Institut für Informatik

Datenbanksysteme
Lehr- und Forschungseinheit 2

Ludwig Maximilians Universität München
LMU

SML zum Mitnehmen

Eine Kurzreferenz von SML – Funktionen

(Version 0.1)

Eshref Januzaj

Ludwig-Maximilians-Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme
Oettingenstraße 67, D-80538 München

Inhaltsverzeichnis

1 Ganze Zahlen (der SML-Typ **`int`**)

`~` 1
`*` 2
`+` 2
`-` 2
`>` 3
`>=` 3
`<` 3
`<=` 3
`=` 4
`abs` 4
`compare` 6
`div` 8
`max` 14
`min` 15
`mod` 15
`sameSign` 21
`sign` 22
`toString` 26

2 Reelle Zahlen (der SML-Typ **`real`**)

`~` 1
`*` 2
`+` 2
`-` 2
`/` 3
`>` 3
`>=` 3
`<` 3
`<=` 3
`==` 4
`abs` 4
`ceil` 5
`compare` 6
`floor` 9
`fromInt` 10
`max` 14
`min` 15
`realCeil` 19
`realFloor` 19
`realTrunc` 19
`round` 21
`sameSign` 21

sign 22
toString 26
trunc 26

3 Andere mathematische Funktionen (**Math.***)

cos 7
e 8
exp 9
ln 14
log10 14
pi 18
pow 18
sin 22
sqrt 23
tan 25

4 Boole'sche Werte (der SML-Typ **bool**)

andalso 5
not 16
orelse 17
toString 26

5 Zeichenfolgen (der SML-Typ **string**)

^ 1
concat 7
explode 9
implode 11
isPrefix 12
size 22
str 23
sub 23
substring 24

6 Zeichen (der SML-Typ **char**)

chr 6
contains 7
isAlpha 11
isAlphaNum 11
isDigit 12
isLower 12
isUpper 13
notContains 16

ord 17
pred 18
succ 24
toLower 25
toUpper 26

7 Listen im SML (**List.***)

@ 1
concat 7
drop 8
hd 10
last 13
length 13
nth 16
null 17
rev 20
take 24
tl 25

8 Andere vordefinierte SML-Funktionen (**OS.FileSys.***)

chDir 5
getDir 10
mkDie 15
remove 20
rmDir 20
use 27

Vorwort

Diese Kurzreferenz, die einige ausgewählte SML-Funktionen beschreibt, soll den Studenten der Informatik 1 als eine kleine Hilfe, in Form einer begleitenden Funktionssammlung, bei der Bearbeitung von Hausaufgaben dienen! Diese Kurzreferenz erhebt nicht den Anspruch auf Vollständigkeit und für mögliche Fehler gibt es keine Garantie. Verbesserungsvorschläge sind jederzeit willkommen.

Einige schon in der Vorlesung zu Informatik 1 behandelten SML–Funktionen, Operationen bzw. Vergleichsoperatoren etc., werden hier nicht mehr besprochen.

Alle hier beschriebenen SML–Funktionen wurden mit dem SML–Compiler von New Jersey getestet. (`Standard ML of New Jersey, Version 110.0.7, September 28, 2000 [CM&CMB])`

Eshref Januzaj (januzaj@dbs.informatik.uni-muenchen.de)

München, den 21.12.2001

Alphabetisch sortierte SML-Funktionen und -Operatoren

^ — string1 ^ string2, String.^(string1,string2)

Signatur : `val ^ : (string * string) -> string`

Beschreibung : Hängt die zwei Zeichenketten **`string1`** und **`string2`** zusammen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- "Infor" ^ "matik";
val it = "Informatik" : string
- String.^("Infor", "matik");
val it = "Informatik" : string
-
```

@ — liste1 @ liste2, List.@(liste1,liste2)

Signatur : `val @ : ('a list * 'a list) -> 'a list`

Beschreibung : Verkettet zwei Listen zu einer einzigen Liste (Konkatenation).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- [1,2,3] @ [4,5,6];
val it = [1,2,3,4,5,6] : int list
- List.@ ([1,2,3], [4,5,6]);
val it = [1,2,3,4,5,6] : int list
-
```

