6.3.4 Special Table Calculations

6.3.4.1 Calculating with undefined values

In every data set (map or table) you might encounter missing values or illegal outcomes of operations (e.g. a division by zero or the square root of a negative value). These are called undefined and are represented by a question mark (`?`). The outcome of calculations using an undefined value will always result in another undefined value with only one exception. This and the testing for, and assigning of undefined values is described below.

ISUNDEF(*a*)	tests whether *a* is undefined; returns a Bool
IFUNDEF(*a*,*b*)	returns *b* when *a* is undefined, else *a* is returned
IFUNDEF(*a*,*b*,*c*)	returns *b* when *a* is undefined, else c is returned
IFNOTUNDEF(*a*,*b*)	returns *b* when *a* is not undefined, else the undefined remains
IFNOTUNDEF(*a*,*b*,*c*)	returns *b* when *a* is not undefined, else the undefined is replaced by *c*

The ISUNDEF(*a*) function tests whether *a* (expression or column name) is undefined or not known. Function ISUNDEF may be used on columns with a domain Value, Class, ID, Group, or String. Below, the ISUNDEF function is combined with a conditional IFF.

Argument *a* can be the outcome of an expression, a column name or a record in a column.

☞ Do not use the expression: `Column1 = ?` because then the undefined value is assigned to every record in that column and further calculations are of no use.

Examples of testing for undefined values using the ISUNDEF function

```
Result1 = ISUNDEF(Value)

Result2 = IFF(ISUNDEF(Value), 1000, Value)
Result2 = IFUNDEF(Value, 1000, Value)
Result2 = IFUNDEF(Value, 1000)

Result3 = IFF(ISUNDEF(Value), MAX(Value), 500)
Result3 = IFUNDEF(Value, MAX(Value), 500)
```

Sitenr	Value	Result1	Result2	Result3
1	850	False	850	500
2	?	True	1000	1250
3	600	False	600	500
4	950	False	950	500
5	1250	False	1250	500
6	450	False	450	500
7	?	True	1000	1250
8	900	False	900	500

`Result1` is assigned True if column Value is undefined, else False.

Column `Result2` is assigned a value 1000 when a record of column `Value` is undefined; else the original value is assigned.

If column `Value` is undefined, then the maximum value from the rest of the records is assigned in column `Result3`, else `Result3` is assigned the value 500.

```
Result4 = IFF(ISUNDEF(Commval), 0, MAX(Commval))
Result4 = IFUNDEF(Commval, 0, MAX(Commval))

Result5 = IFF(ISUNDEF(landuse), ”unknown”, landuse)
Result5 = IFUNDEF(landuse, ”unknown”, landuse)
Result5 = IFUNDEF(landuse, ”unknown”)

Result6 = IFUNDEF(Commval,1000)
```

Sitenr	Commval	Landuse	Result4	Result5	Result6
1	4500	crops	9000	crops	4500
2	3300	grazing	9000	grazing	3300
3	9000	urban area	9000	urban area	9000
4	?	?	0	unknown	1000
5	5000	crops	9000	crops	5000
6	?	?	0	unknown	1000
7	3800	grazing	9000	grazing	3800
8	8000	urban area	9000	urban area	8000

Every site has a certain commercial value and a certain land use. The third formula tests whether there is an undefined in the column `Commval` (domain Value). If so, the undefined is replaced by the value 0, otherwise column `Result4` is assigned the maximum of the original commercial value.

The fifth formula tests whether there is an undefined in the column `Landuse` (domain String). If so, the undefined is replaced by the description ”unknown” else the original description is maintained.

`Result6` is assigned 1000 when column `Commval` is undefined. The original value is maintained when column `Commval` has a value. This works as well for columns having a non-value domain. Then, the second expression between the brackets needs quotation marks (” ”).

Assigning undefined values

When a value is out of range you may want to assign an undefined to these records. That may be done as follows:

```
Result1 = IFF(INRANGE(Value, 250, 1500), Value, ?)
```

```
Result2 = SQRT(Value)
```

Sitenr	Value	Result1	Result2
1	850	850	29.15
2	10000	?	100.00
3	600	600	24.49
4	950	950	30.82
5	1050	1050	32.40
6	-450	?	?
7	1	?	1.00
8	900	900	30.00

Every value in column `Value` which is not in the range 250 to 1500 is replaced by an undefined (`?`).

```
Result3 = IFF((Landuse=”unknown”),”?”,Landuse)
```

Landuse	Result3
crops	crops
grazing	grazing
urban area	urban area
unknown	?
crops	crops
unknown	?
grazing	grazing
urban area	urban area

The column `Result3` is assigned a ? if the column `Landuse` shows ”`unknown`”. To assign undefined values, use a question mark `?` when the output domain is a domain Value, and use ”`?`” when the output domain is a domain Class or ID. Besides values also Class, ID or Strings may be used in the expression.

6.3.4.2 Predefined variables and record specific operations

ILWIS enables the user to make calculations on specific records or items in a domain or use map lines and columns in an expression. These predefined variables are listed below.

%R	variable to calculate with record numbers in a table according to the order in which they appear in the domain
%K	variable to calculate with class names IDs or Groups of the domain of a table
%X	variable to calculate with X-coordinates in a raster map
%Y	variable to calculate with Y-coordinates in a raster map

☞ X- and Y-coordinates also are considered as predefined variable but they will be described comprehensively in the paragraph: Calculations on coordinates in the section Special Topics.

%R and %K are so called record specific and require a special syntax.

Syntax of record specific operations

Column[expr]	to use values of a column indexed by record numbers, classes or IDs
Table.Column[expr]	to use values of a column in another table, indexed by record numbers, classes or IDs
Table.Ext.Column[expr]	to use values of a column in a table with another extension than .TBT (e.g. .HIS), indexed by record numbers, classes or IDs.

Predefined variable %R

Normally TabCalc expressions are applied on all records in a table. However, if you wish to apply an expression on specific records, you can indicate so by using [%R] as an index behind a column name in an expression (R for record). Also, in between the square brackets, you can write an expression.

