BASIC FUNCTIONS

In typing any function, there must be no space between the function name and the left parenthesis or bracket.

Square root:	*sqrt(x)*	$x \geq 0$,
Natural Log:	*log(x)*	$x > 0$,
Exponential:	*exp(x)*	$e^x < 10^{64}$ (use exp(1) for e),
Sine, cosine:	*sin(x), cos(x)*	$\vert x \vert < 100$, radian measure.

MATH computes the true value rounded to nine significant digits in most cases. Care is also taken to hit certain "magic values" (familiar argument values) exactly; i.e., computational algorithms are modified at these values.

Argument:	*arg(x,y)* is the angle of the point x,y in radians;
	arg(0,0) = 0 by definition;
	arg(-1,0) = π
Arcsine:	$\sin^{-1}(x)$ = *arg[sqrt(1-x*2),x]*.
Arc-cosine:	$\cos^{-1}(x)$ = *arg[x,sqrt(1-x*2)]*.
Arctan:	$\tan^{-1}(x)$ = *arg[1,x]*.

See page 63 for degree-radian conversion.

```
Type sqrt(2), log(10), exp(1), sin(3.1415).
     sqrt(2) =          1.41421356
     log(10) =          2.30258509
      exp(1) =          2.71828183
 sin(3.1415) =          9.26535898·10*(-5)

Type exp(log(13.1)).
exp(log(13.1)) =      13.1

c=1/log(10)
Let L(x) = c·log(x).                               Log base 10.
Let T(x) = arg(1,x).                               Arctangent.
Let S(x) = arg(sqrt[1-x*2],x).                     Arcsine.

Type L(10), L(100), T(0), T(10000), S(1), S(0).
        L(10) =          .999999999
       L(100) =         2
         T(0) =         0
     T(10000) =         1.57069632
         S(1) =         1.57079633
         S(0) =         0
```

Signum:	$sgn(x)$ =-1,0,+1	as x<0,x=0,x>0,
Integer part:	$ip(x)$	carries the sign of x,
Fraction part:	$fp(x)$	carries the sign of x,
Digit part:	$dp(x)$	carries the sign of x,
Exponent part:	$xp(x)$	carries the sign of the power of 10.

Note that

$$x = ip(x)+fp(x),$$
$$x = dp(x)\cdot 10*xp(x).$$

Absolute value: |...|.

Set x = -123.4567.

Type x,sgn(x),ip(x),fp(x),dp(x),xp(x),|x|.

```
      x =      -123.4567
 sgn(x) =         -1
  ip(x) =      -123
  fp(x) =          -.4567
  dp(x) =         -1.234567
  xp(x) =          2
    |x| =       123.4567
```

Type x=ip(x)+fp(x), x=dp(x)·10*xp(x).

```
x=ip(x)+fp(x) =        true
x=dp(x)·10*xp(x) =         true
```

SPECIAL FUNCTIONS

In defining the special MATH functions, the notation x=a(b)c:f(x) means that f(x), any function or expression involving x and including the constant function, is to be evaluated for x=a,a+b,a+2b,...,c, where a,b,c need not be integers (indeed they can be any expressions), and it is not necessary that a+nb=c (See page 18). The letter x is a dummy variable in the sense that the value of this same letter, if already stored (say by *Set*) will not be affected.

Maximum: *max(a,b,c,d)* or *max(x = a(b)c:f(x))*.
(Gives the largest in the list a,b,c,d or the largest f(x) in the range a(b)c.)

Minimum: *min(a,b,c,d)* or *min(x = a(b)c:f(x))*.

Sum: *sum(a,b,c,d)* or *sum(x = a(b)c:f(x))* means a+b+c+d or f(a)+f(a+b)+...+f(c), and, in particular,

$$\sum_{i=1}^{N} f(i) = sum(i = 1(1)N:f(i)).$$

```
x = 100.7

Type min[ x=.1(.1)4: x*2-2•x ]        The x within the min is a dummy.
min[ x=.1(.1)4: x*2-2•x ] = -1

Type x.
        x =        100.7

Type max(1,2,3,4,5,1/2,sin(5),.1,log(x)*2,.01).
max(1,2,3,4,5,1/2,sin(5),.1,log(x)*2,.01) = 21.2718889

Type min[ x=1(.1)10: (x-1.5)•(x-9.3) ].
min[ x=1(.1)10: (x-1.5)•(x-9.3) ] = -15.21
```