~ — ~x

Signatur :
```
val ~ : int  -> int
val ~ : real -> real
```

Beschreibung : Ändert das Vorzeichen von **`x`** bzw. liefert den negativen Wert von **`x`**. Gilt für ganze und reelle Zahlen gleich.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- ~1;
val it = ~1 : int
-
```

+ x + y

Signatur : `val + : (int * int) -> int`
`val + : (real * real) -> real`

Beschreibung : Addition von **x** und **y**. Gilt für ganze und reelle Zahlen gleich. Das Ergebnis wird dann in den jeweiligen Typ ausgegeben.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 4 + 2;
val it = 6 : int
- 4.0 + 4.0;
val it = 8.0 : real
-
```

– x - y

Signatur : `val – : (int * int) -> int`
`val – : (real * real) -> real`

Beschreibung : Subtraktion von **x** und **y**. Gilt für ganze und reelle Zahlen gleich. Das Ergebnis wird dann in den jeweiligen Typ ausgegeben.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 4 – 2;
val it = 4 : int
- 6.3 – 2.0;
val it = 4.3 : real
-
```

* x * y

Signatur : `val * : (int * int)  -> int`
`val * : (real * real) -> real`

Beschreibung : Liefert das Produkt von **x** und **y** (Multiplikation). Gilt für ganze und reelle Zahlen gleich. Das Ergebnis wird dann in den jeweiligen Typ ausgegeben.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 3*4;
val it = 12 : int
- 4.0 * 6.3;
val it = 25.2 : real
-
```

`/` x / y

Signatur : `val / : (real * real) -> real`

Beschreibung : Die Division von **`x`** durch **`y`**.
Achtung: Gilt nicht für ganze Zahlen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 3.5 / 2.2;
val it = 1.59090909091 : real
- 6.0 / 2.0;
val it = 3.0 : real
-
```

`>, >=, <, <=` x > y, x >= y, x < y, x <= y

Signatur :

```
val >  : (int * int)|(real * real) -> bool
val >= : (int * int)|(real * real) -> bool
val <  : (int * int)|(real * real) -> bool
val <= : (int * int)|(real * real) -> bool
```

Beschreibung : Vergleichsoperatoren, die **`true`** oder **`false`** zurückliefern. Gelten für ganze und reelle Zahlen gleich.
Achtung: Diese Vergleichsoperatoren gelten auch für Zeichenfolgen (Typ **`string`**). Siehe auch den Vergleichsoperator „**=**“!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 2 < 3;
val it = true : bool
- 5 >= 9;
val it = false : bool
-
```

= x = y

Signatur : `val =  : (int * int) -> bool`

Beschreibung : Vergleicht zwei ganze Zahlen **x** und **y** und liefert als Ergebnis **true** oder **false** zurück.
Achtung: Gilt nicht für reelle Zahlen. Gilt aber für Zeichenfolgen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 2 = 3;
val it = false : bool
- 5 = 5;
val it = true : bool
- "Info" = "Mathe";
val it = false : bool
-
```

== Real.==(x, y)

Signatur : `val == : (real * real) -> bool`

Beschreibung : Vergleicht zwei reelle Zahlen **x** und **y** und liefert als Ergebnis **true** oder **false** zurück.
Achtung: Gilt nicht für ganze Zahlen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.==(2.0, 4.8);
val it = false : bool
- Real.==(4.0, 4.0);
val it = true : bool
-
```

abs(x) Int.abs(x), Real.abs(x)

Signatur :

```
val abs : int  -> int
val abs : real -> real
```

Beschreibung : Liefert den Betrag von **x**. Gilt für ganze und reelle Zahlen gleich.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.abs(~3);
val it = 3 : int
- Real.abs(~2.0);
val it = 2.0 : real
-
```

andalso bool1 andalso bool2

Signatur : `val andalso : (bool * bool) -> bool`

Beschreibung : Während der Auswertung des Ausdruckes wird zunächst nur **bool1** ausgewertet. Hat **bool1** den Wert **false**, so wird **bool2** nicht ausgewertet und als Rückgabewert wird dann **false** geliefert. Hat **bool1** den Wert **true**, so wird dann auch **bool2** ausgewertet. **true** wird nur dann als Rückgabewert geliefert, wenn beide, **bool1** und **bool2**, den Wert **true** haben.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- true andalso false;
val it = false : bool
- true andalso true;
val it = true : bool
-
```

ceil(x) Real.ceil(x)

