14. Functions and Operators

Introduction

Functions modify the value of one or more variables to produce a result (i.e., ROUND(2.33333) produces the value 2).

Operators are used to combine two items (i.e., the + operator combines Var1 and Var2 to produce a sum, as in Var3=Var1+Var2).

Functions and operators appear within commands and are used for common tasks that include extracting a year from a date, combining two numeric values, or testing logical conditions.

Almost all functions require arguments enclosed in parentheses and separated by commas. If arguments are required, do not place any spaces between the function name and the left parenthesis. Syntax rules must be followed. Quotes that must enclose text strings are displayed in question or prompt dialog boxes. Parentheses must enclose arithmetic expressions and can explicitly control the order of operations. Parentheses also enclose function arguments.

Syntax Notations

The following rules apply when reading this manual and using syntax:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL CAPITALS</td>
<td>Epi Info commands and reserved words are shown in all capital letters similar to the READ command.</td>
</tr>
<tr>
<td>&lt;parameter&gt;</td>
<td>A parameter is information to be supplied to the command. Parameters are enclosed with less-than and greater-than symbols or angle brackets &lt; &gt;. Each valid parameter is described following the statement of syntax for the command. Parameters are required</td>
</tr>
<tr>
<td>Syntax</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>by the command unless enclosed in braces { }. Do not include the &lt; &gt; symbols in the code.</td>
</tr>
<tr>
<td>[&lt;variable 1&gt;]</td>
<td>Brackets [ ] around a parameter indicates that there can potentially be more than one parameter.</td>
</tr>
<tr>
<td>{&lt;parameter&gt;}</td>
<td>Braces {} around a parameter indicate that the parameter is optional. Do not include the {} symbols in the code.</td>
</tr>
<tr>
<td></td>
<td>The pipe symbol ‘</td>
</tr>
<tr>
<td>/*</td>
<td>The combination of backslash and asterisk in the beginning of a line of code and an asterisk and backslash, as shown in some code samples, indicates a comment. Comments are skipped when a program is run.</td>
</tr>
<tr>
<td>*/</td>
<td>Quotation marks must surround all text values as in: DIALOG &quot;Notice: Date of birth is invalid.&quot;</td>
</tr>
</tbody>
</table>
Operators

There are various types of operators discussed in this appendix. The following types are provided:

- **Arithmetic Operators** are used to perform mathematical calculations.
- **Assignment Operators** are used to assign a value to a property or variable. Assignment Operators can be numeric, date, system, time, or text.
- **Comparison Operators** are used to perform comparisons.
- **Concatenation Operators** are used to combine strings.
- **Logical Operators** are used to perform logical operations and include AND, OR, or NOT.
- **Boolean Operators** include AND, OR, XOR, or NOT and can have one of two values, true or false.

Operator Precedence

If several operations occur in an expression, each part is evaluated and resolved in a predetermined order called Operator Precedence. Parentheses can be used to override the order of precedence and evaluate some parts of an expression before others. Operations within parentheses are always performed before those outside. Within parentheses, however, normal Operator Precedence is maintained.

If expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators next, and logical operators last. Comparison operators all have equal precedence; they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence:
### Arithmetic

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation (-)</td>
<td>Equality (=)</td>
<td><code>&lt;expression&gt; &amp; &lt;expression&gt;</code></td>
</tr>
<tr>
<td>Exponentiation (^)</td>
<td>Inequality (&lt;&gt;).</td>
<td></td>
</tr>
<tr>
<td>Multiplication and division (*, /)</td>
<td>Less than (&lt;).</td>
<td></td>
</tr>
<tr>
<td>Integer division ()</td>
<td>Greater than (&gt;).</td>
<td></td>
</tr>
<tr>
<td>Modulus arithmetic (Mod)</td>
<td>Less than or equal to (&lt;=).</td>
<td></td>
</tr>
<tr>
<td>Addition and Subtraction (+, -)</td>
<td>Greater than or equal to (&gt;=).</td>
<td></td>
</tr>
<tr>
<td>String concatenation (&amp;)</td>
<td>IS</td>
<td></td>
</tr>
</tbody>
</table>

### Comparison

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Logical

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If addition and subtraction, multiplication and division, occur together respectively in an expression, each operation is evaluated as it occurs from left to right.

