QBasic Tutorial

By Neil C. Obremski
obremski@nwrain.com
http://www.nwrain.com/~deadeye

Introduction:

This little tutorial will assume you already know how to get into QBasic, save files in QBasic, load files, and run programs in QBasic. A section may be added about QBasic's Editor if enough people have questions about it.

The code snippets in this tutorial are seperated from the actual "teaching" text by their font. Anything in Courier (`This is the courier font, it is mono-spaced which means all the characters have the same spacing. Its great for showing code snippets`) is a code snippet while anything in Arial (This is Arial, you'll see it the most) is the stuff you should read not type. I also use CompStyle for titles. Also note that all ouput is on the left margin while code snippets are usually tabbed in at least once.

Remember to practice every concept you learn. I give you specific chances to, they look like this (TRY IT). But I encourage you to mess around as much as possible so you can grasp everything more fully.

And start using the keyboard more than the mouse. In QBasic it is more efficient to use keyboard commands then to hop around with the mouse. Just remember that SHIFT+F5 is start the program, F5 is start from where you left off, F4 outputs the screen, etc.

Section 1: Basic Commands

Typing your programs in QBasic is as easy as typing a report on a word processor. In fact QBasic's editor is a word processor, but it doesn't process english and it doesn't have a spell checker. It processes BASIC commands and has an automatic line syntax checker (as you'll see everytime you mistype a command).

Pressing keys will result in characters appearing on the screen, space will push things over as well as the cursor, delete will remove characters, backspace will crush the character immediatly to its left, the arrows will navigate the cursor, and the rest is all coding!

This first section gives you three basic commands. Enjoy.

The **`END`** Command:

This command is typed at the point where you wish your program to end. It actually isn't required in your programs, but if not typed in you may not know where your program stops. In small uncomplicated programs, it can be avoided. But when your programs become complex it is a necessity. In this tutorial END will be used in every program and I encourage its use in all of yours.

The **`PRINT`** Command:

The PRINT command outputs text to your monitor. It doesn't output it to the printer, LPRINT, its "brother" command, is used for that. PRINT is used simply by typing 'PRINT' followed by what you wish to be printed. If you wish some text printed to the screen then it must be enclosed in quotes.

```
WRONG: PRINT Hello World
RIGHT: PRINT "Hello World"
```

However, numerical values don't need to be enclosed in quotes. But note that if you print a numerical value without the quotes it will add spaces around the output.

```
RIGHT: PRINT "123"
RIGHT: PRINT 123
```

The output for the previous two statements would be:

```
123
 123
```

This command automatically moves the cursor position down one row and back to the left edge of the screen. Therefore you can use PRINT to move the cursor down. You don't actually have to specify text with a PRINT statement.

```
PRINT "Hello World, this is line one"
PRINT "and this is line two"
```

Will yield an output of:

```
Hello World, this is line one
and this is line two
```

And the following...

```
PRINT "Hello World, this is line one"
PRINT
PRINT "and this is line three"
```

Will yield an output of:

```
Hello World, this is line one

and this is line three
```

However, if you add a semicolon (;) at the end of your print statement, the cursor position will remain on the same row after the text is printed.

```
PRINT "Hello World, this is line one";
PRINT ", and this is still line one"
```

Yields the output of:

```
Hello World, this is line one, and this is still line one
```

The semicolon is also used to print text along with the information stored in variables, but I will save that until section 2. The semicolon attaches additional text directly to the end of the last text that was printed, but you can also use a comma (,) which attaches the text at the next tab stop (approximately every 15 spaces). So if we replace the semicolon in the last code section with a comma, this is what will be printed:

```
Hello World, this is line one             , and this is still line one
^             ^             ^             ^             ^
```

The ^'s represent the existant tab stops. It just so happens that the end of the first statement ends exactly on a tab stop, otherwise the distance between it and the next statement would have probably been less. The semicolons and commas are useful when the statement you are typing in QBasic runs off the screen, but when the program is run its on one line. I.E., you can use them to use them to make your QBasic code neater. Say the following statement goes off the screen when you type it in QBasic (which it does).

```
PRINT "This is a long statement to show how you can make your code easier to
read"
```

Well, since it runs off the screen you have to scroll over to read the end. So instead of using a single PRINT statement, you could use two (if you so feel inclined).

```
PRINT "This is a long statement to show how you can make your";
PRINT "code easier to read"
```

Just remember that both semicolons and commas prevent the cursor from moving down to the next line: semicolons attach trailing text to the end of the last statement and commas to the next tab stop on the current line.