Signatur : `val ceil : real -> int`

Beschreibung : Typumwandlung von **real** nach **int**. Liefert dabei die nächstgrößere ganze Zahl.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.ceil(3.2);
val it = 4 : int
- Real.ceil(3.9);
val it = 4 : int
- Real.ceil(3.0);
val it = 3 : int
-
```

chDir(verzeichnis) OS.FileSys.chDir(verzeichnis)

Signatur : `val chDir : string -> unit`

Beschreibung : **chDir** wechselt in den Verzeichnis (**verzeichnis**).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- OS.FileSys.chDir "test";
val it = () : unit
- OS.FileSys.chDir "..";
val it = () : unit
-
```

`chr(x)` — Char.chr(x)

Signatur : `val chr : int -> char`

Beschreibung : Liefert das Zeichen mit dem Code **`x`**. (ASCII-Zeichensatz)

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.chr(105);
val it = #"i" : char
- Char.chr(85);
val it = #"U" : char
-
```

`compare(x,y)` — Int.compare(x,y), Real.compar(x,y)

Signatur :
```
val compare : (int * int)   -> order
val compare : (real * real) -> order
```

Beschreibung : **`Int.compare`** Vergleicht zwei ganze Zahlen **`x`** und **`y`** und liefert als Ergebnis LESS (kleiner), EQUAL (gleich) oder GREATER (größer) zurück.
`Real.compare` Vergleicht zwei reelle Zahlen **`x`** und **`y`** und liefert als Ergebnis LESS (kleiner), EQUAL (gleich) oder GREATER (größer) zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.compare(2, 4);
val it = LESS : order
- Int.compare(3, 3);
val it = EQUAL : order
- Real.compare(4.0, 2.0);
val it = GREATER : order
-
```

concat — String.concat list_of_string, List.concat(liste)

Signatur : `val concat : string list -> string`
`val concat : 'a list list -> 'a list`

Beschreibung : **`String.concat`** hängt die Listenelemente (vom Typ `string`) zu einer Zeichenkette zusammen.
`List.concat` liefert eine neue Liste als Verkettung aller Elemente der Liste (**`liste`**), die wiederum selbst Listen sind.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.concat(["Inf", "or", "mat", "ik"]);
val it = "Informatik" : string
- List.concat ([ [1,2,3], [4,5,6], [7,8,9] ]);
val it = [1,2,3,4,5,6,7,8,9] : int list
-
```

contains string char — Char.contains string char

Signatur : `val contains : string -> char -> bool`

Beschreibung : Liefert `true`, wenn **`char`** in **`string`** enthalten ist. Sonst `false`.
Anmerkung : Nicht mit **`Char.notContains`** verwechseln!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.contains "Informatik"  #"a";
val it = true : bool
- Char.contains "Informatik"  #"z";
val it = false : bool
-
```

cos(x) — Math.cos(x)

Signatur : `val cos : real -> real`

Beschreibung : Berechnet Kosinus von **`x`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.cos(0.0);
val it = 1.0 : real
-
```

div — x div y

Signatur : `val div : (int * int) -> int`

Beschreibung : Die Division von **`x`** durch **`y`**, ohne Rest.
Achtung: Gilt nicht für reelle Zahlen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 7 div 2;
val it = 3 : int
-
```

drop(liste,n) — List.drop(liste,n)

Signatur : `val drop : ('a list * int) -> 'a list`

Beschreibung : **`List.drop`** liefert die restlichen Elemente der Liste zurück, die nach dem Löschen von ersten *n* Elementen aus der Liste (**`liste`**) übrig bleiben.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.drop ([1,2,3,4,5], 3);
val it = [4,5] : int list
-
```

e — Math.e

Signatur : `val e : real`

Beschreibung : Liefert den Wert der Zahl **`e`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.e;
val it = 2.71828182846 : real
- 2.0 * Math.e;
val it = 5.43656365692 : real
-
```

`exp(x)` — Math.exp(x)

Signatur : `val exp : real -> real`

Beschreibung : Exponentialfunktion von **`x`** (e^x).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.exp(1.0);
val it = 2.71828182846 : real
- Math.exp(10.0);
val it = 22026.4657948 : real
-
```

`explode(string)` — String.explode(string)

Signatur : `val explode : string -> char list`

Beschreibung : Typumwandlung einer Zeichenkette zu einer Liste von Zeichen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.explode("abc");
val it = [#"a",#"b",#"c"] : char list
-
```

`floor(x)` — Real.floor(x)