Examples of Predefined variable %R

```
Nr    = %R
Col2  = Col1[6-%R]
Diff  = Col1[%R+1] - Col1[%R]
Avg = (Col1[%R-1] + Col1 + Col1[%R+1]) / 3
```

Domain	Nr	Col1	Col2	Diff	Avg
a	1	10	56	15	?
b	2	25	48	12	24.0
c	3	37	37	11	36.7
d	4	48	25	8	48.0
e	5	56	10	?	?

Column `Nr` gives the record numbers according the order in which the items appear in the domain of the table. When using the record numbers of the domain you can simply type %R. When you want to use records from a certain column you have to type the name of the column followed by %R between square brackets [%R].

`Col2` shows the values of `Col1` in the opposite order.

Column `Diff` contains the difference of each record and its successive record. For the first record: 25-10 = 15

Column `Avg3` contains the average value of each three successive records. For the second record: (10+25+37) / 3 = 24.0

Predefined variable %K

In a table of domain type Class, Group or ID, you may use [%K] behind a column name in an expression to make calculations on class names, groups or IDs in a domain (K for key). Replace the %K with a class name or ID (between double quotes), or a class or ID column name to make calculations on specific items in the domain.

Example of predefined variable %K

```
Arvac=iff((%K=”vacant”), landuse.his.area, 1)
```

Histogram table

Domain Landuse	Area
Vacant	50000
Residential	23972267
Industrial	865923
Institutional	9651

Attribute table

Domain Landuse	ArVac
Vacant	50000
Residential	1
Industrial	1
Institutional	1

Only when the land use is ”`vacant`” according to the domain of the table the area Column `Area` is retrieved from `Landuse`.his. Histograms of a raster map (.HIS) and histograms of a polygon map (.HSA) always contain a column `Area`. The link between the current table and `Landuse.his` is made through the domain of the table: `%K.`

```
LandArea = Landuse.his.area[%k]
```

```
LandArea = Landuse.his.area
```

```
ArVac = LandArea + LandArea[”vacant”]
```

Domain Landuse	LandArea	ArVac
Residential	250000	300000
Industrial	50000	100000
Institutional	25000	75000
Vacant	50000	100000

Column `Area` is retrieved from `Landuse.his` (a histogram of raster map `Landuse`, histograms always contain a column `Area`); the link between the current table and `Landuse.his` is made through the domain of the table %K (domain Landuse) which is also the domain of the histogram. As the domain of the histogram and the table are the same, the first and the second statement give the same result.

Column `ArVac` contains for each record the value of column `Area` of each record, plus the area of class ”vacant”. This represents a situation when the vacant area would be used by each other class individually.

In the next example %K will be used in an IFF statement

```
Result =  IFF((%K<>”Residential”), LandArea / Aggsum
          (Area), 0)
```

Domain Landuse	LandArea	ArVac	Result
Residential	250000	300000	0
Industrial	50000	100000	0.09
Institutional	25000	75000	0.04
Vacant	50000	100000	0.09

If the item in the domain is not equal to "`Residential`" then calculate the relative proportion of the area with a certain land use, else (when the land use is "`Residential`" assign a 0.
The relative proportion of the area having a certain land use was calculated by dividing the Area (in square meters) by the total area. Function `Aggsum` calculates the sum of all areas.

6.3.4.3 Conversions

Sometimes it is desirable to convert Values to Strings or vice versa. In ILWIS two conversion functions are available:

Conversion functions

STRING(*a*)	returns value *a* as a string
VALUE(*a*)	returns string *a* as a value

Example of a conversion function

```
ResString = IFF(ISUNDEF(Value),"undefined",string(Value))
ResValue  =  VALUE(ResString)
```

Value	Resstring	ResValue
872	872	872
43	43	43
?	undefined	?
9285	9285	9285

Column `String` contains a number written as a string using a domain String. The records in this column have to be treated as text which means they require " ".
Column `Resvalue` contains the converted values using a domain Value.
In column `Resstring` these values are converted back into a string again.

6.3.4.4 Classify columns

Classified data are less detailed then the original ones. But they often easier to read because of the limited number of groups or classes.
When you want to classify a map, you have to predefine a Domain Group in which you indicate the upper limit for each class.

CLFY(*a*,*domGr*)	to classify the values of *a* according to a *domain Group*.

ILWIS enables you to classify maps and table columns (with domain Value) according to a predefined domain Group. A domain Group lists the upper boundaries of the groups and the group names.

A classification formula generally looks like:

`OUTCOL = CLFY(`InputColName, DomainGroup`)`

where:

OUTCOL	is the name of your output column
CLFY	is the function to classify values according to a domain Group
InputColName	is the name of your input column (domain Value column)
DomainGroup	is the name of the domain Group; this lists the boundaries of the values and the group names for the output column.

Example CLFY function
In this example soil depth values are classified according to the USDA Soil Classification System.

- The Group domain `SDepClas` is defined as follows:

Upper Boundary	Group Name
25	Very shallow
50	Shallow
100	Moderately deep
120	Deep
400	Very deep

- The classification is executed by typing the following formula on command line of your table window:

```
SoilDepC = CLFY(SoilDep,SDepClas)
```

- In the table, column `SoilDepC` is created and filled according to the Group domain.

Sample	SoilDep	SoilDepC
1	15	Very shallow
2	45	Shallow
3	95	Moderately deep
4	20	Very shallow
5	40	Shallow
6	60	Moderately deep
7	110	Deep

6.3.4.5 Aggregating values

Aggregation means that you get one aggregate value, for instance the average or the sum, of a whole column. You may also perform the aggregation per group of classes and/or assign weight values.

Aggregations	▪ aggregating values of all records of column *a*, ▪ aggregating values in column *a* for a group of records of the same class (key column *g*) , or ▪ aggregating values in column *a* for a group of records of the same class (key column *g*) while using weight factors in a weight column *w*.