The string concatenation operator (&) is not an arithmetic operator, but in precedence, it does fall after all arithmetic operators and before all comparison operators. The IS operator is an object reference comparison operator. It does not compare objects or their values; it checks only to determine whether two object references refer to the same object.

### & Ampersand

**Description**

This operator forces text string concatenation of two expressions. Text concatenation operator connects or concatenates two values to produce a continuous text value.

**Syntax**

`<expression> & <expression>`

- The `<expression>` represents any valid logical expression.

Whenever an expression is not a string, it is converted to a String subtype. If both expressions are Null, the result is Null. However, if only one expression is Null, that expression is treated as a zero-length string ("") when concatenated with the other expression. Any expression that is Empty is also treated as a zero-length string.

**Example**

```ebnf
READ {C:\My_Project_Folder\Sample\Sample.prj}: Oswego
```
DEFINE NameVar TEXTINPUT
ASSIGN NameVar=LastName&FirstName
LIST NameVar LastName FirstName

= Equal Sign

Description
This operator assigns a value to a variable or property. Comparison operator also used as
an equal to; the result of comparison operators is usually a logical value, either true or
false.

Syntax
<variable> <operator> <value>

- The <variable> represents any variable or any writable property.
- The <value> represents any numeric or string literal, constant, or expression.

Comments
The name on the left side of the equal sign can be a simple scalar variable or an element of
an array. Properties on the left side of the equal sign can only be those writable properties
at run time.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Newvar NUMERIC
ASSIGN Newvar =Age
LIST Newvar Age

Addition (+)

Description
This operator provides the sums of two numbers. Basic arithmetic operator used for
addition; the result of an arithmetic operator is usually a numeric value.

Syntax
[expression1] <operator> [expression2]

Comments
Although the + operator can be used to concatenate two character strings, the & operator should be used for concatenation to eliminate ambiguity and provide self-documenting code. If + operator is used, there may be no way to determine whether addition or string concatenation will occur. The underlying subtype of the expressions determines the behavior of the + operator in the following way:

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both expressions are numeric</td>
<td>Add</td>
</tr>
<tr>
<td>Both expressions are strings</td>
<td>Concatenate</td>
</tr>
<tr>
<td>One expression is numeric and the other is a string</td>
<td>Add</td>
</tr>
</tbody>
</table>

If one or both expressions are Null expressions, the result is Null. If both expressions are Empty, the result is an integer subtype. However, if only one expression is Empty, the other expression is returned unchanged as a result.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Newvar NUMERIC
ASSIGN Newvar = Age + 5
LIST Age Newvar

AND

Description
This operator performs logical conjunction on two Boolean expressions. If both expressions evaluate to True, the AND operator returns True. If either or both expressions evaluate to False, the AND operator returns False.

Syntax
[Logical Expression] AND [Logical Expression]
Comments
The expression is any valid logical expression in Epi Info.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Smoke
DEFINE Result TEXTINPUT
IF Age > 75 AND Sex = 2 THEN
    ASSIGN Result="Senior"
END
SELECT Result = "Senior"
LIST Result Age Sex

In this case, the value of "Senior" is assigned to all records that meet both criteria Age > 75 and Sex = 2.

ARITHMETIC
Description
These basic arithmetic operators can be used in combination with other commands. The result is a numeric value.

Syntax
\([\text{Expression}] \langle \text{Operator} \rangle [\text{Expression}]\)
- \([\text{Expression}]\) is a numeric value or a variable containing data in numeric format.

Comments
The results are expressed in numeric format. The basic mathematical operators that can be used in Epi Info are as follows:

- **Addition** + Basic arithmetic operator used for addition; the result of an arithmetic operator is usually a numeric value (example 3 + 3).
- **Subtraction** – Basic arithmetic operator used for subtraction or negation; the result of an arithmetic operator is usually a numeric value (example 3 – 1).
- **Multiplication** * (Asterisk) Basic arithmetic operator used for multiplication; the result of an arithmetic operator is usually a numeric value.
- **Division** / Basic arithmetic operator used for division; the result of an arithmetic operator is usually a numeric value.
- **Exponentiation** ^
- **Modulus or Remainder MOD**
Arithmetic operators are shown in descending order of precedence. Parentheses can be used to control the order in which operators are evaluated. The default order, however, frequently achieves the correct result.