Signatur : `val floor : real -> int`

Beschreibung : Typumwandlung von **`real`** nach **`int`**. Liefert dabei die nächstkleinere ganze Zahl nicht größer als **`x`** zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.floor(3.2);
val it = 3 : int
- Real.floor(3.9);
val it = 3 : int
-
```

`fromInt(x)` — Real.fromInt(x)

Signatur : `val fromInt : int -> real`

Beschreibung : Führt für **`x`** eine Typumwandlung vom `int` nach `real` durch.
Anmerkung : Auch die SML-Funktion `real()` liefert dasselbe Ergebnis (!!).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.fromInt(3);
val it = 3.0 : real
- real(3);
val it = 3.0 : real
-
```

`getDir()` — OS.FileSys.getDir()

Signatur : `val getDir : unit -> string`

Beschreibung : Liefert den absoluten Pfad des aktuellen Verzeichnisses.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- OS.FileSys.getDir();
val it = "D:\\sml\\beispiele\\test" : string
-
```

`hd(liste)` — List.hd(liste)

Signatur : `val hd : 'a list -> 'a`

Beschreibung : **`hd`** liefert das erste Elemente der Liste (**`liste`**) zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.hd ([1,2,3,4,5]);
val it = 1 : int
-
```

`implode(zeichen list)` — String.implode(zeichen list)

Signatur : `val implode : char list -> string`

Beschreibung : Typumwandlung einer Liste von Zeichen zu einer Zeichenkette.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.implode [#"a", #"b", #"c"];
val it = "abc" : string
-
```

`isAlpha(char)` — Char.isAlpha(char)

Signatur : `val isAlpha : char -> bool`

Beschreibung : **`isAlpha`** überprüft, ob **`char`** eine Buchstabe ist.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.isAlpha(#"5");
val it = false : bool
- Char.isAlpha(#"a");
val it = true : bool
-
```

`isAlphaNum(char)` — Char.isAlphaNum(char)

Signatur : `val isAlphaNum : char -> bool`

Beschreibung : **`isAlphaNum`** überprüft, ob **`char`** ein Alphanumerisches Zeichen ist.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.isAlphaNum(#"4");
val it = true : bool
- Char.isAlphaNum(#"b");
val it = true : bool
-
```

`isDigit(char)` — Char.isDigit(char)

Signatur : `val isDigit : char -> bool`

Beschreibung : **`isDigit`** überprüft, ob **`char`** eine Ziffer ist.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.isDigit(#"a");
val it = false : bool
- Char.isDigit(#"5");
val it = true : bool
-
```

`isLower(char)` — Char.isLower(char)

Signatur : `val isLower : char -> bool`

Beschreibung : **`isLower`** überprüft, ob **`char`** kleingeschrieben wird.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.isLower(#"a");
val it = true : bool
-
```

`isPrefix string1 string2` — String.isPrefix string1 string2

Signatur : `val isPrefix : string -> string -> bool`

Beschreibung : Überprüft, ob **`string1`** Präfix vom **`string2`** ist.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.isPrefix "Haus" "Hausfest";
val it = true : bool
- String.isPrefix "Haus" "Unifest";
val it = false : bool
-
```

`isUpper(char)` — Char.isUpper(char)

Signatur : `val isUpper : char -> bool`

Beschreibung : **`isUpper`** überprüft, ob **`char`** großgeschrieben wird.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.isUpper(#"a");
val it = false : bool
-
```

`last(liste)` — List.last(liste)

Signatur : `val last : 'a list -> 'a`

Beschreibung : **`last`** liefert das letzte Elemente der Liste (**`liste`**) zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.last ([1,2,3,4,5]);
val it = 5 : int
-
```

`length(liste)` — List.length(liste)

Signatur : `val length : 'a list -> int`

Beschreibung : Liefert die Anzahl der Elemente der Liste (**`liste`**) zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.length ([1,2,3,4,5]);
val it = 5 : int
-
```

`ln(x)` — Math.ln(x)

Signatur : `val ln : real -> real`

Beschreibung : Berechnet den natürlichen Logarithmus von **`x`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.ln(1.0);
val it = 0.0 : real
- Math.ln(2.0);
val it = 0.69314718056 : real
- Math.ln(Math.e);
val it = 1.0 : real
-
```

`log10(x)` — Math.log10(x)