To perform an aggregation through a dialog box, display the table containing data to be aggregated in a table window, open the Columns menu and choose Aggregation.

The Aggregation dialog box appears in which you specify:

- the column that you want to aggregate,
- the aggregation 'function' that you want to use. Several 'functions' are available to aggregate the values of a column,
- specify:
  - whether you want to aggregate (group by) a *key column*; if yes, select the key column name,
  - whether you want to use weights in a *weight column*; if yes, select the weight column name,
  - whether you want the aggregation results in a separate table; if yes, type a table name,
- the output column name of the aggregation.

Aggregations can also be performed by typing a formula on the command line of a table window.

Aggregation functions

AGGAVG(*col*)	calculates the average value of column *col*
AGGAVG(*col*,*g*)	calculates the average value of *col* per group *g*
AGGAVG(*col*,*g*,*w*)	calculates the average value of *col* per group *g* using weights *w*
AGGCNT(*col*)	counts the number of times that expression *col* is true
AGGCNT(*col*,*g*)	counts the number of times that expression *col* is true per group *g*
AGGMIN(*col*)	determines the minimum value of *col*
AGGMIN(*col*,*g*)	determines the minimum value of *col* per group *g*
AGGMED(*col*)	calculates the median value of *col*
AGGMED(*col*,*g*)	calculates the median value of *col* per group *g*
AGGMED(*col*,*g*,*w*)	calculates the median value of col per group *g* using weights *w*
AGGMAX(*col*)	determines the maximum value of *col*
AGGMAX(*col*,*g*)	determines the maximum value of *col* per group *g*
AGGPRD(*col*)	determines the predominant value of *col*
AGGPRD(*col*,*g*)	determines the predominant value of *col* per group *g*
AGGPRD(*col*,*g*,*w*)	determines the predominant value of *col* per group *g* using weights *w*
AGGSTD(*col*)	calculates the standard deviation of *col*
AGGSTD(*col*,*g*)	calculates the standard deviation of *col* per group *g*
AGGSTD(*col*,*g*,*w*)	calculates the standard deviation of *col* per group *g* using weights *w*
AGGSUM(*col*)	calculates the sum of *col*
AGGSUM(*col*,*g*)	calculates the sum of *col* per group *g*

☞ When no weight column is specified, then all data get the same weight value which is per default 1.

☞ When you do not want to group the data but you do want to use a weight column then only omit the column name to group but leave the commas in the syntax. For example: AGGAVG(Col, ,w). Otherwise ILWIS can not make the distinction between the column used to group and the column containing the weights.

Examples of aggregation functions on the TabCalc command line

In the following example the average value of a parcel, the average value of a parcel per land use class and the weighted average value per land use class are calculated.

```
Avg1 = AGGAVG(Value)
Avg2 = AGGAVG(Value,Landuse)
Avg3 = AGGAVG(Value,Landuse,Weight)
```

Parcel	Landuse	Value	Weight	Avg1	Avg2	Avg3
100	Residential	4000	3	10230	5100	5200
124	Residential	3500	2	10230	5100	5200
157	Commercial	17500	1	10230	11750	9833
162	Residential	7500	3	10230	5100	5200
181	Industrial	20000	1	10230	20000	20000
202	Institutional	12500	1	10230	16650	18033
225	Residential	5000	4	10230	5100	5200
269	Commercial	6000	2	10230	11750	9833
288	Institutional	20800	2	10230	16650	18033
295	Residential	5500	3	10230	5100	5200

Column `Avg1` shows the average value of all parcels. It does not matter which type of land use is practiced on a parcel.

Column `Avg2` shows the average value of the parcels grouped by the type of land use. The weight factor is not taken into account.

Column `Avg3` shows the weighted average value of the parcels per land use class.

You can also retrieve and aggregate values from another table. This requires specification of the table and the columns to be used; this may look like:

```
Population = AGGSUM(Municip.pop,municip.prov)
```

Table `Municip` has a column `Pop` (showing the number of inhabitants of a municipality) and a column `Prov` (indicating the province in which the municipality is found). A new (or existing) column `Population` in the table that you are working on contains the sum of the population in all municipalities per province.

To write output values into another existing table, in a new column, use the following syntax:

```
Table2.Avg = AGGAVG(Table1.Area,Table1.Landuse)
```

In other words:
Column `Avg` in Table2 contains the average area size per land use type indicated in `Table1`.

When performing aggregations from the command line of a table window, you must know that argument *col* as used above, refers to nothing else than a column name.
For the complete list of Aggregation syntaxes refer to the Appendices.

☞ Within brackets, no expressions can be used.
☞ For ILWIS 1.41 users: argument *g* can be considered as the key column.

6.3.4.6 Joining columns

The join operation enables you to read a column from a second table and join it into the current table. To do so, you need a link between the tables. This link is made via the domain of the tables or columns in the tables. There are four possibilities:

1. Both tables use the same domain.
2. A column in the current table uses the same domain as the second table.
3. The current table uses the same domain as a column in the second table.
4. A column in the current table uses the same domain as a column in the second table.

Example Join
Column `Area` in the histogram table of polygon map `Landuse` will be joined into the attribute table `Landuse`. Display attribute table `Landuse` in a table window. Choose Join from the Columns menu. Then specify from which table you want to join a column and select that column.

Landuse.tbt

Domain Landuse	Commval
Residential	1000
Commercial	2000
Industrial	5000
Institutional	4000
Agricultural	2000

Landuse.hsa

Domain Landuse	Area
Residential	9920800
Commercial	4131200
Industrial	2161600
Institutional	4918400
Agricultural	10461600

Landuse.tbl after joining

Domain Landuse	Commval	Area
Residential	1000	9920800
Commercial	2000	4131200
Industrial	5000	2161600
Institutional	4000	4918400
Agricultural	2000	10461600

The same result is obtained when you type the following TabCalc formula on the command line of the table window:

```
Area = ColumnJoin(Landuse.hsa.Area)
```

The joining of column `Area` in the histogram table `Landuse.hsa` is done via the domain `Landuse` which was the domain of both tables.