While it is possible to do date math with dates considered as a number of days (example IncubationDays = SymptomDateTime – ExposureDateTime), the behavior of the database services underlying Epi Info makes it more efficient to use time interval functions (e.g., IncubationDays = MINUTES(ExposureDateTime, SymptomDateTime)/(24*60)). For doing date math, the following rules apply:

Date + Date produces Date
Date – Date produces Days
Date * Date not permitted
Date / Date not permitted
Date ^ Date not permitted
Date + Number produces Date
Number + Date produces Number

The last two rules apply as well to other math operations: -, *, /, ^

The "zero day" for date math is December 30, 1899.

Example

READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE var1 NUMERIC
ASSIGN var1=1250 MOD 100
DEFINE var2 NUMERIC
ASSIGN var2=1+1
DEFINE var3 NUMERIC
ASSIGN var3=2-1
DEFINE var4 NUMERIC
ASSIGN var4=1*1
DEFINE var5 NUMERIC
ASSIGN var5=8/4
DEFINE var6 NUMERIC
ASSIGN var6=5^2
LIST var1 var2 var3 var4 var5 var6
## COMPARISONS

### Description

These comparison operators can be used in If, Then, and Select statements in Check Code and Analysis programs. Yes/No variables can only be tested for equality against other Yes/No constants (+), (-), and (.).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to Comparison operator used for equal to; the result of comparison operators is usually a logical value, either True or False. EX. A1 = B1</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than comparison operator. Compares a value greater than another value; the result of comparison operators is usually a logical value, either True or False. Comparison operator used for comparing a value greater than another value; the result of comparison operators is usually a logical value, either True or False. EX. A1 &gt; B1.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than comparison operator. Compares a value less than another value; the result of comparison operators is usually a logical value, either True or False. Comparison operator used for comparing a value less than another value; the result of comparison operators is usually a logical value, either True or False. EX. A1 &lt; B1</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>LIKE</td>
<td>Left side variable matches right side pattern; in pattern, ‘*’ matches any number of characters, ‘?’ matches any one character.</td>
</tr>
</tbody>
</table>

### Syntax

[Expression] <Operator> [Expression]

[Expression] is any valid expression.
Comments
Comparison operators are executed from left to right. There is no hierarchy of comparison operators. The <> operator can be used only with numeric variables. For non-numeric variables, use NOT.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
SELECT Age>20
LIST Age Disease

READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
SELECT Age<45
LIST Age Disease

READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
SELECT Age>=38
LIST Age Disease

READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
SELECT Age<>77
LIST Age Disease

LIKE Operator

Description
This operator is used with the SELECT command to locate subsets of information using a wildcard search. LIKE can be used only to locate data in text variables and uses asterisks (*) to define the select value. It can also be used to create IF/THEN statements.

Syntax
SELECT <variable> LIKE "*value*"
SELECT <variable> LIKE "*val*"
SELECT <variable> LIKE "v*"
SELECT <variable> LIKE "*v"

- The select variable must be a text type. The value can be a whole or partial text value. Text variables must be enclosed in quotes.

Comments
The results appear in the Output window. Use LIST to view the selected records.

**Examples**

```plaintext
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE Sick NUMERIC
IF Disease LIKE "h*" THEN
    ASSIGN Sick = 0
END
SELECT Disease LIKE "h*"
LIST Age Disease DateAdmitted Sick GRIDTABLE
```

**NOT**

**Description**
This operator reverses the True or False value of the logical expression that follows.

**Syntax**

```
NOT [Expression]
```

The expression represents any valid logical expression in Epi Info.

**Comments**
If the value of an expression is True, NOT returns the value False. If the expression is False, NOT <expression> is True.

**Example**

```plaintext
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE NoVanilla YN
IF NOT Vanilla = (+) THEN
    NoVanilla = (+)
ELSE
    NoVanilla = (-)
END
FREQ NoVanilla Vanilla
```

<table>
<thead>
<tr>
<th>VANILLA</th>
<th>NOVANILLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**OR**
Description
This operator returns True if one or the other or both expressions are True. If either expression evaluates to True, OR returns True. If neither expression evaluates to True, OR returns False.

Syntax
[Logical Expression] OR [Logical Expression]
[Logical Expression] represents any valid logical expression in Epi Info.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE IceCream YN
IF VANILLA=(+) OR CHOCOLATE=(+) THEN
   IceCream=(+)
ELSE
   IceCream=(-)
END
FREQ IceCream

<table>
<thead>
<tr>
<th>VANILLA</th>
<th>CHOCOLATE</th>
<th>ICE CREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

XOR (eXclusive OR)

Description
This operator performs a logical exclusion on two expressions.