Signatur : `val log10 : real -> real`

Beschreibung : Berechnet den Logarithmus zur Basis 10 von **`x`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.log10(1.0);
val it = 0.0 : real
- Math.log10(2.0);
val it = 0.301029995664 : real
- Math.log10(10.0);
val it = 1.0 : real
-
```

`max(x,y)` — Int.max(x,y), Real.max(x,y)

Signatur :
```
val max : (int * int)   -> int
val max : (real * real) -> real
```

Beschreibung : **`Int.max`** liefert die größte ganze Zahl (**`x`** oder **`y`**).
`Real.max` liefert die größte reelle Zahl (**`x`** oder **`y`**).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.max(3, 5);
val it = 5 : int
- Real.max(3.0, 5.0);
val it = 5.0 : real
-
```

min(x,y) — Int.min(x,y), Real.min(x,y)

Signatur :
```
val min : (int * int)   -> int
val min : (real * real) -> real
```

Beschreibung : **`Int.min`** liefert die kleinste ganze Zahl (**`x`** oder **`y)`**.
`Real.min` liefert die kleinste reelle Zahl (**`x`** oder **`y)`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.min(3, 5);
val it = 3 : int
- Real.min(7.3, 5.0);
val it = 5.0 : real
-
```

mkDir(verzeichnis) — OS.FileSys.mkDir(verzeichnis)

Signatur : `val mkDir : string -> unit`

Beschreibung : **`mkDir`** erstellt einen neuen Verzeichnis (**`verzeichnis`**).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- OS.FileSys.mkDir "test";
val it = () : unit
-
```

mod — x mod y

Signatur :
```
val mod : (int * int) -> int
```

Beschreibung : Liefert den Rest der Division von **`x`** durch **`y`**.
Achtung: Gilt nicht für reelle Zahlen.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- 7 mod 2;
val it = 1 : int
-
```

`not` — not wert, Bool.not wert

Signatur : `val not : bool -> bool`

Beschreibung : Liefert die logische Negation des Bool'schen Wertes von **`wert`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- not true;
val it = false : bool
- Bool.not false;
val it = true : bool
-
```

`notContains string char` — Char.notContains string char

Signatur : `val notContains : string -> char -> bool`

Beschreibung : Liefert `true`, wenn **`char`** in **`string`** nicht enthalten ist. Sonst `false`.
Anmerkung : Nicht mit **`Char.contains`** verwechseln!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.notContains "Informatik"  #"a";
val it = false : bool
- Char.notContains "Informatik"  #"z";
val it = true : bool
-
```

`nth(liste,n)` — List.nth(liste,n)

Signatur : `val nth : ('a list * int) -> 'a list`

Beschreibung : **`nth`** liefert das ***n***-te Elemente der Liste (**`liste`**) zurück.
Achtung: Die Zählung der Position beginnt mit 0!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.nth ([1,2,3,4,5], 4);
val it = 5 : int
-
```

`null(liste)` — List.null(liste)

Signatur : `val null : 'a list -> bool`

Beschreibung : Überprüft ob die Liste (**`liste`**) leer ist oder nicht.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.null ([1,2,3]);
val it = false : bool
- List.null ([]);
val it = true : bool
-
```

`ord(x)` — Char.ord(x)

Signatur : `val ord : char -> int`

Beschreibung : Liefert den Code eines Zeichens **`x`**. (ASCII-Zeichensatz)

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.ord(#"I");
val it = 73 : int
- Char.ord(#"s");
val it = 115 : int
-
```

`orelse` — bool1 orelse bool2

Signatur : `val orelse : (bool * bool) -> bool`

Beschreibung : Während der Auswertung des Ausdruckes wird zunächst nur **`bool1`** ausgewertet. Hat **`bool1`** den Wert **`true`**, so wird **`bool2`** nicht ausgewertet und als Rückgabewert wird dann **`true`** geliefert. Hat **`bool1`** den Wert **`false`**, so wird dann auch **`bool2`** ausgewertet. **`true`** wird dann als Rückgabewert geliefert, wenn mindestens einer von beiden Parametern, **`bool1`** oder **`bool2`**, den Wert **`true`** haben.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- true orelse false;
val it = true : bool
- false orelse true;
val it = true : bool
-
```

pi — Math.pi

Signatur : `val pi : real`

Beschreibung : Liefert den Wert der mathematischen Zahl **`Pi`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.pi;
val it = 3.14159265359 : real
- 3.0 * Math.pi;
val it = 9.42477796077 : real
-
```

pow(x,y) — Math.pow(x,y)