Join 2
You can also use the domain of a key-column in the current table to make the link to the second table. Choose Join from the Columns menu. Then specify from which table you want to join a column, and select that column. You also have to specify the key column in the current table.

Join 3
When the domain of your current table is the same as the domain of a column in your second table you make the link as follows:
Select Join from the Columns menu in the table window. Then select the second table used for the join operation and the column that you want to join into your current table.
DO NOT select a key column in your current table. In this way you use the domain of the current table and not the domain of one of the columns in your current table. In your second table you select a key column according which the data are grouped. When the column in the second table contains values you may also want to perform aggregation operations. Finally the output column name is by default the same name as the name of the column that you want to join into the current table.

Join 4
The last possibility is when you join a column into a the table via the corresponding domains of columns in both of your tables. Select Join from the Columns menu in the table window. Select the table name which you want to use and the column which you want to join into your current table. Then, select a key column in your current table and a column in the second table to group your data. Note that these two columns must have the same domain. You may choose to perform aggregation operations but this can only be performed on values. The join will take place as was described before. When an item of your key columns is linked to several possibilities in the column you want to join into your current table, then an undefined is the result.

6.3.5 Special Topics

6.3.5.1 Calculations on coordinates

In ILWIS coordinates may be used in calculations. Coordinates can be read into tables as separate X- and Y-components. You may retrieve coordinates from raster, polygon, segment and point maps. Special functions were designed for distance calculations, transformations to and from other coordinate systems and for calculations with point data and data properties.

Functions available for the retrieval of coordinates

MAPCRD (*rasmap*)	returns the coordinates of raster map *rasmap.mpr* on the current row and column (MapCalc only)
MAPCRD (*rasmap, rowexpr, colexpr*)	returns the coordinates of raster map *rasmap.mpr* on row number rowexpr and column number colexpr
PNTCRD (*pntmap,pntnr*)	returns the coordinates of the point number in point map *pntmap.mpp*
PNTCRD (*pntmap, "pnamexpr"*)	returns the coordinates of a point with the name "pnamexpr" in point map *pnt.mpp*
CRDX (*coordexpr*)	returns the X-coordinate of a coordinate expression
CRDY (*coordexpr*)	returns the Y-coordinate of a coordinate expression
COORD (*value, value, coordsys*)	returns the X- and Y-coordinate according two input values and coordinate system *coordsys.csy*
MINCRDX (*basemap.ext*)	returns the minimum X-coordinate in a raster, segment, polygon or point map *basemap.ext*
MAXCRDX (*basemap.ext*)	returns the maximum X-coordinate in a raster, segment, polygon or point map *basemap.ext*
MINCRDY (*basemap.ext*)	returns the minimum Y-coordinate in a raster, segment, polygon or point map *basemap.ext*
MAXCRDY (*basemap.ext*)	returns the maximum Y-coordinate in a raster, segment, polygon or point map *basemap.ext*

A *rowexpr*, *colexpr* or *coordexpr* is an expression resulting in a row number, column number or a coordinate respectively.

☞ These functions all result in coordinate values. They can be used as input statements in calculations.
☞ MAPCRD (*rasmap*) is defined as MAPCRD(rasmap,%L, %C)
☞ MAPCRD (*rasmap*) is used as input in an expression. It is not possible to produce an output map with a domain containing coordinates.

Other functions on coordinates:

TRANSFORM (*coordexpr*, *coordsys*)	transforms the coordexpr to another coordinate system *coordsys.scy*
DIST (*coordexpr*, *coordexpr*)	returns the distance between two coordinates (the use of different coordinate systems is allowed)
DIST2 (*coordexpr*, *coordexpr*)	returns the square of the distance between two coordinates (the use of different coordinate systems is allowed)
MAPVALUE (*basemap.ext*, *coordexpr*)	returns the value of a location described by coordexpr in a raster, segment, polygon or point map *basemap.ext*
MAPROW (*rasmap*, *coordexpr*)	returns a row number in raster map *rasmap* on coordinate coordexpr
MAPCOL (*rasmap*, *coordexpr*)	returns a line number in raster map *rasmap* on coordinate coordexpr
RASVALUE (*rasmap*, *rowexpr*, *colexpr*)	returns the value of a raster map on row *rowexpr* and column *colexpr*

A *rowexpr*, *colexpr* or *coordexpr* is an expression resulting in a row number, column number or a coordinate respectively.

Examples

You may use the pocket line calculator if you want to have a quick look at pixel values of other maps on a specified location (Refer for more information on the pocket line calculator to section 6.5). This location may be specified as line and column number or as coordinates.

```
? MAPVALUE (tmb1,coord (800000,8080000),cochabam)
```

This expression returns the value of a raster, polygon, segment or point map on the location with the X-coordinate 800000 and Y-coordinate 8080000.

```
? RASVALUE (tmb1, 100, 200)
```

This expression returns the pixel value of the pixel on line 100 and column 200.

```
? CRDX(PNTCRD(Rainfall,”PROMIC”))
```

This expression returns the X-coordinate of the rainfall station named PROMIC in the point map `Rainfall`

Example of the DIST function

The function DIST and the point map `Rainfall` (location of rainfall stations in the area of Cochabamba, Bolivia) are used in this example. The distance is calculated between the rainfall station named: PROMIC and all the other stations.

```
Distance =  DIST (COORD(x,y), PNTCRD(rainfall,”PROMIC”))
```

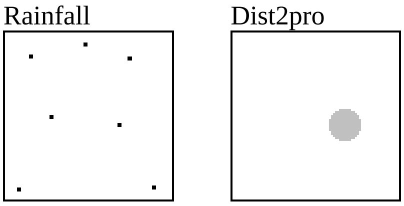
	X	Y	Name	Distance
1	803429	8074631	UMSS	5908.62
2	799748	8082415	Taquina	3455.55
3	803511	8087422	Laguna Santa Rosa	6996.87
4	801281	8088967	Laguna Totura	8587.93
5	802613	8080483	PROMIC	0.00
6	806816	8076585	Aro Cagua	5732.33
7	797465	8078352	Colca Pithua	5571.63
8	796346	8087736	Cerro MachaMach	9585.47

Example of the DIST2 function

With function DIST2 you can calculate a map showing an area within a certain distance around a point. When you want to calculate the area within a distance of 2000 m. from rainfall station PROMIC you can type in the command line of the Main window:

```
dist2pro =  DIST2(PNTCRD(Rainfall, ”PROMIC”), MAPCRD
            (Rainfall)) < SQ(2000)
```

A Raster Map Definition box pops up showing the expression and domain type `Bool`. Accept the defaults. Then force calculation of the map by double-clicking the newly created raster map `Dist2pro` in the Catalog.