Just because you add a semicolon or comma at the end of your statement, it doesn't mean you have to type in another PRINT command below it. The following is perfectly legal.

```
PRINT "Hello World, line one"; ", still line one"
```

Outputs as:

```
Hello World, line one, still line one.
```

This is useful when adding variables to your PRINT statements (section 2). Semicolons and commas can be added anywhere as long as they are not inside the quotes.

```
WRONG: PRINT "Hello World, this is line one;" ", still line one"
RIGHT: PRINT "Hello World, this is line one"; ", still line one"
```

Of course, QBasic isn't going to allow you to type that wrong statement. It will automatically try to correct your mistake. So if you typed in the wrong statement above, QBasic will change it automatically to:

```
PRINT "Hello World, this is line one;"; ", still line one"
```

And your ouput would be:

```
Hello World, this is line one;, still line one
```

Since commas and semicolons can be put anywhere in print statements, try the following and see what comes out. Expirement a little, try printing your name in approximately the center of the screen.

```
PRINT ,,,"Hello World"
```

(TRY IT) Make a small program that prints your name, skips the next line, then prints your address. Use multiple PRINT commands to print your street address and/or your city-state-zip onto the same line. Run it then replace your semicolons with commas to see the difference in the output. Remember to space down a line, using a PRINT statement with nothing after it. The output of your program should look something like below:

```
Neil C. Obremski

2221 99th. Ct. S.
Tacoma, WA 98444
```

And then after replacing the semicolons with commas, mine looked like this:

```
Neil C. Obremski

2221 99th.     Ct. S.
Tacoma        , WA           98444
```

Make sure to add END at the bottom of your program.

The **CLS** Command:

Before beginning a program, it is usually best to clear the screen so your output is not garbled with past output. That is what the CLS command is used for. It clears the entire screen and sets the cursor position to the upper left of the screen.

(TRY IT) Make a program that clears the screen, then prints your favorite quote. For practice, use two PRINT statements but have the quote on only one line. So your ouput would look similar to.

```
Imagination is more important than knowledge - Albert Einstein
```

Section 2: Data

Data in QBasic is stored in CONSTANTS and VARIABLES. There are specific uses for each of these two very different data types. To be slightly technical, CONSTANTS are data that never changes, i.e. remains constant, while VARIABLES are data that have the ability to be changed, i.e. vary.

To put everything in visuals, a constant would be a box with no openings with the value of its contents written on the side with a permanent marker. A variable, however, would be a box with a lid that could be removed so that the information inside could be added to, changed, removed, or read.

The **CONSTANTS**:

You've been using constants all along up to this point. Constants are simply values that never change. For example, in one part of your program you may have the following:

```
PRINT "Please enter your name:"
```

Well, everything enclosed in those quotes is a constant. It never changes. Constants, therefore, are data that never changes. Its the same everytime the program is run. That statement will always be printed like that, it never changes. PI is a constant, because PI will always be 3.14. You don't have to make a variable for PI because it will always be that value. It never changes.

So if in a program you must multiply a value by 3, then 3 is a constant because it will always be 3 in that part of the program.

To use the PRINT command to output the value of constants you do the same as you always have.

```
PRINT "The number"; 7; "is my lucky number"
```

Has the output of:

```
The number 7 is my lucky number
```

The **VARIABLES**:

Variables contain data that has the ability to change during the course of the program. For example if you wanted to count the number of times a certain function was called you would use a variable, because it may be called a different number of times each time you run the program. You may not always run that function twice or three times, you may run it as many times as needed. Therefore you would need to store the value in a variable.

Qbasic keeps track of variables by their names. Just like us, variables are assigned names so that we can keep track of them. If everyone was just "a person", then identifying each person might pose problems. Not only that, but you'd get sick of hearing "hey you!". You also want to make sure the

variable name stays the same. If you have a variable representing how many wishes you had come true, and you called it 'numwish', Qbasic would not recognize it if you wanted to print the value of it when you refered to it as 'numwishes'.

Here are rules to what variable names can be.

