TableForm()

```
TableForm(list)
```

The **TableForm()** procedure prints the contents of a list in the form of a table. Each member in the list is printed on its own line, and this sometimes makes the contents of the list easier to read:

```
In> testList := [2,4,6,8,10,12,14,16,18,20]
Result: [2,4,6,8,10,12,14,16,18,20]

In> TableForm(testList)
Result: [2,4,6,8,10,12,14,16,18,20]
Side Effects:
2
4
6
8
10
12
14
16
18
20
```

Incrementing And Decrementing Variables With The ++ And -- Operators

Up until this point we have been adding 1 to a variable with code in the form of **x := (x + 1)** and subtracting 1 from a variable with code in the form of **x := (x - 1)**. Another name for **adding** 1 to a variable is **incrementing** it and **decrementing** a variable means to **subtract** 1 from it. Now that you have had some experience with these longer forms, it is time to show you shorter versions of them.

Incrementing Variables With The ++ Operator

The number 1 can be added to a variable by simply placing the ++ operator after it like this:

```
In> x := 1
Result: 1

In> x++
Result: 2

In> x
Result: 2
```

Here is a program that uses the ++ operator to increment a loop index variable:

```
%mathpiper

index := 1;

While(index <=? 10)
{
    Echo(index);

    index++; //The ++ operator increments the index variable.
}

%/mathpiper

    %output,preserve="false"
      Result: True

      Side Effects:
      1
      2
      3
      4
      5
      6
      7
      8
      9
      10
.   %/output
```

Decrementing Variables With The -- Operator

The number 1 can be subtracted from a variable by simply placing the -- operator after it like this:

```
In> x := 1
Result: 1

In> x--
Result: 0

In> x
Result: 0
```

Here is a program that uses the -- operator to decrement a loop index variable:

```
%mathpiper

index := 10;
```

```
While(index >=? 1)
{
    Echo(index);

    index--; //The -- operator decrements the index variable.
}
```

```
%/mathpiper

    %output,preserve="false"
      Result: True

      Side Effects:
      10
      9
      8
      7
      6
      5
      4
      3
      2
      1
.   %/output
```

The +:= And -:= Operators

The +:= and -:= operators are similar to the ++ and -- operators, except their increment values can be specified:

```
In> x := 1
Result: 1

In> x +:= 5
Result: 6

In> x -:= 2
Result: 4
```

The For() Looping Procedure

The For() procedure provides an easy way to create loops that use an index variable. This is the calling format for the For() procedure:

```
For(initialization, predicate, changeIndex) body
```

The parameter named "initialization" is an expression that is usually used to assign an initial value to

the index variable. The parameter named "predicate" is an expression that is evaluated before the body is evaluated. If this "predicate" evaluates to True, then the body is evaluated. If "predicate" evaluates to False, the body is not evaluated, and the For() procedure finishes. The parameter named "changeIndex" is used to increase or decrease the value that is assigned to the index variable.

The following code uses a For() procedure to print the integers from 1 to 10 inclusive:

```
%mathpiper

For(index := 1, index <=? 10, index++)
{
    Echo(index);
}

%/mathpiper

    %output,preserve="false"
      Result: True

      Side Effects:
      1
      2
      3
      4
      5
      6
      7
      8
      9
      10
.   %/output
```

The Break() procedure

The **Break()** procedure is used to end a loop early and here is its calling format:

```
Break()
```

The following program has a While loop that is configured to loop 10 times. However, when the loop counter variable **index** reaches 5, the Break() procedure is called and this causes the loop to end early:

```
%mathpiper

index := 1;
```

```
While(index <=? 10)
{
    Echo(index);

    If(index =? 5)
    {
       Break();
    }

    index++;
}
```

```
%/mathpiper

    %output,preserve="false"
      Result: True

      Side Effects:
      1
      2
      3
      4
      5
.   %/output
```

When a Break() procedure is used to end a loop, it is called "**breaking out**" of the loop. Notice that only the numbers 1-5 are printed in this program.

