are the same: 30°= 58%, 45°= 100%, 60°= 173%, 80°= 567%. In this example it was not the case but slope values in the `SlopePct` map might rise above 100%.

For more information, refer to section ILWIS objects : Scripts, or Appendix F ILWIS script language (syntax).