- Variable names can not begin with a number (`'1num'` is not a valid variable name while `'num1'` is).
- Variable names can only include letters and periods (special characters used in identifing the TYPE of the variable are allowed, but only at the end of the name).
- Variable names CANNOT include spaces.
- Variable names CANNOT be the name of a QBasic command. You CANNOT have a variable name called PRINT, CLS, END, etc.

Here are some good guidlines when naming variables.

- Eight characters is plenty for a variable name.
- Make sure the name is simple, but a good representation of what the variable stands for.
- If you make all your variables with lowercase letters, it is easier to distinguish them from your code.

Here are some silly variable names:

```
This.Is.A.Number%
This.Is.My.Name$
```

Here are some good variable names:

```
num%
name$
```

To create a variable you simply use it and QBasic will set aside memory for it automatically.

```
PRINT var
```

That statement will automatically create a variable called 'var' if it hasn't been used before in the program. However, in large programs this gets very messy. In large programs it is better to tell QBasic what variables you are going to be using. In this way you can keep track of every variable. To *declare* a variable, use the `DIM` command, like so:

```
DIM var
```

To declare multiple variables, seperate their names by commas:

```
DIM var1, var2, var3
```

Because there are different types of data you will have different types of variables and different ways to identify them. This is where the complexity of variables really gets started. But to put it simply, you have two main types of variables, those that store numerical values (51, 893.5, 0.005, etc.) and those that store text ("Hello World", "A", "Neil Obremski", etc.).

Variables that store text are called strings, because they store a string of characters, and in memory they use a string of bytes to store each character. Generally strings take the most data (any string over 8 characters long will take more than any numerical variable you'll ever use) and cannot be used to solve mathematical equations. This makes sense of course, when have you ever heard multiply "Neil Obremski" times 5.

Variables that store numbers I will refer to now only as numerical variables because there are many different types of numerical variables. Numerical values take very little memory because their values only require 2-8 bytes. Numerical values are used for mathematical equations.

To use a variable of a specific type you can do one of two things: You can declare it (with `DIM`) as a specific type or you can add a symbol to the end of its name to signify its type.

String Variables:

Because strings are fairly simple in QBasic I will deal with them first. To declare a variable as a string you would type something like the following.

```
DIM mystring AS STRING
```

This tells QBasic to set aside space in memory for a variable that is a string. This is one way to make a string variable, the other way is to simply *use* it during program execution (messy, but effective).

```
mystring$ = "This is my string"
```

The dollar sign at the end of the variable ($) signifies that the variable to be made is a string variable. The values of string variables are ***always*** enclosed in quotes. To put a value into a string variable simply type the name of the variable, then equals, and then the value (enclosed in quotes because it is a string). Remember also that if you make the variable during the course of the program (like above) to remember to add the dollar sign at the end of its name, otherwise QBasic will assume the variable you are making is numerical and you will get an error. However, if you have declared the variable as a string at the beginning of the program (`DIM`) then remember NOT to add the dollar sign ($) or you will get an error there also. Don't tell me its screwy, because I already know! Below is two sets of example code.

```
mystring$ = "Hello World"
PRINT mystring$
```

----------[next code segment]----------

```
DIM mystring AS STRING
mystring = "Hello World"
PRINT mystring
```

Hopefully by this point you have a pretty good idea of the concept of what a string variable is. You may also have noted, that a CONSTANT such as "Hello World" is a string, but it is a CONSTANT string because it will always be "Hello World". "Hello World" and a variable containing that value will be printed the exact same way and yield the same output, but remember that if you enclose a variable in quotes it will print what it sees, the variable's name not its contents:

```
DIM mystring AS STRING
mystring = "Hello World"
PRINT mystring
PRINT "Hello World"
```

So to print a VARIABLE string:

```
WRONG: PRINT "mystring$"
RIGHT: PRINT mystring$
```

In recent discovery I also noticed that variables with the same name, regardless if they have the proper identifier. Look at the following code:

```
DIM mystring AS STRING

mystring$ = "This was created during execution"
mystring = "This was declared at the beginning"
PRINT mystring$
PRINT mystring
```

Even though one has an identifier and one doesn't QBasic considers them the same. So the output will be "`This was declared at the beginning`".