Syntax
[Logical Expression] XOR [Logical Expression]
The [Logical Expression] represents any valid logical expression in Epi Info 7 for Windows.

Comments
If one, and only one, of the expressions evaluates to True, the result is True. However, if either expression is Null, the result is also Null. When neither expression is Null, the result is determined according to the following table:

<table>
<thead>
<tr>
<th>If expression1 is</th>
<th>And expression2 is</th>
<th>Then result is</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Example

READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego

DEFINE Oneicecream YN

IF Vanilla = (+) XOR Chocolate = (+) THEN
  Oneicecream = (+)
ELSE
  Oneicecream = (-)
END

LIST Vanilla Chocolate Oneicecream GRIDTABLE
Functions

Do not put a space before the first parenthesis. Functions take the value of one or more variables and return the result of a calculation or transformation.

ABS Function

Description
The ABS function returns the absolute value of a variable by removing the negative sign, if any.

Syntax
ABS<variable>

• The <variable> can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

Comments
Results will be numeric.

<table>
<thead>
<tr>
<th>Value</th>
<th>ABS Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.0025</td>
<td>0.0025</td>
</tr>
</tbody>
</table>

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Age2 NUMERIC
DEFINE Age3 NUMERIC
ASSIGN Age2 = Age * -1
ASSIGN Age3 = ABS(Age2)
LIST Age Age2 Age3
DAY

Description
The DAY function extracts the day from the date.

Syntax
DAY (<variable>)
The <variable> is in date format.

Comments
If the date is stored in a text variable, the function will not be processed, and will be null.

Example
READ {C: \My_Project_Folder \Sample \Sample.prj}:Oswego
DEFINE CurrentDay NUMERIC
ASSIGN CurrentDay = DAY(01/15/2007)
LIST CurrentDay

DAYS

Description
The DAYS function returns the number of days between <var2> and <var1>. If any of the variables or values included in the formula is not a date, the result will be null.

Syntax
DAYS(<var1>, <var2>)
The <variable> is in a date format.

Comments
If the date stored in <var1> is later (more recent) than the date in <var2>, the result is the difference in days expressed as a negative number.

Example
READ {C: \My_Project_Folder \Sample \Sample.prj}:Oswego
DEFINE SickDays NUMERIC
ASSIGN SickDays = DAYS(04/18/1940, DateOnset)
LIST SickDays GRIDTABLE
EXISTS
Description
This function returns True if a file exists. Otherwise, it returns False.

Syntax
EXISTS(<variable>)

<variable> represents the complete path and file name in text format.

Comments
If you do not have permission to access the file, a False may be returned.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE var1 TEXTINPUT
ASSIGN var1="C:\epi_info\epimap.exe"
IF EXISTS(Var1) =(+) then
    DIALOG "Hello"
END
IF Exists("C:\Epi_Info\EpiInfo.mnu")=(+) then
    DIALOG "File epiInfo.mnu exists"
END

EXP
Description
This function raises the base of the natural logarithm (e) to the power specified.

Syntax
EXP(<variable>)

Comments
This variable can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE ExpA NUMERIC
ASSIGN ExpA=EXP(Age)
LIST ExpA Age
FILEDATE

Description
This function returns the date a file was last modified or created. If FILEDATE is specified with a file path that lacks a directory, the current directory is used. If FILEDATE is specified without a file, or with a file that does not exist, the function returns missing.

Syntax
FILEDATE(<variable>)
The <variable> represents the complete file path and the name is text format.

Comments
This function is useful when several users are updating a large database.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:RBHepatitis
DEFINE NewUpdate DATEFORMAT
ASSIGN NewUpdate=FILEDATE("C:\epi_info\Sample.mdb")
IF FILEDATE("C:\epi_info\Sample.mdb") > NewUpdate THEN
    DIALOG "This information may be out of date. Please check the source."
    TITLETEXT="Warning"
END
LIST NewUpdate

FINDTEXT

Description
This function returns the position in a variable in which the string is located.

Syntax
FINDTEXT(<variable1>,<variable2>)
The <variable1> represents the string of characters to be found. The <variable2> represents the string to be searched.