Product: *prod(a,b,c,d)* or *prod(x = a(b)c:f(x))* means a•b•c•d or f(a)•f(a+b)...f(c), and, in particular,

$$\prod_{i=1}^{N} f(i) = prod(i = 1(1)N:f(i)).$$

First: *first(x = a(b)c:* any mathematical or logical condition involving x). (Gives the first, and only the first, value of x in the range, for which the condition is satisfied.)

```
Type sum(1,2,3,4,5).
sum(1,2,3,4,5) =       15
Type sum(i=1(1)10: i*2 ).
sum(i=1(1)10: i*2 ) = 385

Type prod(x=1(1)6:x).
prod(x=1(1)6:x) =    720                        Factorial.

Type first(x=1,2,3,4,3.5,4.8,7+8,-1*3: x>2).
first(x=1,2,3,4,3.5,4.8,7+8,-1*3: x>2) = 3

Type min[ x=1(.1)10: (x-1.5)•(x-9.3) ].
min[ x=1(.1)10: (x-1.5)•(x-9.3) ] = -15.21

Type first[ x=1(.1)10: (x-1.5)•(x-9.3) = -15.21 ].
first[ x=1(.1)10: (x-1.5)•(x-9.3) = -15.21 ] = 5.4

Type first(x=1,2,3,4,5: x=2.5 ).
first(x=1,2,3,4,5: x=2.5 ) = ???
```

LOGICAL FUNCTIONS (EXPRESSIONS)

The words *not, and, or*(inclusive) can be used freely to form functions or expressions and are used extensively in conditions (*if* clauses, conditional expressions, *first* function). These words are used with letters or propositions having a truth value and not with letters having numerical values. Given a mathematical expression or logical proposition P that is either true or false, MATH can be asked to *Type P.*. MATH will respond *P = true* or *P = false*.

Precedence is (1) *not*, (2) *and*, (3) *or* in evaluating logical expressions. MATH would interpret "A or B and not C or D" as "A or (B and (not C)) or D." It seems wise to make your desires clear by using parentheses in constructing the expression.

```
1.1 Type a  or  b and not  c  or  d.
1.2 Type (a or b) and not (c or  d).
b=false
c=true
d=false
Do part  1 for a= true, false.
a  or  b and not  c  or  d =          true
(a or b) and not (c or  d) =          false
a  or  b and not  c  or  d =          false
(a or b) and not (c or  d) =          false

Type 1=1, 2<.03 and 5>25 or not 17≥17.
         1=1 =         true
2<.03 and 5>25 or not 17≥17 =         false

Type false=false=true.
false=false=true =         false
Type (false=false)=true.
(false=false)=true =       true
```

MATH makes a careful distinction between decimal values and logical values in its internal storage. It carries this distinction to the user by using the words *true* and *false* when a letter or expression has a logical value. Since it is common practice to use the decimal value 1 for true and 0 for false, MATH provides conversion between the two value forms by the translation value function $tv(x)$:

$tv(P) = 1$	for P = *true*, P a satisfied condition,
$tv(P) = 0$	for P = *false*, P not satisfied,
$tv(x)$ = *true*	for $x \neq 0$, x a number,
$tv(x)$ = *false*	for $x = 0$.

We also have

$\lvert P\rvert = 1$	for P = *true*,
$\lvert P\rvert = 0$	for P = *false*.

```
Set x= true.
Type tv(x).                                   Value true stored at x.
        tv(x) =             1
1.1 Type tv(x),|x|.
Do step 1.1 for x=5, 0, -.3, true, false.
        tv(x) =          true
          |x| =             5
        tv(x) =         false
          |x| =             0
        tv(x) =          true
          |x| =            .3
        tv(x) =             1
          |x| =             1
        tv(x) =             0
          |x| =             0
Type tv(2>3).
     tv(2>3) =              0                 Transient calculation.
Set p = 2>3.
Type p,tv(p),2>3.
            p =         false
        tv(p) =             0                 Value false stored at p.
          2>3 =         false
```

CONJUNCTION/DISJUNCTION

The function *conj(i = 1(1)n:P(i)) = true*, if P(1) and P(2) and ...P(n) are all *true*. Otherwise the function *conj(i = 1(1)n:P(i)) = false*.