When refering to a variable remember to keep its name the same EVERYTIME you refer to it. For example, if you create a string variable, put a value into it, then try to print it but you typed the name wrong, QBasic will assume you're trying to make a new variable and create a new and BLANK string variable:

```
DIM mystring AS STRING

mystring = "Hello World"
PRINT mystrng
```

Because the previous was typed wrong, the output will be NOTHING. Always remember to type your variable names correctly. This is probably the number one error in all my QBasic programs, because you don't get any errors if you mistype a variable name. Type carefully!

To print a CONSTANT string and a string variable on the same line, you would use your trusty semicolon or comma, just like before:

```
PRINT "This is my name:"; myname$
```

Of course you can have multiple string VARIABLES and CONSTANT strings in the same print statement:

```
PRINT "This is my name:"; myname$; " And this is my nickname:"; nickname$
```

(TRY IT) Make a program that declares a string variable at the beginning of the program, and one during the course of the program. Put your name in one the first, and someone else's in another. Then type a print statement that uses both of them in a sentence. Mine looks like this:

```
My name is Neil Obremski.
```

Okay, enough of strings. I encourage you to keep expirementing because there are a great number of things possible with just strings alone. For example, try to make a program that declares a string variable at the beginning, puts your first name in that variable, then on a seperate line adds your last name to it, then print the variable. If you can't figure this out, I'll give you the answer, just ask. You may want to try figuring that one out after reading this next part on NUMERICAL VARIABLES!!!!!!

Numerical Variables:

There are more than one variable types that store numerical data. In a technical list we have integers, long integers, single precision floats, and double precision floats. You saw two main words, right? Integers and Floats. These are the two main types of numerical variables. Integers can only store whole numbers (LIKE: 7, 28, 195 / NOT: 1.5, 30.832, 38.2), while floats can store decimal values. One would wonder, then, why would you ever use integers. The truth is integers take less memory and usually take less time to compute in mathematical equations (this is changing with new processors). Below is a small description of each type of integer and float:

INTEGER: A 16-bit variable. If you know how bits work then you may recall, this means that it should be able to store values up to 65536 (65535 when not counting 0). Well, this would only be true if

integers in QBasic were unsigned, which means the value was always positive. However, all variables in QBasic are signed (have the ability to be positive or negative), therefore the maximum value an integer can have is 32767, and the minimum of course -32767, because they need an extra bit to show if the variable is positive or negative, i.e. leaving 15 bits.

LONG INTEGER: A 32-bit variable. It can store values in excess of 2 billion and below negative 2 billion. You will rarely have a need for such capacity.

SINGLE PRECISION FLOAT: A 32-bit variable with 7-digit precision. This means 7 digits of the variable are precise, any more than that gets averaged to the last digit.

DOUBLE PRECISION FLOAT: A 64-bit variable with 15-digit precision. This means that 15 digits of the variable are precise.

What does all that mean? Well, normal integers can only store values up to 32767 or down to -32767. That is the maximum value you can put into them, otherwise QBasic will give you an "Overflow" error.

```
ERROR: a% = 70007
RIGHT: a% = 7000
```

And the precision of floats? What does that mean? Well look at the following code.

```
x! = 0.123456789
y! = 12345.6789
z! = 123456789.0
```

QBasic automatically changes what I just typed in to the following:

```
x! = .123456789#
y! = 12345.6789#
z! = 123456789#
```

The output of this is:

```
.1234568
12345.68
1.234568E+08
```

Notice how QBasic averages the last digits (past precision). So always remember to keep your values for floats within their precision range (7 on single precision, 15 on double). The exclamation point (!) at the end of the variable names in the above examples tells QBasic that the variable is a single precision float. Read on for more information on numerical variable identifiers.

You create a numerical variable just as you would a string variable, except you use different identifiers tagged onto the end, and you '`DIM`' them differently. Below is a list of identifiers for each of the variable types we will probably be using:

```
% - Integer
& - Long Integer
! - Single Precision Float
# - Double Precision Float
```

The following is a list of the keyword to use to '`DIM`' each at the beginning of the program:

```
INTEGER - Integer
LONG - Long Integer
SINGLE - Single Precision Float
DOUBLE - Double Precision Float
```

So if we wish to declare an integer variable, we would do the following:

```
DIM myint AS INTEGER
```

And if we wanted it made when the program is run:

```
myint% = 9
```

Remember to create a variable during run-time we simply use it. Using a variable could mean anything. You could give it a value (like above), you could output it (`PRINT myint%`), you could use it in a mathematical operation, etc. All you have to do is use it and QBasic will create it. But remember that this is a sloppy way to program, and I recommend `DIM`ing your variables at the BEGINNING of your program (you can `DIM` in the middle of your program, but that is also very sloppy).