Comments
If the sting is not found, the result is 0; otherwise it is a number corresponding to the position of the string starting from the left. The first character is 1. If the result is 0, the test was not found.
Example
READ [C:\My_Project_Folder\Sample\Sample.prj]:Surveillance
DEFINE Var11 NUMERIC
VAR11=FINDTEXT("M",LASTNAME)
LIST LASTNAME Var11

FORMAT

Description
This function changes the format of one variable type to text in a specified format. If no format is specified it returns text and converts a number to text.

Syntax
FORMAT(<variable>, ["Format Specification"])

The <variable> represents a variable in any format and the [Format Specification] can represent any of the following:

<table>
<thead>
<tr>
<th>Format Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date Formats</strong></td>
<td></td>
</tr>
<tr>
<td>General Date</td>
<td>11/11/1999 05:34</td>
</tr>
<tr>
<td>Long Date</td>
<td>System's long date format</td>
</tr>
<tr>
<td>Medium Date</td>
<td>System's medium date format</td>
</tr>
<tr>
<td>Short Date</td>
<td>System's short date format</td>
</tr>
<tr>
<td>Long Time</td>
<td>System's long time format</td>
</tr>
<tr>
<td>Medium Time</td>
<td>System's medium time format</td>
</tr>
<tr>
<td>Short Time</td>
<td>System's short time format</td>
</tr>
<tr>
<td><strong>Number Formats</strong></td>
<td></td>
</tr>
<tr>
<td>General Number</td>
<td>No thousand separator</td>
</tr>
<tr>
<td>Currency</td>
<td>Thousand separator plus two decimal places (based on system settings)</td>
</tr>
<tr>
<td>Fixed</td>
<td>At least #.##</td>
</tr>
<tr>
<td>Standard</td>
<td>#,###.##</td>
</tr>
<tr>
<td>Format Specification</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Percent</td>
<td>Number multiplied by 100 plus a percent sign</td>
</tr>
<tr>
<td>Scientific</td>
<td>Standard scientific notation</td>
</tr>
<tr>
<td>Yes/No</td>
<td>Displays NO if number = 0, else displays Yes</td>
</tr>
<tr>
<td>True/False</td>
<td>False if number = 0</td>
</tr>
</tbody>
</table>
| On/Off               | True if number <> 0  
| Displays 0 if number = 0, else displays 1 |
| Custom Format        | Allows for the creation of customized formats |

**Comments**

Output may vary based on the specific configuration settings of the local computer.

```
Format(Time, "Long Time")
MyStr = Format(Date,"Long Date")
MyStr = Format(MyTime,"h:m:s")
Returns "17:4:23"
MyStr = Format(MyTime,"hh:mm:ssAMPm")
Returns "05:04:23 PM"
MyStr = Format(MyDate,"dd, mmm yyyy")
Returns "Wednesday, ' Jan 27 1993". If format is not supplied, a string is returned.
MyStr = Format(23)
Returns "23".
User-defined formats
MyStr = Format(5459.4, "##,##0.00")
Returns "5,459.40"
MyStr = Format(334.9, "###0.00")
Returns "334.90"
MyStr = Format(5, "0.00%")
Returns "500.00%"
MyStr = Format("HELLO", ">")
Returns "hello"
MyStr = Format("This is it", ">;*")
Returns "THIS IS IT"
```

**Example**

READ 'C:\Epi_Info\Refugee.MDB':Patient
DEFINE var2 NUMERIC
DEFINE var3 NUMERIC
DEFINE var4 NUMERIC
DEFINE var5 NUMERIC
DEFINE var6 NUMERIC
DEFINE var7 YN
DEFINE var8 Boolean
DEFINE var9
DEFINE var10
var2=FORMAT(BOH, "Currency")
var3=FORMAT(BOH, "fixed")
var4=FORMAT(BOH, "Standard")
var5=FORMAT(BOH, "Percent")
var6=FORMAT(BOH, "Scientific")
var7=FORMAT(BOH, "Yes/No")
var8=FORMAT(BOH, "True/false")
var9=FORMAT(BOH, "On/Off")
var10=FORMAT(BOH, "VB\s #,###.##")
LIST dob var2 var3 var4 var5 var6 var7 var8 var9 var10

HOUR

Description
This function returns a numeric value that corresponds to the hour recorded in a date/time or time variable.

Syntax
HOUR(<variable>)
The <variable> represents a variable in date format.

