# Formelinterpreter

## Formelsyntax

Eine Formel endet immer mit einem Semikolon.

## Datentypen

- Es stehen folgende Datentypen zur Verfügung:
- Fließkomma
- Ganzzahl
- Datum
- Zeichenkette
- Boolean (Ja/Nein)

## Wertzuweisung

Variablen können in Formeln als konstante oder für Zwischenrechnungen verwendet werden. x1 = ref1 + IF( ref2 < 10, ref3 - 5, ref4 + 5 );

## Arithmetik

| Ausdruck | Bedeutung | Beispiel | Ergebnis |
|---|---|---|---|
| + | Addition | 3+2 | 5 |
| - | Subtraktion<br>Negation | 3-2<br>-3 | 1<br>-3 |
| * | Multiplikation | 3*2 | 6 |
| / | Division | 3/2 | 1.5 |
| ^ | Potenzierung | 3^2 | 9 |

Die Negation von Funktionen ist nicht zulässig: Falsch: -SUM( x1; x2 ); Richtig: 0 - SUM( x1, x2 );

Negation von Zahlen oder Variablen innerhalb arimethischer Ausdrücke ist ebenfalls nicht zulässig: Falsch: X1 + -X2; Richtig: X1 - X1;

Mehrfaches Potenzieren: 3^2^4; entspricht 3^(2^4);

## Vergleichsoperatoren

| Ausdruck | Bedeutung | Beispiel | Ergebnis |
|---|---|---|---|
| == | Gleich | 3==7; | FALSE |
| > | Größer als | 3>7; | FALSE |

| Ausdruck | Bedeutung | Beispiel | Ergebnis |
|---|---|---|---|
| < | Kleiner als | 3<7 | TRUE |
| >= | Größer gleich | 3>=7; | FALSE |
| ⇐ | Kleiner gleich | 3⇐7; | True |
| != | Ungleich | 3!=7; | TRUE |

# Logik

**AND(Wert1, …, WertX);**
Liefert TRUE, wenn alle Argumente TRUE sind. Liefert FALSE, wenn mindestens ein Argument FALSE ist.

**OR(Wert1, …, WertX);**
Liefert FALSE, wenn alle Argumente FALSE sind. Liefert TRUE, wenn mindestens ein Argument TRUE ist.

**XOR(Wert1, Wert2);**
Liefert TRUE, wenn genau ein Argument TRUE ist. Liefert FALSE, wenn mindestens zwei Argumente TRUE oder alle Argumente FALSE sind.

**NOT(Wert);**
Liefert TRUE, wenn das Argument FALSE ist. Liefert FALSE, wenn das Argument TRUE ist.

# Bedingungen & Prüfungen

**IF(Prüfung, DannWert, SonstWert);**

Liefert DannWert, wenn Prüfung TRUE ist. Ist DannWert nicht angegeben, wird TRUE zurückgegeben. Liefert SonstWert, wenn Prüfung FALSE ist. Ist SonstWert nicht angegeben, wird FALSE zurückgegeben.

Für die Erstellung komplexerer Bedingungen lassen sich maximal sieben IF-Funktionen verschachteln. Sobald die Argumente DannWert und SonstWert ausgewertet sind, liefert IF den Wert, den diese Anweisungen zurückgeben.

Ist eines der an eine IF-Funktion übergebende Argument eine Matrix, so wird bei der Ausführung dieser IF-Anweisung jedes Element der Matrix ausgewertet. Sind einige der als DannWert und SonstWert übergebenen Argumente ausführbare Funktionen, werden diese vollständig ausgeführt.

Beispiel: IF(A==B, 100, 0); ergibt 100, wenn A identisch B ist. IF(Gesamtsumme>1000, „Mehr als Eintausend“, „Weniger als Eintausend“);

**Beachten Sie, dass das weglassen des Dannwert oder des Sonstwert zu undefinierten zuständen in nachfolgenden Formeln führen kann. Sie sollten immer beide Fälle angeben, um Formelfehler in nachfolgenden Formeln zu vermeiden.**

**SWITCH(Testwert, Vergleich1, Anweisung1, …, VergleichX, AnweisungX);**

Liefert AnweisungN, wenn VergleichN identisch ist mit Testwert. Diese Funktion benötigt mindestens drei Argumente.