To put values into numerical variables you do it the same way (with string variables), except **you don't use quotes**. Which is logical because you don't multiply "5" by 9, instead you would multiply 5 by 9 (no quotes, not a string, not text). Because when QBasic sees quotes around something it automatically thinks its a constant string.

```
WRONG: num% = "7"
RIGHT: num% = 7
```

Below is an example that creates all four types of numerical variables (using a single `DIM` statement at the beginning), and puts values into each of them.

```
DIM myint AS INTEGER, mylong AS LONG, mysing AS SINGLE, mydoub AS DOUBLE

myint = 7
mylong = 1509375902
mysing = 109.5732
mydoub = 1.00000000071839
```

And of course to simple make numerical variables during the program:

```
myint% = 7
mylong& = 1509375902
mysing! = 109.5732
mydoub# = 10094.00000000071839
```

Remember that if you `DIM` a variable as a certain type at the beginning then you don't need to append the variable type's identifier at the end of its name when you use it:

```
DIM myint AS INTEGER
myint% = 3
```

The previous works fine, but the % is not needed because QBasic already knows it is an integer variable. However, also remember that had you not put the `DIM` statement in, you *would* need the % identifier or QBasic will assume it is a single precision float (QBasic's variable default).

And then you print them like any string variable, because when you print the content of variables you always use the same rules:

```
WRONG: PRINT "myint%"
RIGHT: PRINT myint%
```

The "WRONG" in the previous example would run fine, but your output would be "myint%" instead of the actual value of myint%.

Since printing numerical variables is like printing string variables, it is easy to integrate them into print statements containing constant strings:

```
PRINT "This is my integer:"; myint%;
PRINT ", This is my long integer:"; mylong&
PRINT "This is my single precision float:"; mysing!;
PRINT ", This is my double precision float:"; mydoub#
```

And knowing this, it is easy to print them along with string variables and other numerics:

```
PRINT mystring$; myint%; mysing!
```

If you have expiremented a little, you may have noticed that if you just use a variable with no identifier it is assumed as a number. If you just make a variable during your program with no identifier, it is automatically a single precision float.

```
avar = 89
```

The previous is automatically a single precision float variable even if it is a whole number, QBasic decides that you may need decimal values later so that's why it does that.

Again, remember that a variable with an identifier compared to a variable with the same name but no identifier, is considered the same one:

```
DIM myint AS INTEGER

myint = 7
myint% = 8
PRINT myint
PRINT myint%
```

In the previous snippet of code, the output will be "8" followed on the next line by "8". However, if you DIM a variable as an INTEGER and then put a FLOAT identifier on the end, you will get a "Duplicated Definition" error because QBasic knows the variable is an INTEGER and yet you are trying to make a FLOAT with the same name. So the following will give you an error:

```
DIM myint AS INTEGER
myint! = 9.8
```

(TRY IT) Make a program that declares an integer variable, a single precision float variable, and a string variable at the beginning of the program. Put your favorite number in the integer variable, put your favorite number divided by 100 in the float variable, and put your name in the string variable. Then print them all on the same line in a sentence. Mine looks like this (part of mine is on the second line, try to make yours on one line by printing something other than what I have):

```
My name is Neil Obremski, my favorite number is 7 , my favorite number divided
by 100 is 0.07.
```

In conclusion remember that you cannot change a constant, therefore the following will not work:

```
4 = 4
"Hello World" = "Hello World"
```

Even though the values you are trying to impress are the same you cannot touch the data of a constant because it never changes. Yet you can use constants like variables and variables like constants.

```
var1% = var2% + 4
var1% = 6 + var2%
PRINT var1%
PRINT 10
```

If you grasp that concept then move onto the next section.

Section 3: Operators

Just putting CONSTANT numbers into numerical VARIABLES isn't going to do any good if you can't mess around with them. That's where operators come into the picture. And these operators aren't like the kind you get on the phone, they're the kind that add, divide, subtract, multiply, etc.

