

Lesson 1: Understanding Variables

What we learned so far has shown only that we are capable of displaying information. The PRINT statement directs output to the selected device (usually a window) and places items there. That's a fancy way of saying that PRINT works for Macintosh windows, printers, and files.

Another important facet of programming involves variables. A variable is a name given to a place in the computer where information is stored. Think of it as a postal address, except that we use names like Counter instead of P.O. Box 123.

Integers: The most common type of variable is an integer. An integer variable points to a mailbox large enough to hold 16 bits. In math terms, it is a number between -32768 and +32767. When an integer is assigned, it can use the explicit variable type of a percent sign or the percent sign can be omitted.

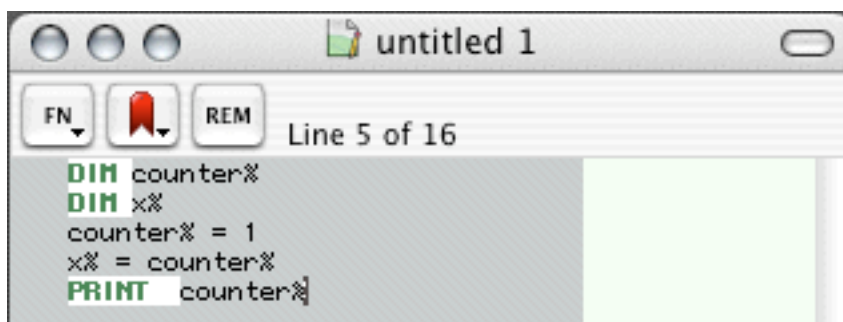
`DIM counter` is the same as... `DIM counter%` is the same as... `DIM counter AS INTEGER`

DIM means that you want space set aside to hold a value and that you wish to refer to that piece of memory with the name, *counter%*. In our examples, we will use the optional syntax of "%" to show that our stored value is an integer.

Assigning variables: We need to be able to put something into our named variable mailbox. This is accomplished with an equal sign ("="). Information always travels from the right side of an equal sign to the left side.

In this example, the value 1 is placed in the memory mailbox that we refer to as *counter%*. We can use this same methodology to transfer information from one variable to another. Enter this code in the edit window and run the program.

Can you predict the results?



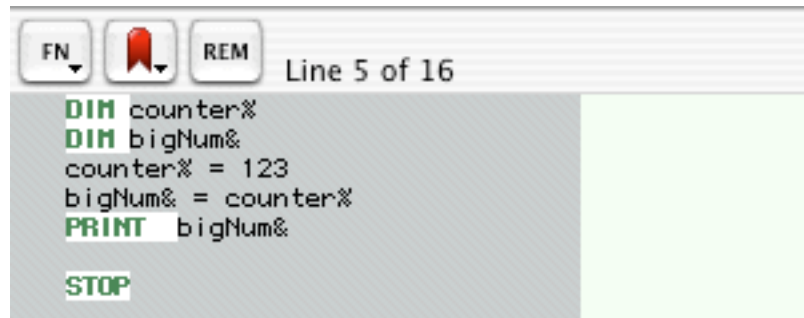
```
FN  [red bookmark] REM
Line 5 of 16
DIM counter%
DIM x%
counter% = 1
x% = counter%
PRINT counter%
```

If you guessed that the output window would display "1", you're on target. If not, reexamine the code until you understand what makes everything work.

Long integers: Another common variable type is the long integer. This memory mailbox has enough room to store a value that is +/- 2 billion. An integer and a long integer can only be used to store whole numbers. You can use it to store 23, but not 3.14. A long integer is designated with an ampersand ("&").

FB^4 can automatically translate from one type of variable to another. If you move the value of an integer into a long integer memory mailbox, FB^4 automatically adjusts it to fit. Enter the following code and attempt to predict the outcome. Assign new values to counter% to see how it works.

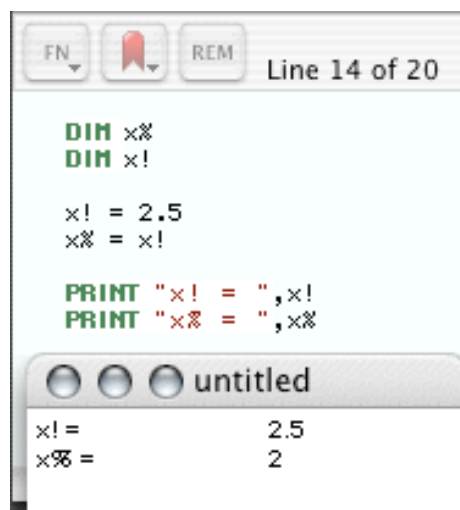
Lesson 1: Understanding Variables



```
FN
REM Line 5 of 16
DIM counter%
DIM bigNum%
counter% = 123
bigNum% = counter%
PRINT bigNum%
STOP
```

Floating Point: Floating point variables hold real numbers with fractions, like: 2.5, 3.14, 25.54321. Floating point variables must end with an exclamation mark, (!).