Als Argument kann der String „DEFAULT“ für den Standardfall verwendet werden.

Beispiel: SWITCH(BEN1_deu, „Frankreich“, „Baguette“, „England“, „Fish and Chips“, „DEFAULT“, „Leberkäse“); ergibt „Leberkäse“, wenn in BEN1_deu weder Frankreich noch England steht.

**Beachten Sie, dass „DEFAULT“ der letzte Vergleichswert ist, denn „DEFAULT“ trifft immer zu**

# Mathematik & Statistik

**ABS(Zahl);**
Gibt den absoluten Wert der Zahl zurück.

**ATAN(Zahl);**
Berechnet den Arkustangens der Zahl.

**COS(Zahl);**
Berechnet den Kosinus der Zahl.

**EXP(Zahl);**
Berechnet den Exponenten zur Basis e.

**EXP10(Zahl);**
Berechnet den Exponenten zur Basis 10.

**POW(X, Y);**
Berechnet xy

**LOG(Zahl);**
Berechnet den Logarithmus zur Basis 10.

**LN(Zahl);**
Berechnet den Logarithmus zur Basis e.

**RND();**
Liefert eine Zufallszahl zurück.

**SIN(Zahl);**
Berechnet den Sinus der Zahl.

**SQRT(Zahl);**
Berechnet die Quadratwurzel der Zahl.

**TAN(Zahl);**
Berechnet den Tangens der Zahl.

**MOD(X, Y);**
Berechnet den Divisionsrest von X dividiert durch Y.

# Datentypumwandlung

**INT(Wert);**
Gibt den ganzzahligen Wert der Zahl zurück.

**ATOI(Wert);**
Wandelt dir Zeichenkette in eine Ganzzahl um.

**ATOF(Wert);**
Wandelt die Zeichenkette in eine Fließkommazahl um.

**ITOA(Zahl);**
Gibt die Ganzzahl als Zeichenkette zurück.

**RTOS(Zahl);**
Gibt die Fließkommazahl als Zeichenkette zurück.

**ROUND(Zahl, Stellen);**
Rundet die Fließkommazahl auf die angegebenen Stellen ab.

**CEIL(Zahl);**
Rundet die Zahl auf die nächst höhere Ganzzahl auf.

**FLOOR(Zahl);**
Rundet die Zahl auf nächst kleinere Ganzzahl ab.

**MIN(Wert1, …, WertX);**
Ermittelt den minimalen Wert der übergebenen Zahlen.

**MAX(Wert1, …, WertX);**
Ermittelt den maximalen Wert der übergebenen Zahlen.

# Zeichenkettenverarbeitung

**STRUPR(Text);**
Wandelt die Zeichenkette in Großbuchstaben um.

**STRLWR(Text);**
Wandelt die Zeichenkette in Kleinbuchstaben um.

**STRCAT(Text, …);**
Hängt die übergebenen Zeichenketten aneinander und gibt eine Zeichenkette zurück.

**FORMAT(Formatbeschreibung, Wert);**
Formatiert den Wert entsprechend der Formatbeschreibung.

**LEN(Text);**
Gibt die Länge der Zeichenkette zurück.

**LEFT(Text, Länge);**
Gibt den linken Teil der Zeichenkette mit der definierten Länge zurück.

**MID(Text, Position, Länge);**
Gibt einen Teil der Zeichenkette beginnend mit der Position und der Länge zurück.

**RIGHT(Text, Länge);**
Gibt den rechten Teil der Zeichenkette mit der definierten Länge zurück.

**FIND(Text, Suchtext [, Startposition]);**
Gibt die Position des zu suchenden Texts in einer Zeichenkette zurück. Wenn der Suchtext nicht enthalten ist wird -1 zurückgegeben.