The four main operators I will describe are +, -, /, and *.

`(+)` – used to add one value to another
`(-)` – used to subtract one value from another
`(/)` – used to divide one value by another
`(*)` – used to multiply one value by another

I'm sure you already know how to add, subtract, multiply, and divide; and in QBasic its done the same way. The only difference is that you have to put the value of the answer somewhere. You can either put in a variable or output it to the screen. The following prints the answer to multiply two constants.

```
PRINT 2 * 2
```

Of course this isn't really practical because you can't change the answer at any time. So what you do is put the answer into a variable using the equal sign operator (=). You've already used this, so you probably won't have any problem using it.

```
myvar% = 2 * 2
PRINT myvar%
```

Just like usual you can do as many different operations as you want in one line:

```
myvar% = 2 * 2 / 2 + 2 - 2
```

The previous is completely legal. And can you guess what the answer is? The first thing QBasic does when it sees that is do the multiplication and division from left to right: Multiply two by two (4), then divide that by two (2). Next it does the addition and subtraction from left to right: Add two to our previous sub-answer (4), Subtract two from that (2). The resulting answer is 2.

To prevent QBasic from doing one thing before another you can use paranthesis. Everything in parenthesis is done first:

```
myvar% 2 * 2 / (2 + 2) - 2
```

So QBasic reads the previous code snippet and does the operations inside the parenthesis first: Add two plus two (4). Then it does multiplication and division from left to right: Multiply two by two (4) and divide that by the sub-answer in the parenthesis (4) to get a sub-answer of one. Next it does addition and subtraction from left to right: Subtract two from our sub-answer of one (-1). The resulting answer is negative one. Most of these things you should have learned from algebra and the order of operations. If you don't know the order of operations just think MY DEAR AUNT SALLY. Multiply,

Divide, Add, Subtract: (M)y (D)ear (A)unt (S)ally. And of course, parenthesis first, but I don't remember the limyric for that. There are other operators and they usually come before My Dear Aunt Sally, but these simple ones will do for this tutorial.

Using numerical variables in mathematical operations is just like using constants, just put in the variable name instead of the constant:

```
var1% = 7
var2% = 3
myint% = var1% + var2%
PRINT myint%
myint% = 7 + 3
```

Will yield:

```
10
10
```

When you use variables int mathmatical equations, the resulting answer depends on what variable you're putting it into. If you are simply outputting the answer to the screen then you will get the exact result. However, if your answer is decimal and you put it in an integer variable then the value is rounded to the next whole number (up if the decimal is 0.5 or above and down if it is below).

```
intvar% = 8.5 / 5.8
PRINT 8.5 / 5.8
PRINT intvar%
```

Will yield:

```
1.465517
1
```

While:

```
intvar% = 9 / 5
PRINT 9 / 5
PRINT intvar%
```

Will yield:

```
1.8
2
```

Adding Strings:

You can also use the addition operator `(+)` to add one string to another. The following prints two constant strings added together:

```
PRINT "Hello " + "World"
```

Adding two string variables uses the same concept:

```
str1$ = "Hello "
str2$ = "World"
PRINT str1$ + str2$
str3$ = str1$ + str2$
PRINT str3$
```

The previous will yield the following output:

```
Hello World
Hello World
```

Even though the additions operator can be use to connect two strings, you cannot use it to get a numerical answer:

```
PRINT "2" + "2"
```

The previous will give you an output of:

```
22
```

Because all QBasic thinks your doing is connecting two constant strings for output as one constant string. And any other operator will yield a "Type Mismatch" error:

```
PRINT "2" * "2"
PRINT "2" / "2"
PRINT "2" - "2"
```

And of course, just like numerical operations, you can add as many strings together in one line as you wish. The following works fine:

```
PRINT "H" + "e" + "l" + "l" + "o" + " World"
```

(TRY IT) Make a program that declares a string variable and an integer variable at the beginning of the program. Put your name in the string variable and your zip code in the integer variable. Now print your name, then your street address on the next line, then your city-state-zip on the next line. Make sure you print your zip code using the integer variable you declared. Mine looks like this:

```
Neil C. Obremski
2221 99th Ct. S.
Tacoma, WA 98444
```

CLOSING NOTE: It is better to declare your variables as their various types, at the beginning of the program. In this way you don't have to worry about what identifier to add to the end of the name.