The function *disj(i = 1(1)n:P(i)) = true*, if at least one of P(1),P(2),...,P(n) is *true*; otherwise it is *false*.

```
a=3
b=5
Type conj[ 4>a≠10, a•b≥a+b, b>4 or a>4 ].
conj[ 4>a≠10, a•b≥a+b, b>4 or a>4 ] =        true
```

List of arguments.

```
Type disj[ a-b>0, b+3•a=7, a*2•b≥25 ].
disj[ a-b>0, b+3•a=7, a*2•b≥25 ] =       true
```

Range of values.

```
1.1 Type conj[ i=1(1)b: a<i≤c ].
1.2 Type disj( x=1(1)b: a≤x≤c ).

Do part 1 for c=4,1.
conj[ i=1(1)b: a<i≤c ] =        false
disj( x=1(1)b: a≤x≤c ) =        true
conj[ i=1(1)b: a<i≤c ] =        false
disj( x=1(1)b: a≤x≤c ) =        false
```

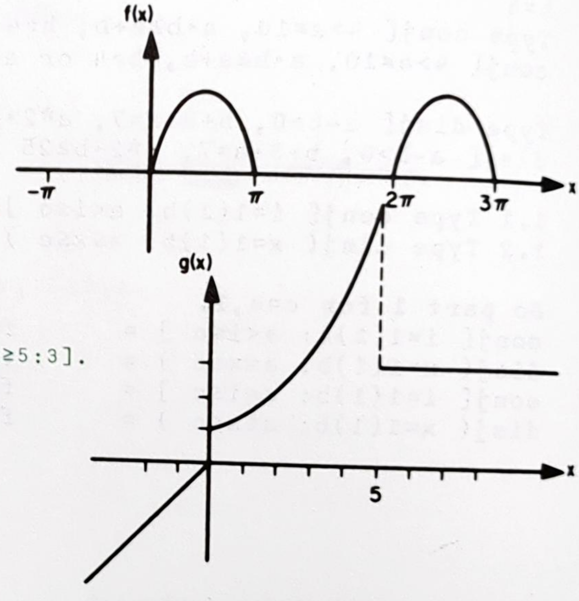
CONDITIONAL FUNCTIONS (EXPRESSIONS)

The notation *(A:b;C:d;e)* is read as follows:

If the mathematical condition/logical proposition A is satisfied/true, then use the value or expression b. If A is *false* and C is *true*, use d. If A and C are *false*, use e (which need not be preceded by a condition). Length of string is limited only by line capacity. Reading from left to right, the first true condition will determine the expression used. Conditional expressions can be used anywhere.

Functions can be defined freely by

Let f(x) = (A:b(x);C:d(x);e).




```
Type [1=2:1;2] + (1≠2:3;4).
    [2] + (3) =              5
Let f(x) = (sgn(sin[x])=-1:0; sin(x)).
Type f(-1), f(2), f(4).
        f(-1) =              0
         f(2) =               .909297427
         f(4) =              0

Let g(x) = [x<0:x; 0<x<5:exp(x/2.5); x≥5:3].
1.1 Type x, g(x) in form 1.
Form 1:
 ____        ____
Do part 1 for x=-1(2)3,4,7.
-1        -1.00
 1         1.49
 3         3.32
 4         4.95
 7         3.00

Do part 1 for x=0.
Error at step 1.1 (in formula g):   [x<0:x; 0<x<5:exp(x/2.5); x≥5:3] = ???
```

Note that g(x) is undefined at x=0.

RECURSIVE FUNCTIONS

Speaking loosely, a function is defined recursively if it appears in its own definition. If f(x) is the function to be defined, and g(x,y) is a given function, then f(x) is defined recursively by f(x) = g(x,f(x-1)).

Each time a function is evaluated by MATH, space is required to hold temporarily the progress of the execution including partial results needed to complete the calculation.

If we say *Let f(x) = x•f(x-1).*, and then *Type f(2).*, MATH will try f(2) = 2•f(1) = 2•1•f(0) = 2•1•0•f(-1) =....MATH will not stop when 2•1•0•f(-1) is reached because the computation, in his view, is not completed. Since space is needed for these partial results, MATH types *I ran out of space..* Some condition is always needed in conjunction with a recursive definition.

```
Let f(x) = x•f(x-1).
Type f(3).
Revoked. I ran out of space (in formula f).

Let f(x) = [ x=0:1; x•f(x-1) ].          If x=0, f(x)=1; otherwise x•f(x-1).
Type f(6).
        f(6) =        720
Type f(0).
        f(0) =          1

Let f(x) = ( x=0:1; fp(x)=0:prod(i=1(1)x:i) ).   f(x)=x! (factorial).
Type f(0), f(6).
        f(0) =          1
        f(6) =        720

Type f(5.4).
Error in formula f:   ( x=0:1; fp(x)=0:prod(i=1(1)x:i) ) = ???
```