Comments
If the time is stored in a text variable, the function will not be processed, and the result will be null.
Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Local DATEFORMAT
ASSIGN Local = SYSTEMTIME
LIST Local
DEFINE hour1 NUMERIC
ASSIGN hour1=hour(local)
LIST Local hour1

HOURS

Description
This function returns the number of hours between <var1> and <var2> in numeric format.

Syntax
HOURS(<var1>, <var2>)
<var1> and <var2> represent variables in time or date/time format.

Comments
If the time stored in <var1> is later (more recent) than the time in <var2>, the result will be the difference in hours expressed as a negative number. Both variables must contain data in date, time, or date/time format. If any of the variables or values included in the formula is not a date, the result will be null.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE hour1 NUMERIC
ASSIGN hour1=HOURS(Timesupper,Dateonset)
LIST hour1
LIST hour1 Timesupper Dateonset

LN

Description
The function LN returns the natural logarithm (logarithm in base e) of a numeric value or variable. If the value is zero or null, it returns a null value.
Syntax
LN(<variable>)

The <variable> can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Natlogofage NUMERIC
ASSIGN Natlogofage = LN(AGE)
LIST Age Natlogofage

LOG

Description
This function returns the base 10 logarithm (decimal logarithm) of a numeric value or variable. If the value is 0 or null it returns a null value.

Syntax
LOG(<variable>)

The <variable> can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

Comments
The results will be numeric.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Declog NUMERIC
ASSIGN Declog = LOG(Age)
LIST Age Declog

MINUTES

Description
This function returns the number of minutes between <var1> and <var2> in numeric format.
**Syntax**

MINUTES(<var1>, <var2>)

<var1> and <var2> represent variables in time or date/time format.

**Comments**

If the time stored in <var1> is later (more recent) than the time in <var2>, the result will be the difference in minutes expressed as a negative number. Both variables must contain data in date, time, or date/time format. If any of the variables or values included in the formula is not a date, the result will be null.

**Example**

READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Min1 NUMERIC
ASSIGN Min1=MINUTES(timesupper,dateonset)
LIST Min1

**MONTH**

**Description**

This function extracts the month from the date.

**Syntax**

MONTH(<variable>)

The <variable> represents a variable in date format.

**Comments**

If the date is stored in a text variable, the function will not be processed, and the result will be null.

**Example**

READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE CurrMonth NUMERIC
ASSIGN CurrMonth = MONTH(01/01/2005)
LIST CurrMonth
MONTHS

Description
This function returns the number of months between <var1> and <var2>. If any of the variables or values included in the formula is not a date, the result will be null.

Syntax
MONTHS(<var1>, <var2>)

<var1> and <var2> represent variables in date format.

Comments
If the date stored in <var1> is later (more recent) than the date in <var2>, the result will be the difference in months expressed as a negative number.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE AgeMonths NUMERIC
ASSIGN AgeMonths = MONTHS(BirthDate,01/01/2000)
LIST AgeMonths

NUMTODATE

Description
This function transforms three numbers into a date format.

Syntax
NUMTODATE(<year>, <month>, <day>)

- <year> represents a numeric variable or a number representing the year.
- <month> represents a numeric variable or a number representing the month.
- <day> represents a numeric variable or a number representing the day.

Comments
If the date resulting from the conversion is not valid (e.g., December 41, 2000), the date is recalculated to the corresponding valid value (e.g., January 10, 2001). When <Year> ranges between 0 and 29, it is represented as the respective year between 2000 and 2029. Values from 30 to 99 are represented as the respective year between 1930 and 1999. The earliest date that can be recorded is Jan 01, 100.
Example
READ [C:\My_Project_Folder\Sample\Sample.prj]:Surveillance
DEFINE day1 NUMERIC
DEFINE month1 NUMERIC
DEFINE year1 NUMERIC
ASSIGN day1 = day(BirthDate)
ASSIGN month1 = month(BirthDate)
ASSIGN year1 = year(BirthDate)
define date2 DATEFORMAT
ASSIGN date2 = NUMTODATE(year1, month1, day1)
LIST month1 day1 year1 date2 BirthDate GRIDTABLE

NUMTOTIME

Description
This function transforms three numbers into a time or date/time format.

Syntax
NUMTOTIME(<hour>, <minute>, <second>)

- <hour> represents a numeric constant or variable representing hours.
- <minute> represents a numeric constant or variable representing minutes.
- <second> represents a numeric constant or variable representing seconds.