Section 4: Etiquette

Its really nice when you can easily read your code, especially in large programs. This is where programming etiquette comes into play. Some people refer to it as a programmers style, but I kinda think that everyone should make their code as readable as possible without compromising anything. Therefore I call it Etiquette.

All this is, is tabbing in your code, adding spaces, and remarks/comments to your code to make it neat and easily read. QBasic doesn't really care if you add extra tabs in your code, or extra blank lines, so you don't have to worry about screwing up your code. Take the following example:

```
CLS
DIM myname AS STRING
myname = "Neil C. Obremski"
DIM progname AS STRING
progname = "My Messy Program By " + myname
PRINT "This program is called: "; progname
PRINT "It was made by: "; myname
```

```
DIM myint AS INTEGER
myint = 7
PRINT "The Programmer's Favorite Number is"; myint
DIM myfloat AS SINGLE
myfloat = myint / 7
PRINT "The Programmer's Favorite Number divided by 100 is"; myfloat
END
```

In larger programs messiness is more of a problem, but I think most of us can agree that the following is a lot nicer to look at:

```
'                       /---------------------\                        '
'                       |   My Neat Program   |                        '
'                       | By Neil C. Obremski |                        '
'                       |September 23rd, 1997 |                        '
'                       \---------------------/                        '
DIM myint AS INTEGER, myfloat AS SINGLE
DIM myname AS STRING, progname AS STRING

CLS
      ' --- -- - Fill Variables With Values - -- --- '
      myname = "Neil C. Obremski"
      progname = "My Neat Program By " + myname
      myint = 7
      myfloat = myint / 7

      ' --- -- - Show Output - -- --- '
      PRINT "This program is called: "; progname
      PRINT "It was made by: "; myname
      PRINT "The Programmer's Favorite Number is"; myint
      PRINT "The Programmer's Favorite Number divided by 100 is"; myfloat

      ' --- -- - Now we're done - Goodbye - -- --- '
END
```

Now, the program looks a lot longer, but it still does the same thing. It's just easier to read. Later, with larger programs, you may find that a couple extra spaces here and there make the program a lot more comprehensible at first glance.

You may noticed I used single quotation marks, known to everyone as apostrophe's. In QBasic they're used for comments or remarks. Everything past one of these is completely ignored:

```
      PRINT myname$     ' Outputs my name to the screen
```

You can also use the REM command, but this has to be at the beginning of the line and it looks bulky:

```
      REM This is my program
      REM Its by me
```

There is no set rules where to use comments, remarks, extra spaces, tabs, or anything. Just use your best judement. There are freaks out there who pride themselves on how neat or cryptic their code is. Its all a matter of preference and style. But keep in mind those that will be reading your code (especially you, looking back at it).

Section 5: User Input

If all your program does is run its happy little way by itself, figuring everything out by itself, and doing its little duty, its not going to be very interactive for the user. Unless the user is you the programmer. User Input is a very important part of a program because it allows the user to interact with the program. Of course you limit the interaction. This section is going to explain the INPUT statement as well as syntax.

The **INPUT** Command:

This command allows the user to type in letters or numbers at the cursor position. Pressing enter will put what they have typed in into the specified variable(s). If they type in the wrong type of data, a "Redo From Start" error will occur (not fatal error). Wrong type of data? Well, if they type in their name where it says their age then it won't know how to put that letter string into a numerical variable and give you that error. However, if it says their name and they type their age, then QBasic will assume that what they typed is correct and put what they typed in, into that string variable.

The syntax for this command is:

```
INPUT [;] ["prompt" {; | ,}] [variablelist]
```

What is syntax? Syntax is how the command is used, i.e. what parameters and variables it takes, etc. If you get a "Syntax Error!" it means you gave the statement the wrong number of parameters or the wrong type.

A lot of commands will show their syntax like above. In fact, if you go to the QBasic help and look up `INPUT` in the index, it will show you something very similar to the above syntax. What does all of it mean?