**CHR(Zeichencode);**
Gibt einen Text zurück, der das Zeichen enthält, das dem angegebenen Zeichencode entspricht.

**ASC(Text);**
Gibt den Zeichencode entsprechend dem ersten Buchstaben in der Zeichenfolge zurück.

**ATRIM(Text);**
Schneidet alle führenden und nachfolgenden Leerzeichen von der Zeichenkette ab.

**LTRIM(Text);**
Schneidet alle führenden Leerzeichen von der Zeichenkette ab.

**RTRIM(Text);**
Schneidet alle nachfolgenden Leerzeichen von der Zeichenkette ab.

**REPLACE(Text, Suchtext, Ersatztext);**
Ersetzt alle im Text vorkommenden Suchtextpassagen durch den Ersatztext.

# Pfade und Dateinamen

**PathAddBackslash(Path)**
Adds a backslash to the end of a string to create the correct syntax for a path. If the source path already has a trailing backslash, no backslash will be added.

**PathAddExtension(Path, Ext)**
Adds a file extension to a path string.

**PathAppend(Path, More)**
Appends one path to the end of another.

**PathBuildRoot(Path)**
Creates a root path from a given drive number.

**PathCombine(Dir, File)**
Concatenates two strings that represent properly formed paths into one path, as well as any relative path pieces.

**PathFileExists(File)**

Determines whether a path to a file system object such as a file or directory is valid.

**PathFindExtension(Path)**
Searches a path for an extension.

**PathFindFileName(Path)**
Searches a path for a file name.

**PathGetDriveNumber(Path)**
Searches a path for a drive letter within the range of 'A' to 'Z' and returns the corresponding drive number.

**PathRemoveBackslash(Path)**
Removes the trailing backslash from a given path.

**PathRemoveExtension(Path)**
Removes the file extension from a path, if there is one.

**PathRemoveFileSpec(Path)**
Removes the trailing file name and backslash from a path, if it has them.

**PathRenameExtension(Path, Ext)**
Replaces the extension of a file name with a new extension. If the file name does not contain an extension, the extension will be attached to the end of the string.

**PathSkipRoot(Path)**
Parses a path, ignoring the drive letter or UNC server/share path parts.

**PathStripPath(Path)**
Removes the path portion of a fully qualified path and file.

**PathStripToRoot(Path)**
Removes all parts of the path except for the root information.

**PathUndecorate(Path)**
Removes the decoration from a path string.

**PathDecorate(Path, Decorate)**
Adds a decoration to the given path string.

# Datum- und Zeitfunktionen

**DATE(Jahr, Monat, Tag);**
Gibt einen Wert vom Typ Datum zurück, der die angegebene Jahres-, Monats- und Tageszahl enthält.

**DATESTR(Datum [, Format]);**
Gibt das Datum als Zeichenkette formatiert zurück.

**DATEVAL(Text);**
Analysiert die Zeichenkette und wandelt sie in einen Datumswert um.

**DAY(Datum);**
Ergibt den Tag des Monats im Bereich 1 bis 31.

**MONTH(Datum);**
Ergibt den Monat des Jahr im Bereich 1 bis 12.

**YEAR(Datum);**
Ergibt die Jahreszahl.

**NOW();**
Ergibt das aktuelle Datum.

**WEEKDAY(Datum);**
Gibt den Wochentag als Ganzzahl zurück. 0 entspricht Sonntag.

**WOY(Datum);**
Gibt die Kalenderwoche zurück.

# Einschränkungen

### Zahlenbereiche

Ganzzahl
Fließkommazahl

### Grenzwerte der Funktionen und Operatoren

Formellänge
Addition, Subtraktion, Multiplikation, Division
Logische Verknüpfungen
Sonstige Funktionen

From:
https://wiki.speedy-pdm.de/ - **speedyPDM - Wiki**

Permanent link:
**https://wiki.speedy-pdm.de/doku.php?id=speedy:30_modules:interpreter&rev=1461770044**

Last update: **2016/04/27 17:14**