Map `Rainfall.mpp` is a point map showing the location of several rainfall stations. The one just below and a bit right of the center of the map is the rainfall station PROMIC. Map `Dist2pro` shows the area (with a radius of 2000 m) around
this station.

☞ This is a an expensive operation in the sense that ILWIS has to calculate for every pixel the distance to the location of rainfall station PROMIC. This costs much time.

Assigning pixel values to a back drop line

This example describes the assignment of average values to pixels of an image in which whole lines are missing. Occasionally satellite images suffer from so-called back drop lines. In this example every 8th line is missing. To these lines, values which are the average values from the line above and the line below will be assigned.

```
Result =  IFF((%L MOD 8=0), (RASVALUE (Image1, %L-1, %C)
          + RASVALUE (Image1, %L+1, %C))/2, Image1)
```

Image1

25	39	45	12
51	6	29	63
?	?	?	?
8	18	53	91

Result

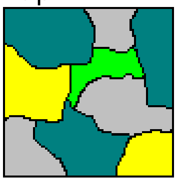
25	39	45	12
51	6	29	63
29	12	41	77
8	18	53	91

In `Image1` some pixels are shown which belong to line 6 up to 9 while line 8 is missing. These missing values are replaced by the average values (rounded down) of the pixels of the line above and the line below.

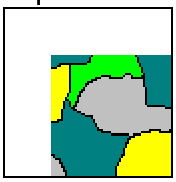
Example of predefined variables %X and %Y

```
Map2 = IFF((%X > 80000) AND (%Y < 8080000), Map1, "?")
```

`Map1`



`Map2`



`Map2` has the same content as `Map1` if the X-coordinate is larger than 800000 and the Y-coordinate is smaller than 8080000. Outside this area `Map2` is undefined.

Example of a transformation to another coordinate system

Coordinates from point data will be transformed into another coordinate system. The data originate from coordinate system `Cochabam` (Cochabamba, Bolivia) and will be transformed into longitudes and latitudes (`Latlon` coordinate system).

```
Newx = CRDX(TRANSFORM(COORD(x,y,Cochabam),Latlon))
Newy = CRDY(TRANSFORM(COORD(x,y,Cochabam),Latlon))
```

	X	Y	Name	Newx	Newy
1	799491	8086120	Number1	- 66.18	- 17.29
2	800085	8080962	Number2	- 66.18	- 17.34
3	797710	8077028	Number3	- 66.20	- 17.37
4	798934	8083040	Number4	- 66.19	- 17.32
5	803610	8078624	Number5	- 66.14	- 17.36
6	803313	8085712	Number6	- 66.15	- 17.29

In words the above mentioned expressions read:
Take the X-component of a coordinate belonging to coordinate system Cochabam and transform it to a longitude value. Take the Y-component of a coordinate belonging to coordinate system Cochabam and transform it to a latitude value.

Example of calculating with minimum and maximum values of coordinates

It might occur that you have a point data set from which only a part of all points lay inside a certain area. You can select only these points to show or to use them for further calculation. In the next example it will be tested whether the points of a point map lay within the area of an existing land use map.

The coordinates from the lower left corner of the land use map are: (795480, 8071880) and the coordinates from the upper right corner: (808220, 8090520).

```
Select = IFF (INRANGE (x,MINCRDX (landuse), MAXCRDX
         (landuse)) and (INRANGE (y, MINCRDY (landuse),
         MAXCDRY (landuse)))), "yes", "no")
```

	X	Y	Name	Select
1	799491	8086120	Number1	yes
2	800085	8080962	Number2	yes
3	783622	8077028	Number3	no
4	826749	8083040	Number4	no
5	803610	8078624	Number5	yes
6	803313	8085712	Number6	yes
7	798204	8123951	Number7	no

Points 3, 4 and 7 do not lie inside the boundaries of the land use map. The X-coordinate of point Number3 is lower than the minimum X-coordinate of the landuse map while point Number4 has an X-coordinate which is too high. And point Number7 has a Y-coordinate which is higher than the maximum Y-coordinate in the landuse map.

6.3.5.2 Calculations on point data

As you may have seen above, many of the functions for calculations on coordinates will be applied on point maps. Five special functions are described here especially for calculations on point data. A point map can be opened as a map and as a table. Extra columns with attribute data may be added, joined or created in this table. They can be displayed in attribute maps.

Functions for calculations on point data

PNTNR (*pntmap*, *pnameexpr*)	returns the number of the point with name `pnamexpr` in point map *pntmap.mpp*
PNTVAL (*pntmap*, *pntnr*)	returns the value of a point with the specified number in point map *pntmap.mpp*
PNTVAL (*pntmap*, *pnamexpr*)	returns the value of a point with the specified name `pnamexpr` in point map *pntmap.mpp*
PNTCRD (*pntmap*,*pntnr*)	returns the coordinates of the pntnr in point map *pnt.mpp*
PNTCRD (*pntmap*, *pnamexpr*)	returns the coordinates of a point with the name pnamexpr in point map *pnt.mpp*

Examples of calculations on point data

In this example the `Rainfall` point map is used which stores information about rainfall stations located in the area of Cochabamba (Bolivia). The pocket line calculator can be used to retrieve the point number of a certain rainfall station:

```
? PNTNR (rainfall, "Laguna Santa Rosa")
```

This will return 3 when Laguna Santa Rosa is the third rainfall station.