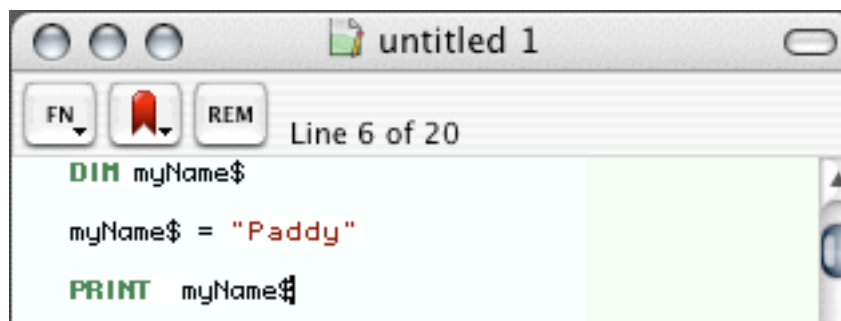
Note: If you assign an integer or long integer a floating point value, such as 2.5, the decimal portion will be lost and the variable will equal the integer portion, 2 in this case.



```
FN
REM Line 14 of 20
DIM x%
DIM x!
x! = 2.5
x% = x!
PRINT "x! = ", x!
PRINT "x% = ", x%
```

x! =	2.5
x% =	2

Strings: A string is a collection of characters. It's a lot like a sentence. You can tell that a memory mailbox holds a string when the dollar sign appears after the variable name. The example below shows how a string may be assigned and printed.

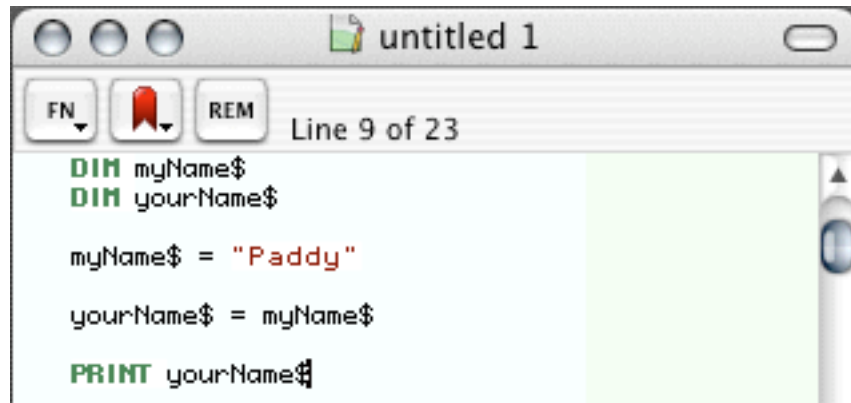


```
FN
REM Line 6 of 20
DIM myName$
myName$ = "Paddy"
PRINT myName$
```

Using String Variables: The variable, **myName\$**, is on the left side of the equal sign, so it will receive the information from the right side. In this example, the information being moved into the variable is limited by quote marks. The quotes tell FB^4 where the starting and ending points of the string reside. As with

Lesson 1: Understanding Variables

integer and long integer variables, you may pass information from one string variable to another. The following example passes information from the memory mailbox called myName\$ to the one called yourName\$. Can you predict what will be printed?



```
untitled 1
FN
REM
Line 9 of 23
DIM myName$
DIM yourName$

myName$ = "Paddy"

yourName$ = myName$

PRINT yourName$
```

Review: In this section, we have learned how to set aside space in memory mailboxes that we call variables. The size of that variable is determined by the character that follows its name. When we DIMension a variable, we specifically lay out the amount of space required to hold its information.

Integers (%) -32,766 to +32,767

Long integers (&) +/- 2 Billion

Strings (\$) Up to 255 characters, a sentence

Floating Point (!) decimals/fractions, up to 8 places of precision

We also learned how to send simple output to the window so that we might see the results of our work.

Exercises.

1.1 Type the following program, run it and explain what it does.

```
PRINT
PRINT "Hello"
PRINT "Welcome to FB^4 programming!"

STOP
```

1.2 Write a program that will give this result.

Lesson 1: Understanding Variables



1.3 Type in the programs and explain the similarities and differences.

a) `PRINT "one"` b) `PRINT "one";`
`PRINT "two"` `PRINT "two";`
`PRINT "three"` `PRINT "three"`
`STOP` `STOP`

1.4 Fix the mistakes in this program so that it runs.

```
DIM name$  
name = Paddy  
PRINT "Top o' the mornin', name$  
STOP
```

1.5 Fix this program and run it.

```
DIM a%  
DIM b  
a% = 250  
b = 100000  
c& = a%+b  
? "a = " a%  
? "b = " b  
?  
?"a+b = "c&  
  
STOP
```

Lesson 1: Understanding Variables

Programming Problems

- 1.1 Write a program that will translate your height in feet and inches to feet. ($ft = ft + in/12$)
- 1.2 Write a program that will define variables for your name, age and height, and then print this information on the screen.