Comments
Time must be entered in 24-hour format. Invalid dates will be recalculated to the respective valid time. If the number of the hour exceeds 24, the resulting variable will have a date/time format and the default day 1 will be December 31, 1899.
Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE Var3 DATEFORMAT
ASSIGN Var3=SYSTEMTIME
DEFINE Hour1 NUMERIC
DEFINE Minute1 NUMERIC
DEFINE Second1 NUMERIC
ASSIGN Hour1=HOUR(VAR3)
ASSIGN Minute1=MINUTE(VAR3)
ASSIGN Second1=SECOND(VAR3)
DEFINE Time2 DATEFORMAT
ASSIGN Time2=NUMTOTIME(HOUR1,MINUTE1,SECOND1)
LIST Var3 Hour1 Minute1 Second1 Time2

RECORDCOUNT

Description
This function returns the number of records in the current View. In Analysis, this takes into account any SELECT statement and value of the Process (Deleted) setting.

Syntax
RECORDCOUNT
**Example**

READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego

IF RECORDCOUNT=0 THEN
   DIALOG "No records found."
   QUIT
END

**RND**

**Description**

This function generates a random number between `<var1>` and `<var2>`.

**Syntax**

`RND(<min>, <max>)`

- The `<min>` represents a number or numeric variable that corresponds to the lowest value of the random number to be generated.
- The `<max>` represents a number or numeric variable that is one higher than the highest possible value for the random number to be generated.

**Comments**

The random number generated is from `<min>` up to but not including `<max>`. For a set of random numbers consisting of only 0 and 1, the syntax `RND(0, 2)` would be used to generate a random number from 0 up to but not including 2. If the value for `<min>` is greater than the value for `<max>` a syntax error results.

**Example**

READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego

DEFINE Random1 NUMERIC
DEFINE Random2 NUMERIC
DEFINE Random3 NUMERIC
ASSIGN Random1=RND(1,100)
ASSIGN Random2=RND(1,100)
ASSIGN Random3=RND(1,100)
LIST Random1 Random2 Random3
**ROUND**

**Description**
This function rounds the number stored in the variable to the closest integer. Positive numbers are rounded up to the next higher integer if the fractional part is greater than or equal to 0.5. Negative numbers are rounded down to the next lower integer if the fractional part is greater than or equal to 0.5.

**Syntax**

ROUND(<variable>)

The <variable> can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

**Comments**
The results are returned in numeric format.

<table>
<thead>
<tr>
<th>Differences Between TRUNC and ROUND</th>
<th>TRUNC</th>
<th>ROUND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.123456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7.999999999</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>45.545</td>
<td>45</td>
<td>46</td>
</tr>
</tbody>
</table>

**Example**

READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
FREQ AGE
DEFINE Decade NUMERIC
ASSIGN Decade=ROUND(AGE/10)+1
LIST AGE Decade

**SECONDS**

**Description**
This function returns the number of seconds between <var1> and <var2> in numeric format.
**Syntax**
SECONDS(<var1>, <var2>)

<var1> and <var2> represent variables in time or date/time format.

**Comments**
If the time stored in <var1> is later (more recent) than the time in <var2>, the result will be the difference in seconds expressed as a negative number. Both variables must contain data in date, time or date/time format. If any of the variables or values included in the formula is not a date, the result is null.

**Example**
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Sec1 NUMERIC
ASSIGN Sec1=SECONDS(Timesupper,DateOnset)
LIST Timesupper DateOnset Sec1

**SIN, COS, TAN**
**Description**
These functions return the respective trigonometric value for the specified variable.

**Syntax**
SIN(<variable>)

The <variable> can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

**Comments**
The variable is interpreted as the angle in radians. To convert degrees to radians, multiply by pi (3.1415926535897932) divided by 180.

**Example**
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE SinA NUMERIC
DEFINE SinB NUMERIC
DEFINE CosA NUMERIC
DEFINE TanA NUMERIC
ASSIGN SinA=SIN(AGE)
ASSIGN SinB=SIN(AGE)*3.14/180
ASSIGN CosA=COS(AGE)
ASSIGN TanA=TAN(AGE)
LIST SinA CosA TanA SinB
SUBSTRING

Description
This function returns a string that is a specified part of the value in the string parameter.

Syntax
SUBSTRING(<variable>, [First], [Length])

- The <variable> represents a variable in text format.
- The [First] represents the position of the first character to extract from the file.
- The [Length] represents the number of characters to extract.