Example of calculating the distance between two points
The distance between a pixel at the left boundary and a pixel at the right boundary will be calculated of map `Geol`. Function MAPCOLS (see calculating with data properties) returns the number of columns in a map. Therefore, it is unnecessary to put the map on the screen to find out which column is the last one. Since the map has a rectangular shape you can use any row number you like. The first one will be used to perform the calculation in the pocket line calculator.

```
? DIST(MAPCRD(Geol,1,1),MAPCRD(Geol,1,MAPCOLS(Geol)))
```

This calculation returns 12720.00 m. as the distance between the outer two pixels at the left and the right side of the map.

Example of calculating a cross-section
Many researchers are interested in making a cross-section through (a part of) their study area.
This example shows how this might be done using ILWIS. Firstly you have to choose the location of the cross-section. Then you have to make a raster map showing only the pixels displaying the cross-section. This point map is used as a table to add all kinds of attribute data.

Start by having a look at the satellite images, geology and landuse map from the study area Cochabamba in Bolivia. Column 165 seems to cover many different terrain features and is chosen as the cross-section.

To make a segment map showing the cross-section, use the formula:

```
crosssec = IFF(%C=165, geol,?)
```

A raster map `crosssec.mpr` is created showing the geological classes for column 165. All other pixels are undefined. This map is used to create a point map. To do so, drag and drop the newly created raster map `crosssec.mpr` from the Catalog to the RasPnt operation in the Operation-list.

A dialog box appears in which you need to fill in the name of the output point map. Use again the name `crosssec` (point maps have extension `.mpp`) and select the Show check box. Then the map is calculated; accept the default representation `crosssec.rpr` in the Display Options Dialog box and click OK. The point map appears on the screen.

Next, the point map should be opened as a table: click with the right mouse button the point map `crosssec` and choose Open as Table in the context-sensitive menu. The table shows all points, having an X- and a Y-coordinate. In the column `Name` you find the code for the type of `geology`. This column also uses the domain `geol`.

	X	Y	Name
21	798770.00	8090090.00	scc
22	798770.00	8090070.00	scc
23	798770.00	8090050.00	scc
24	798770.00	8090030.00	sun
25	798770.00	8090010.00	sun
26	798770.00	8089990.00	sun

You can add a new column to the table with information on landuse. The map `Landuse.mpr` will be used. Type the following expression on the command line of the table window:

```
Landuse = MAPVALUE (landuse, COORD (x, y))
```

The value of map `Landuse.mpr` on the coordinate defined by the X and Y in the table will be retrieved and stored in the new column `Landuse`.
Accept the default for the domain in the Column Properties dialog box and click OK. Now the new column is calculated and appears in the table.

	X	Y	Name	Landuse
629	798770.00	8077930.00	qfl	Crops with irrigation
630	798770.00	8077910.00	qfl	Crops with irrigation
631	798770.00	8077890.00	qfl	Crops with irrigation
632	798770.00	8077870.00	qfl	Crops with irrigation
633	798770.00	8077850.00	qfl	Urban area
634	798770.00	8077830.00	qfl	Urban area

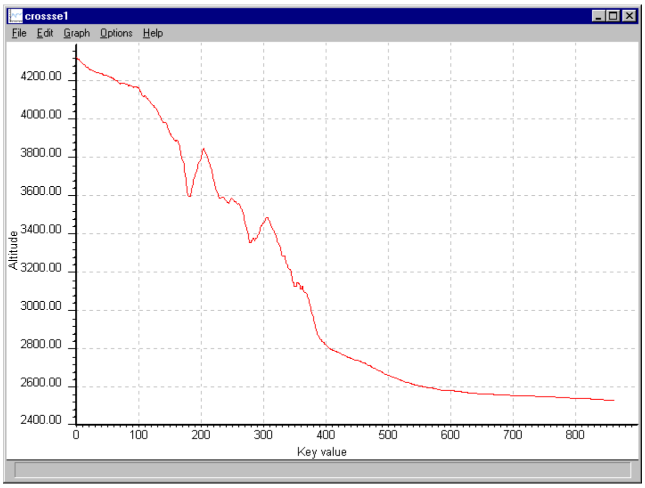
Next, add the column `Altitude` for information on the altitude along the cross-section. Altitude information for a certain locations is retrieved from the digital elevation model `dem.mpr` which is available in the data set. Type the following expression in the command line of the table window:

```
Altitude = MAPVALUE (dem.mpr, PNTCRD (COORD (X, Y)))
```

This means that the value of the digital elevation model `dem` is read into the column `Altitude`. The coordinates used belong to the point with X- and Y-coordinate defined in column X and Y of the current table. Accept the defaults in the Column Properties dialog box and press OK. Altitudes are added to the table.

	X	Y	Name	Landuse	Altitude
21	798770.00	8090090.00	scc	Low grass	4260
22	798770.00	8090070.00	scc	Low grass	4258
23	798770.00	8090050.00	scc	Low grass	4255
24	798770.00	8090030.00	sun	Low grass	4253
25	798770.00	8090010.00	sun	Low grass	4253
26	798770.00	8089990.00	sun	Low grass	4251

Now you can have a look to the relief along the cross-section. You can display in a graph the altitude depending on the geographic position in the cross-section. To do so, open the Options menu in the table window and select Show Graph.
For X-axis (the independent variable) select: key value. This is the order in which the points appear in the domain: key value1 is the start of the cross-section and the last value is the end of the cross-section. For the Y-axis (dependent variable) select: altitude and then click OK. Accept the defaults in the Edit Graph dialog box and the graph is displayed on the screen.



The graph shows the relief along the cross-section. The first point lies high in the mountains on an altitude of around 4300 m while the end of the cross-section is in a relatively flat area around 2500 m altitude.

