QBasic Numeric Functions

ABS Function

Syntax: `n = ABS(number)`
ABS returns the absolute value of a number. In normal mathematics, the absolute value is represented by vertical bars. Basically, this function chops the negative sign off of a number.
Example:
```
PRINT ABS(5); ABS(-5); ABS(-2.16) 'Returns 5, 5, and 2.16
```

SQR Function

Syntax: `n = SQR(number)`
SQR calculates the square root of a number. The number must be non-negative, because QBasic cannot handle complex numbers (if you don't know about these, don't worry about it).
Example:
```
PRINT SQR(4); SQR(16); SQR(.25) 'Returns 2, 4, and .5
```

SIN, COS, TAN, and ATN Functions

Syntax:
```
n = SIN(radians)
n = COS(radians)
n = TAN(radians)
n = ATN(number)
```
These four functions are QBasic's trigonometric functions. SIN, COS, and TAN perform the sine, cosine, and tangent functions. ATN performs the arc tangent or inverse tangent. QBasic's trig functions all expect angle measures to be given in radians. The formula to convert degrees to radians is *radians* = *degrees* * (180 / *pi*), and radians to degrees is *degrees* = *radians* * (*pi* / 180). *pi* is about 3.14159265358979, and can be calculated by ATN(1) * 4. Of course, trying to calculate undefined values such as TAN(pi / 2) (tangent of 90 degrees) will cause an error.
Example:
```
pi# = ATN(1) * 4 'calculate pi to decent accuracy
PRINT SIN(pi / 2); COS(pi / 2) 'Returns 1 and 0
PRINT TAN(pi) 'Returns 0
```

LOG and EXP Functions

Syntax:
```
n = LOG(number)
n = EXP(number)
```
LOG and EXP are QBasic's logarithmic functions. LOG, despite its meaning in normal mathematics, returns the natural (base e) log of a number instead of the base 10 log. EXP raises e to the *number* power. e is the base of the natural log, which is about 2.18281828. Obviously, EXP can be used to calculate e.
Example:
```
e# = EXP(1) 'Returns e ^ 1 = e, about 2.18281828
PRINT LOG(4) 'Returns 1.386294
```

INT and FIX Functions

Syntax:

```
n% = INT(num)
n% = FIX(num)
```

INT and FIX both return a integral value for a number. **Do not confuse this with rounding!** Rounding is done by CINT and CLNG (see below). INT returns the largest integer less than or equal to the number. FIX chops off any fractional component of the number. This sounds confusing, and it is. Look at the examples for help.

Example:

```
PRINT FIX(7.49); FIX(7.51) 'Returns 7 and 7
PRINT INT(5.3), INT(-5.3) 'Returns 5 and -6
```

CINT and CLNG Functions

Syntax:

```
n% = CINT(num)
n& = CLNG(num)
```

CINT and CLNG perform numeric rounding. The only difference between the two is that CINT returns integers, while CLNG handles long integers. That means very large and very negative numbers must be used with CLNG. Like we learned in school, decimal values below 5 are rounded down, 5 and above are round up.

Example:

```
PRINT CINT(5.49); CINT(5.51) 'Returns 5 and 6
PRINT CLNG(654320.623) 'Returns 654321
```

CSNG, CDBL Functions

Syntax:

```
n! = CSNG(num)
n# = CDBL(num)
```

CSNG and CDBL convert a value to either single- or double-precision. These functions aren't used very much. In most cases, QBasic can automatically round a higher-precision value into a lower-precision variable. You may want to use it to force one integral number into single or double to avoid the ever-so-annoying Overflow error (see example). This stems from the fact that calculations are done in the lowest possible precision -- calculations using only integers are treated as integers, calculations using long integers and integers is treated as a long. Forcing a value to become single or double makes the calculation the same, and can avoid the error.

Example:

```
PRINT CSNG(ATN(1) * 4) 'Returns 3.14159
biggie% = 30000 'Integer limit is about 32767
PRINT 2 * CSNG(biggie%) 'Returns 60000
PRINT 2 * biggie% 'Overflow -- 60000 is too big to be an integer
```

LEN Function

Syntax: `n% = LEN(s$)`

LEN returns the number of characters in a string. If the string is empty (that is, ""), the function obviously returns 0.

Example: `PRINT LEN("Hello, world!") 'Returns 13`

VAL Function

Syntax: `n = VAL(s$)`

VAL extracts a number from a string. It starts at the beginning and reads the number. When it encounters a non-numeric character, it stops. VAL can also convert hexadecimal and octal strings into their decimal (base 10) equivalent if you use the &H or &O prefix (see example). VAL is fundamentally the converse of the STR$ function.

Example:

```
PRINT VAL("123 Main St.") 'Returns 123
h$ = HEX$(255) 'Returns FF
PRINT VAL("&H" + h$) 'Returns 255
```

ASC Function

Syntax: `n% = ASC(char$)`

The ASC function is basically the converse of the CHR$ function. It returns the ASCII code for *char$*. If there is more than one character in *char$*, it returns the ASCII code for the first character.

Example: `PRINT ASC(" ") 'Returns 32 (space character)`