Comments
This function cannot be used with non-string variables.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Text1 TEXTINPUT
ASSIGN Text1 = "James Smith"
DEFINE LName TEXTINPUT
ASSIGN LName = SUBSTRING(Text1,7,5)
LIST Text1 LName

SYSTEMDATE

Description
This function returns the date stored in the computer's clock.

Syntax
SYSTEMDATE

Comments
The SYSTEMDATE cannot be changed (assigned) from Classic Analysis. To use the SYSTEMDATE for computations, a new variable must be defined.
Example
To calculate next week's date:
```
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE TodayDate DATEFORMAT
ASSIGN TodayDate =SYSTEMDATE + 7
LIST TodayDate
```

SYSTEMTIME

Description
This function returns the time stored in the computer's clock at the time the command is executed.

Syntax
```
SYSTEMTIME
```

Comments
The SYSTEMTIME cannot be changed from Classic Analysis (assigned). To use the system time for computations, a new variable must be defined.

Example
To calculate a time two hours after the current time:
```
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE Later DATEFORMAT
ASSIGN Later =SYSTEMTIME
LIST Later
ASSIGN Later =SYSTEMTIME+(120)
LIST Later
```

TRUNC

Description
This function removes decimals from a numeric variable, returning the integer part of the number. This follows the same logic as rounding toward zero.

Syntax
```
TRUNC(<variable>)
```
The `<variable>` can be an existing numeric variable, a defined variable containing numbers, or a numeric constant.

Comments
The result will be returned in numeric format.

**Example**

```plaintext
READ {C:\My_Project_Folder\Sample\Sample.prj}:ADDFull
   DEFINE Trc1 Numeric
   ASSIGN Trc1 = TRUNC(ADDSC)
LIST Trc1 ADDSC
```

**TXTTODATE**

**Description**

This function returns a date value that corresponds to the string.

**Syntax**

```plaintext
TXTTODATE(<variable>)
```

The `<variable>` represents a variable in text format.

**Comments**

The text variable can be in any format that can be recognized as a date (e.g., "Jan 1, 2000", "1/1/2000").

**Example**

```plaintext
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE Var1 TEXTINPUT
ASSIGN Var1="05/20/2006"
DEFINE Var2 DATEFORMAT
ASSIGN Var2=TXTTODATE(Var1)
DISPLAY DBVARIABLES
LIST Var1 Var2
```

**TXTTONUM**

**Description**

This function returns a numeric value that corresponds to the string.

**Syntax**

```plaintext
TXTTONUM(<variable>)
```

The `<variable>` represents a variable in text format.
Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Oswego
DEFINE Var1 TEXTINPUT
ASSIGN Var1="12345"
DEFINE Var2 NUMERIC
ASSIGN Var2=TXTTONUM(Var1)
LIST Var1 Var2
DISPLAY DBVARIABLES

UPPERCASE

Description
This function returns a string (text) variable that has been converted to uppercase.

Syntax
UPPERCASE(<variable>)

The <variable> represents a variable in text format.

Comments
Only lowercase letters are converted to uppercase; all uppercase letters and non-letter characters remain unchanged.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE LastName2 TEXTINPUT
ASSIGN LastName2 = UPPERCASE(LASTNAME)
LIST LastName2 LASTNAME

YEAR

Description
This function extracts the year from a date.

Syntax
YEAR(<variable>)

The <variable> represents a variable in date format.
Comments
The date argument is any expression that can represent a date. If the date variable contains null, null is returned.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE CurrentYear NUMERIC
ASSIGN CurrentYear =YEAR(01/01/2006)
LIST CurrentYear

YEARS

Description
This function returns the number of years from <var1> to <var2> in numeric format. If any of the variables or values included in the formula is not a date, the result will be null.

Syntax
YEARS(<var1>, <var2>)
<var1> and <var2> are represented in date format.

Comments
If the date stored in <var1> is later (more recent) than the date in <var2>, the result will be the difference in years expressed as a negative number.

Example
READ {C:\My_Project_Folder\Sample\Sample.prj}:Surveillance
DEFINE SurveyDate DATEFORMAT
ASSIGN SurveyDate=05/15/2001
DEFINE AgeYears NUMERIC
ASSIGN AgeYears =YEARS(BirthDate,SurveyDate)
MEANS AgeYears
LIST AgeYears BirthDate SurveyDate