6.3.5.3 Calculations on data properties

The following functions are available for retrieving information on some data properties:

`MAPMIN(`*basemap*`)`	returns the minimum value of raster, polygon, segment or point map *basemap*
`MAPMAX(`*basemap*`)`	returns the maximum value of raster, polygon, segment or point map *basemap*
`PIXSIZE(`*rasmap*`)`	returns the pixel size of raster map `rasmap.mpr`
`PIXAREA(`*rasmap*`)`	returns the pixel area (square of the pixel size) of raster map *rasmap.mpr*
`PIXSIZE(`*georef*`)`	returns the pixel size of the georeference *georef.grf*
`PIXAREA(`*georef*`)`	returns the pixel area (square of the pixel size) of georeference *georef.grf*
`MAPROWS(`*rasmap*`)`	returns the number of rows of raster map *rasmap.mpr*
`MAPCOLS(`*rasmap*`)`	returns the number of columns of raster map *rasmap.mpr*
`TBLRECS(`*table*`)`	returns the number of records in table *table.tbt*
`TBLCOLS(`*table*`)`	returns the number of columns in table *table.tbt*

Examples of calculating with data properties

```
Col2 = Col1 + MAPMIN (Map1)
Col3 = PIXSIZE (Map1)
Col4 = Col2 * PIXAREA (Map1)
Col5 = Col4 / TBLRECS (Point.mpp)
```

	X	Y	Col1	Col2	Col3	Col4	Col5
1	799491	8086120	25	65	20	26000	260
2	800085	8080962	10	50	20	20000	200
3	783622	8077028	20	60	20	24000	240
4	826749	8083040	30	70	20	28000	280

The formula: `Col2 = Col1 + MAPMIN (Map1)`adds the minimum value appearing in `Map1` (in this example 40) to the value in `Col1`.

The formula: `Col3 = PIXSIZE (Map1)` returns the pixel size of `Map1`.

The formula: `Col4 = Col2 * PIXAREA (Map1)` multiplies the value in `Col2` with the pixel area (square of pixel size) of `Map1` (20 * 20 = 400).

The formula: `Col5 = Col4 / TBLRECS (Point.mpp)` divides `Col4` by the total number of records in the table of point map `Point.mpp` (in this case 100)

When you want to display only the highest values of a map you may use the following formula:

```
Best = map1 > 0.75 * MAPMAX(Map1)
```

Map1

10	35	20	25
35	15	40	15
10	40	30	5
20	5	20	10

Best

F	T	F	F
T	F	T	F
F	T	F	F
F	F	F	F

The map `Best` has domain `Bool` and displays True only for the values between 75 and 100% of the maximum value of `Map1`. The rest of the pixels are assigned False. In this example the maximum of `Map1` is 40 and only for values higher than 30 the statement is true.

6.3.5.4 Calculations on colors

Besides manipulation of colors in the representation of maps, ILWIS also has the possibility to retrieve and/or assign colors in calculations. Calculations on colors require maps or table columns with a color or a picture domain, or representations.

Every color may be seen as a combination of different amounts of the primary colors: red, green and blue. In theory there is an infinite number of colors possible. ILWIS uses for contributions of each of the colors red, green and blue the range from 0 to 255.

For example, the notation:

(0, 0, 0)	returns black,
(255, 0, 0)	returns red,
(0, 255, 0)	returns green,
(0, 0, 255)	returns blue
(255, 255, 0)	returns yellow
(255, 0, 255)	returns magenta
(0, 255, 255)	returns cyan
(255, 255, 255)	returns white.

Another way of composing a color is assigning values to Hue (direction of the color in a three-dimensional color cube), the Saturation (purity of the color), and the intensity (brightness of the color).
To these variables values can be assigned ranging from 0 to 240 (see also Color separation).

Retrieving colors

The following functions return one of the color separations:

`CLRRED(`*colorexpr*`)`	returns the red component
`CLRGREEN(` *colorexpr*`)`	returns the green component
`CLRBLUE(` *colorexpr*`)`	returns the blue component
`CLRYELLOW(` *colorexpr*`)`	returns the yellow component
`CLRMAGENTA(` *colorexpr*`)`	returns the magenta component
`CLRCYAN(` *colorexpr*`)`	returns the cyan component
`CLRGREY(` *colorexpr*`)`	returns the gray component
`CLRHUE(` *colorexpr*`)`	returns the hue component
`CLRSAT(` *colorexpr*`)`	returns the saturation component
`CLRINTENS(` *colorexpr*`)`	returns the intensity component

Example of a color separation function
The contribution of the primary color Green to the colors in `Map1` will be analyzed.

Double-click ColorSep in the Operation-list or drag and drop `Map1` to this item. Then, select the input map `Map1` (only maps with picture or color domain are available for this operation) `Green` and fill in `Sepgreen` as Output Raster Map.

To execute the Color Separation operation it is also possible to type the following formula on the command line of the Main window:

```
Sepgreen = MapColorSep(Map1,Green)
```

```
Sepgreen = Map1.Green
```

Map1

0 ,255, 0	100, 50, 0
255,255,255	0 ,100,150
200, 0 ,100	50, 50 , 50
100,100,255	255,200, 0

Sepgreen

255	50
255	100
0	50
100	200

Pixels in `Map1` have colors build up by red, green and blue. The contribution of green is shown after color separation in the output map `Sepgreen`.