Signatur : `val pow : (real * real) -> real`

Beschreibung : Potenzberechnung x^y.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.pow(2.0, 3.0);
val it = 8.0 : real
- Math.pow(3.5, 2.3);
val it = 17.8384248045 : real
-
```

pred(char) — Char.pred(char)

Signatur : `val pred : char -> char`

Beschreibung : Liefert das vorhergehende Zeichen vor **`char`** zurück (ASCII - Tabelle).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.pred(#"b");
val it = #"a" : char
- Char.pred(#"n");
val it = #"m" : char
-
```

`realCeil(x)` Real.realCeil(x)

Signatur : `val realCeil : real -> real`

Beschreibung : Rundet **`x`** zur nächstgrößeren reellen Zahl auf.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.realCeil(3.1);
val it = 4.0 : real
- Real.realCeil(3.99);
val it = 4.0 : real
-
```

`realFloor(x)` Real.realFloor(x)

Signatur : `val realFloor : real -> real`

Beschreibung : Rundet **`x`** zur nächstkleineren reellen Zahl ab.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.realFloor(2.1);
val it = 2.0 : real
- Real.realFloor(2.99);
val it = 2.0 : real
- Real.realFloor(~2.3);
val it = ~3.0 : real
-
```

`realTrunc(x)` Real.realTrunc(x)

Signatur : `val realTrunc : real -> real`

Beschreibung : Rundet **`x`** zur nächsten reellen Zahl in Richtung Null ab.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.realTrunc(5.1);
val it = 5.0 : real
- Real.realTrunc(5.99);
val it = 5.0 : real
- Real.realTrunc(~2.3);
val it = ~2.0 : real
-
```

remove(datei) OS.FileSys.remove(datei)

Signatur : `val remove : string -> unit`

Beschreibung : Löscht die Datei (**datei**) aus dem Filesystem des Betriebssystems.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- OS.FileSys.remove "test.sml";
val it = () : unit
-
```

rev(liste) List.rev(liste)

Signatur : `val rev : 'a list -> 'a list`

Beschreibung : Liefert die Liste (**liste**) in umgekehrter Reihenfolge zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.rev ([1,2,3,4,5]);
val it = [5,4,3,2,1] : int list
-
```

rmDir(verzeichnis) OS.FileSys.rmDir(verzeichnis)

Signatur : `val rmDir : string -> unit`

Beschreibung : **rmDir** löscht den Verzeichnis (**verzeichnis**).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- OS.FileSys.rmDir "test";
val it = () : unit
-
```

`round(x)` — Real.round(x)

Signatur : `val round : real -> int`

Beschreibung : Typumwandlung von **`real`** nach **`int`**. Rundet dabei **`x`** ab bzw. auf.
Achtung : Die Schwelle ist bei 0.5 bzw. 0.51!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.round(2.49);
val it = 2 : int
- Real.round(2.5);
val it = 2 : int
- Real.round(2.51);
val it = 3 : int
-
```

`sameSign(x,y)` — Int.sameSign(x,y), Real.sameSign(x,y)

Signatur : `val sameSign : (int * int)|(real * real) -> bool`

Beschreibung : Prüft, ob **`x`** und **`y`** das gleiche „Vorzeichen“ haben. Gilt für ganze und reelle Zahlen gleich.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.sameSign(2, 3);
val it = true : bool
- Int.sameSign(~2, 3);
val it = false : bool
- Real.sameSign(2.0, 3.0);
val it = true : bool
- Real.sameSign(~2.0, 3.0);
val it = false : bool
-
```

sign(x) — Int.sign(x), Real.sign(x)

Signatur : `val sign : int|real -> int`

Beschreibung : Liefert das „Vorzeichen“ von **x** (~1, 0, oder 1). Gilt für ganze und reelle Zahlen gleich.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.sign(4);
val it = 1 : int
- Int.sign(0);
val it = 0 : int
- Int.sign(~6);
val it = ~1 : int
- Real.sign(4.3);
val it = 1 : int
- Real.sign(~3.6);
val it = ~1 : int
-
```

sin(x) — Math.sin(x)

Signatur : `val sin : real -> real`

Beschreibung : Berechnet Sinus von **x**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.sin(0.0);
val it = 0.0 : real
-
```

size(string) — String.size(string)