The Continue() procedure

The **Continue()** procedure is similar to the Break() procedure, except that instead of ending the loop, it simply causes it to **skip the remainder of the loop for the current loop iteration**. Here is the Continue() procedure's calling format:

```
Continue()
```

The following program uses a While loop that is configured to print the integers from 0 to 8. However, the Continue() procedure is used to skip the execution of the Echo() procedure when the loop indexing variable **index** is equal to 5:

```
%mathpiper

index := 0;
```

```
While(index <? 8)
{
    index++;

    If(index =? 5)
    {
       Continue();
    }

    Echo(index);
}
```

```
%/mathpiper

    %output,preserve="false"
      Result: True

      Side Effects:
      1
      2
      3
      4
      6
      7
      8
.   %/output
```

Notice that the number 5 is not printed when this program is executed.

The AskUser(), TellUser(), and ToAtom() Procedures

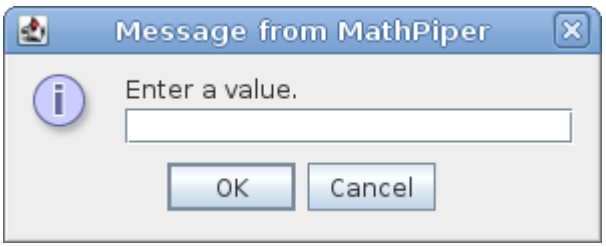
MathPiperIDE worksheet folds and the MathPiper console are designed to enable a user to directly enter and edit **input** and view it without having to use special input and output oriented procedures. However, sometimes it is useful to use procedures to obtain input from the user and output information to the user and MathPiper() has the **AskUser()** and **TellUser()** procedures for this purpose.

AskUser()

The AskUser() procedure obtains information from the user and here is its calling format:

```
AskUser(message)
```

The "message" argument is a string that contains instructions for what kind of input the user should enter. When the AskUser() procedure is called, a graphic dialog window is displayed that contains the message string and a text field into which the user can enter input. The dialog window looks like this:



After entering the input, the user presses the "OK" button, or the <Enter> key on the keyboard, and the input that was entered into the text field is returned by the procedure. For example, the following code asks the user to enter a value and then this value is assigned to inputValue:

```
In> inputValue := AskUser("Enter a number.")
Result: "42"

In> inputValue
Result: "42"
```

When the dialog window was displayed, the number **42** was entered and the number was returned by the procedure as the string "42". If you want to treat the input value as a **number** instead of a string, the ToAtom() procedure can be used to do this:

```
In> ToAtom("42")
```

```
Result: 42
```

The ToAtom() procedure can be used to turn a string into an unbound variable symbol:

```
In> ToAtom("a")
Result: a
```

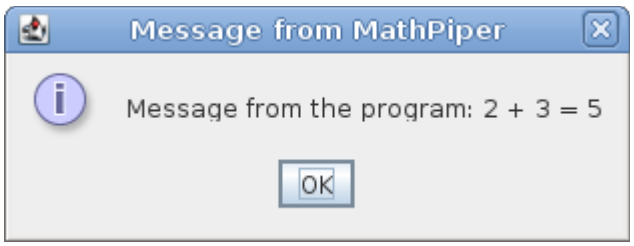
TellUser()

The TellUser() procedure displays a graphic dialog window that contains output information for the user to view and here is its calling format:

```
TellUser(message)
```

The "message" argument is simply a string that contains the output to be displayed to the user. For example, the following code sends the message "Message from the program: 2 + 3 = 5" to the user:

```
In> TellUser("Message from the program: 2 + 3 = " + (2 + 3) )
Result: True
```



Using AskUser() and TellUser() Together In a Loop

The AskUser() and TellUser() procedures can be used separately, but they are also often used together inside of a Repeat() loop. The following program is an infinite loop that repeatedly asks the user to enter an integer and then it tells the user which integer was entered:

```
%mathpiper

While(True)
{
    time := Time() inputValue := AskUser("Enter an integer (or q to quit).");

    If(inputValue =? "q" |? inputValue =? "Q")
    {
        Break();
    }
```

```
    integerValue := ToAtom(inputValue);

    If(!? Integer?(integerValue))
    {
        TellUser("You must enter an integer.");
        Continue();
    }

    TellUser("The integer you entered is " + integerValue + "." + Nl() +
    "It took you " + RoundTo(time,3) + " seconds to enter the integer." );
}

%/mathpiper
```

If the user enters a lower case q or upper case Q, the program stops looping by executing a Break() procedure. If the user presses the "cancel" button in the input dialog window, the program will also stop looping. If the user enters input that is not an integer, the program will tell the user that they must enter an integer.