Functions for retrieving colors

MAPCOLOR (*MapName, rowexpr, colexpr*)	returns the color of the pixel in raster map *MapName* on row number *rowexpr* and column number *colexpr* using the default representation
MAPCOLOR (*MapName*)	returns the colors of the corresponding pixels in raster map *MapName*, using the default representation. (available in MapCalc only)
RPRCOLOR (*Repr.rpr, value expr*)	returns the color given in representation table *MapName.rpr* or *Repr.rpr* for value *value expression*
RPRCOLOR (*Repr.rpr, "class expr"*)	returns the color of given in representation table *MapName.rpr* or *Repr.rpr* for class expression

Example of retrieving colors

To read the map colors from the map `Landuse` on the screen or to make a new map with the Color domain you may use the following expression:

```
Result = MAPCOLOR(Landuse)
```

Landuse

Coffee	Shrubs
Shrubs	Bare Soil
Coffee	Shrubs
Bare Soil	Coffee

Result

140, 0 , 0	255,255, 0
255,255, 0	0 ,150,100
140, 0 , 0	255,255, 0
0 ,150,100	140, 0 , 0

The class map `Landuse` is converted to map `Result` having the Color domain. The pixels which had land use `coffee` and which were brown in map `Landuse`, still have the color brown in the new map but show now the contribution 140,0 and 0 for red, green and blue respectively. Land use shrubs was represented by yellow in map Landuse which is composed of red: 255, green: 255 and blue: 0. Finally Bare Soil had the color green which had a contribution of 0, 150 and 100 of red, green and blue.

To retrieve quickly the color of a certain pixel you may type in the Pocket Line Calculator:

```
? MAPCOLOR (Map1, 100, 200)
```

This command returns the color of `Map1` of the pixel on row nr 100 and column nr 200 in values for red, green, blue.

To retrieve the color of a map value or class you may type in the Pocket Line Calculator:

```
? RPRCOLOR(suit.rpr, 4)
```

```
? RPRCOLOR(landuse,”coffee”)
```

The first statement returns the color for class `4` in a map with suitability classes. The second statement returns the color for land use `coffee` in map `Landuse`.

Using Table Calculation you have the ability to read colors from a representation into another table. This may be done by typing a formula on the command line of your table window.

In the next example, the colors of a map representing parcels with a certain land use will be retrieved into the attribute table. To find the color of every item in the domain, you can use `%k`.

```
Col = RPRCOLOR(Parcel.rpr, %k)
```

Parcel.tbt

	Landuse
parcel 1	Coffee
parcel 2	Shrubs
parcel 3	Shrubs
parcel 4	Bare Soil
parcel 5	Coffee
parcel 6	Shrubs
parcel 7	Bare Soil
parcel 8	Coffee

Parcel.rpr

	Color
parcel 1	140, 0, 0
parcel 2	0, 150, 100
parcel 3	0, 150, 100
parcel 4	255, 255, 0
parcel 5	140, 0, 0
parcel 6	0, 150, 100
parcel 7	255, 255, 0
parcel 8	140, 0, 0

Parcel.tbt

	Land use	Color
parcel 1	Coffee	140, 0, 0
parcel 2	Shrubs	0, 150, 100
parcel 3	Shrubs	0, 150, 100
parcel 4	Bare Soil	255, 255, 0
parcel 5	Coffee	140, 0, 0
parcel 6	Shrubs	0, 150, 100
parcel 7	Bare Soil	255, 255, 0
parcel 8	Coffee	140, 0, 0

This expression returns for every item in the `Landuse` attribute table the color which was described in the representation table.

Assigning colors

These functions enable the user to create colors

COLOR (*redexpr*, *greenexpr*, *blueexpr*)	returns a new color composed by expressions resulting in values for red, green and blue (ranging from 0 to 255)
COLORHSI (*hueexpr*, *satexpr*, *intensexpr*)	returns a new color composed by expressions resulting in values for hue, saturation and intensity (ranging from 0 to 240)

Example of assigning colors

```
Landcol = color(IFF(Landuse=”coffee”,255,0), IFF
          (Landuse=”coffee”, 0, 100), IFF(Landuse=”coffee”,
          0, 200))
```

Landuse

Coffee	Shrubs
Shrubs	Bare Soil
Coffee	Shrubs
Bare Soil	Coffee

Landcol

255, 0 , 0	0 ,100,200
0 ,100,200	0 ,100,200
255, 0 , 0	0 ,100,200
0 ,100,200	255, 0 , 0

The new map `Landcol` has a color domain. It will have value 255 for the red component when the landuse is coffee. For all other types of land use the red color component is 0. In the same way, the green component is 0 when the landuse is coffee and in all cases it has the value 100. Finally the blue component is 0 when the landuse is coffee. For all other types of landuse the blue component has value 200.

Advanced calculations with colors

This example shows a more advanced application of calculating with colors.
A representation will created for a new map, using two input maps and showing features of both original maps.
The first input map is a geology map with a class domain. The second map is a digital elevation model after application of a shadow filter. This second map was stretched using Histogram Equalization with 16 intervals. This resulted in a hill shading map in 16 gray tones. The input maps `Geol` and `Demshad` are crossed, creating a cross table `Geolshad.tbt` and a cross map `Geolshad.mpr`. The domain of the newly created table and map is an ID domain and needs to be converted to a class domain.

Now, new colors will be assigned according to the landuse map but showing the hill shading as well.

In representation table `Geolshad.rpr` a new column `classcol` needs to be created. This column contains for each combination of geology and gray tone the original colors of the corresponding geology in the geology map.

All combinations:
- `moraine deposits * gray tone 1` , up to
- `moraine deposits * gray tone 16`

will all have the same base color as the moraine deposits in the input geology map.

The following command is used on the command line of the representation table:

```
Classcol := Geol.rpr.Color[Geol.tbt.Landuse]
```

Then the column `Color` present in the representation table will be replaced by new colors which also indicate the proportion of the stretch class.

```
Color:=COLOR((CLRRED(classcol)*Geolshad.tbt.Demshad/16),
        (CLRGREEN(classcol)*Geolshad.tbt.Demshad/16),
        (CLRBLUE(classcol)*Geolshad.tbt.Demshad/16))
```

This means in words:
- For each of the color components red, green and blue, the color indicated in column `Classcol` is multiplied by the proportion of gray shading.
- `Class Moraine deposits * gray shade 1` will get color values multiplied by 1/16.
- `Class Moraine deposits * gray shade 16` will get color values divided by 16/16.