Signatur : `val size : string -> int`

Beschreibung : Liefert die Länge der Zeichenkette `string` als `int` Wert zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.size("Informatik");
val it = 10 : int
-
```

`sqrt(x)` — Math.sqrt(x)

Signatur : `val sqrt : real -> real`

Beschreibung : Berechnet die Quadratwurzel von **`x`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.sqrt(16.0);
val it = 4.0 : real
- Math.sqrt(15.45);
val it = 3.93064880141 : real
-
```

`str(zeichen)` — String.str(zeichen)

Signatur : `val str : char -> string`

Beschreibung : Typumwandlung eines Zeichens zu einer Zeichenkette

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.str(#"a");
val it = "a" : string
- String.str(#"l");
val it = "l" : string
-
```

`sub(string,x)` — String.sub(string,x)

Signatur : `val sub : (string * int) -> char`

Beschreibung : Liefert das **`x`**-te Zeichen (Typ `char`) der Zeichenkette `string` zurück.
Achtung: Die Zählung der Position beginnt mit 0!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.sub("Informatik", 3);
val it = #"o" : char
- String.sub("Informatik", 5);
val it = #"m" : char
-
```

substring(string,x,n) String.substring(string,x,n)

Signatur : `val substring : (string * int * int) -> string`

Beschreibung : Liefert **n** Zeichen der Zeichenkette **string** zurück, angefangen bei der Position **x**.
Achtung: Die Zählung der Position beginnt mit 0!

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- String.substring("Informatik", 0, 4);
val it = "Info" : string
- String.substring("Informatik", 5, 5);
val it = "matik" : string
-
```

succ(char) Char.succ(char)

Signatur : `val succ : char -> char`

Beschreibung : Liefert das nächste Zeichen nach **char** zurück (ASCII - Tabelle).

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.succ(#"a");
val it = #"b" : char
- Char.succ(#"m");
val it = #"n" : char
-
```

take(liste,n) List.take(liste,n)

Signatur : `val take : ('a list * int) -> 'a`

Beschreibung : **take** liefert die ersten *n* Elemente der Liste (**liste**) zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.take ([1,2,3,4,5], 3);
val it = [1,2,3] : int list
-
```

`tan(x)` — Math.tan(x)

Signatur : `val tan : real -> real`

Beschreibung : Berechnet Tangens von **`x`**.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Math.tan(0.0);
val it = 0.0 : real
- Math.tan(Math.sin(0.0) / Math.cos(0.0));
val it = 0.0 : real
-
```

`tl(liste)` — List.tl(liste)

Signatur : `val tl : 'a list -> 'a list`

Beschreibung : **`tl`** liefert die Liste (**`liste`**) ohne das erste Elemente zurück.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- List.tl ([1,2,3,4,5]);
val it = [2,3,4,5] : int list
-
```

`toLower(char)` — Char.toLower(char)

Signatur : `val toLower : char -> char`

Beschreibung : **`toLower`** wandelt **`char`** in Kleinschreibung um.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.toLower(#"A");
val it = #"a" : char
-
```

`toString(x)` — Int.toString(x), Real.toString(x), Bool.toString(x)

Signatur : `val toString : int|real|bool -> string`

Beschreibung : **`Int.toString`** führt für **`x`** eine Typumwandlung vom `int` zu `string` durch.
`Real.toString` führt für **`x`** eine Typumwandlung vom `real` zu `string` durch.
`Bool.toString` führt für **`x`** eine Typumwandlung vom `bool` zu `string` durch.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Int.toString(5);
val it = "5" : string
- Real.toString(5.2);
val it = "5.2" : string
- Bool.toString(true);
val it = "true" : string
-
```

`toUpper(char)` — Char.toUpper(char)

Signatur : `val toUpper : char -> char`

Beschreibung : **`toUpper`** wandelt **`char`** in Großschreibung um.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Char.toUpper(#"a");
val it = #"A" : char
-
```

`trunc(x)` — Real.trunc(x)

Signatur : `val trunc : real -> int`

Beschreibung : Typumwandlung von **`real`** nach **`int`**. Rundet dabei **`x`** in Richtung Null ab.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- Real.trunc(5.1);
val it = 5 : int
- Real.trunc(5.99);
val it = 5 : int
-
```

use (datei)

Signatur : `val use : string -> unit`

Beschreibung : Laden (Einlesen) der Datei (**`datei`**) in SML.

Beispiel :

```
Standard ML of New Jersey, Version 110.0.6, October 31, 1999
val use = fn : string -> unit
- use "test.sml";
[opening test.sml]
val it = () : unit
-
```