If the user enters an integer, the program tells the user which integer they entered, and it also tells them how long it took to enter it.

Nested Loops

Now that you have seen how to solve problems with single loops, it is time to discuss what can be done when a loop is placed inside of another loop. A loop that is placed **inside** of another loop it is called a **nested loop** and this nesting can be extended to numerous levels if needed. This means that loop 1 can have loop 2 placed inside of it, loop 2 can have loop 3 placed inside of it, loop 3 can have loop 4 placed inside of it, and so on.

Nesting loops allows the programmer to accomplish an enormous amount of work with very little typing.

Generate All The Combinations That Can Be Entered Into A Two Digit Wheel Lock Using A Nested Loop

The following program generates all the combinations that can be entered into a two digit wheel lock. It uses a nested loop to accomplish this with the "**inside**" nested loop being used to generate **one's place** digits and the "**outside**" loop being used to generate **ten's place** digits.

%mathpiper

```
combinations := [];

outLoopIndex := 0;

While(outLoopIndex <=? 9)
{
```

```
    inLoopIndex := 0;

    While(inLoopIndex <=? 9)
    {
        Append!(combinations, [outLoopIndex, inLoopIndex]);

        inLoopIndex++;
    }

    outLoopIndex++;
}

TableForm(combinations);

%/mathpiper


    %output,preserve="false"
      Result: True

      Side Effects:
      [0,0]
      [0,1]
      [0,2]
      [0,3]
      [0,4]
      [0,5]
      [0,6]
        .
        .  //The middle of the list has not been shown.
        .
      [9,3]
      [9,4]
      [9,5]
      [9,6]
      [9,7]
      [9,8]
      [9,9]
.   %/output
```

The relationship between the outside loop and the inside loop is interesting because each time the **outside loop cycles once**, the **inside loop cycles 10 times**. Study this program carefully because nested loops can be used to solve a wide range of problems and therefore understanding how they work is important.

Introduction To The Plotter Plugin

The PlotterPoint() Procedure

MathPiper is also able to interact directly with Plotter using the **PlotterPoint()** procedure. These two procedures are covered in the following sections. **(Note: the Plotter plugin needs to have been opened at least once after MathPiperIDE has been launched before this procedure will work.)**

The **PlotterPoint()** procedure plots a point on the Plotter drawing pad and its calling format is as follows:

```
PlotterPoint(label, xCoordinate, yCoordinate)
```

The argument "label" is a string and it indicates what the label of the point should be. **The label name must start with an upper case letter and it cannot have any spaces in it.** The arguments "xCoordinate" and "yCoordinate" specify the point's coordinates. For example, the following line of code will **plot a point** labeled "**A**" on the drawing pad at location **2,3**:

```
In> PlotterPoint("A",2,3)
Result: java.lang.Boolean
```

Plotting 5 Random Points On The Drawing Pad

The following program uses **RandomInteger()** and **PlotterPoint()** to plot **5 random points** on the Plotter drawing pad:

```
%mathpiper

PlotterClear();

index := 1;

While(index <=? 5)
{
    xCoordinate := RandomInteger(8);

    yCoordinate := RandomInteger(8);

    PlotterPoint("P" + index, xCoordinate, yCoordinate);

    index++;
}

%/mathpiper
```

Notice that in this program the variable **index** is turned into a string and then concatenated with the

string **"A"** using the + operator. This is done in order to produce a sequence of unique labels such as A1, A2, A3, etc. (**Note: the labels of mathematical objects must start with a capital letter and not have any spaces in them.**).