Well each thing in the square brackets, `[` and `]`, is optional. Of course, everything looks optional, but if you just type in INPUT without anything else, you will get an error, usually an "Expected" error. Where QBasic expected you to put something else after it, but you didn't. When you type the command you also don't put the square brackets around things:

```
WRONG: INPUT [;] [a$]
RIGHT: INPUT ; a$
```

So what does the rest of this silly stuff mean? I have numbered each of the parts and gave a paragraph explanation. Examples are also below each of the paragraphs:

```
INPUT [;] ["prompt" {; | ,}] [variablelist]
       ^      ^         ^             ^
       1      2         3             4
```

`[;]` 1.) If you have this in your input statement then the cursor will not move down a line after pressing enter to put the data into the variable. See the following example:

```
INPUT ; a$
PRINT "<- This is one the same line"
```

The PRINT statement following the INPUT statement will be on the same line.

(TRY IT) Type what I just typed above and run it. Press enter to put the data into the variable. Don't know what to type? Just type your name for now. Notice, how the cursor doesn't move down after you press enter.

`"prompt"` 2.) The prompt is a constant string that the user sees preceding the cursor position. So if you type the following:

```
INPUT "My Prompt>"; a$
```

The user will see `My Prompt>` followed by the cursor where it allows them to type in the data. This is useful because you don't have to type a `PRINT` statement. Look at the following two examples.

```
PRINT "My Prompt>";
INPUT a$
```

----------[next snippet]----------

```
INPUT "My Prompt>"; a$
```

Remember that a semicolon at the end of a `PRINT` statement keeps the cursor from moving down. Both code snippets do exactly the same thing, but one is on a single line and the other takes two. But you also must remember that the prompt can ONLY be a constant string. You cannot put a string variable in its place, otherwise QBasic will assume you are trying to input data into that variable:

```
WRONG: INPUT myprompt$; a$
RIGHT: INPUT "My Prompt>"; a$
```

When you type in the wrong statement and then try to run it you will get an error because QBasic doesn't understand what you are trying to do. So remember, the prompt can ONLY be a single constant string.

`{; | ,}` 3.) You may think by looking at this that you have three things: a semicolon (`;`), a pipe or bar (`|`), and a comma (`,`). However, the pipe simply means OR. Therefore you have two choices: a semicolon or a comma. Notice they are within the same square brackets as the prompt, this means that if you have a prompt then you need to have either of these and if you have either of these you need a prompt. But what do each of these mean? Well below is the difference:

```
INPUT "Comma>", a$          Will yield ->     Comma>_
INPUT "Semicolon>"; a$      Will yield ->     Semicolon>? _
```

The underscore `(_)` represents the cursor position. Notice how the semicolon adds a question mark, a space, and then lets the user type input, while the comma adds nothing. Note, that without the prompt the default of a question mark followed by a space is used. To avoid this you can type an empty prompt plus a comma:

```
INPUT "", a$
```

`[variable list]` 4.) What is the variable list? By this time you may have figured it out, it is what variable(s) the input goes into. You have the option of more than one variable, but in that case you have to seperate the user input with commas:

```
INPUT a$, b$, c$
```

In the previous code snippet the user couldn't just type in his/her name and press enter. They HAVE to type in their first response, COMMA, their second response, etc. So if you were the user you would have to type something like:

```
? Neil, Caleb, Obremski
```

Where "`Neil`" would be put into "`a$`", "`Caleb`" into "`b$`", and "`Obremski`" into "`c$`". This is usually a hassle for the user so it would be more advisable to break up the input statement like so:

```
INPUT "First Name>", a$
```

```
INPUT "Middle Name/Initial>", b$
INPUT "Last Name>", c$
```

The previous code is a lot easier to understand and type in data for.

So what does all this mean? Well it means you can use `INPUT` to get data for your program. See the following example.

```
DIM yourname AS STRING
CLS
INPUT "What is your name"; yourname
PRINT "Ah! So your name is "; yourname ;"?  That's excellent!"
```

So, the output for me is:

```
What is your name? Neil
Ah! So your name is Neil?  That's excellent!
```

The first line declares "`yourname`" as a string variable. The second line clears the screen. The third line is important because it encompasses what this whole section is about. It inputs data into the "`yourname`" variable (string data) preceded by a prompt of "`What is your name`". In the code you don't see a question mark, but in the output you do. Why? Remember that the semicolon following the prompt adds a question mark followed by a space. Try replacing the semicolon with a comma and see what results.

The fourth line is easy. It prints a constant string immediatly followed by a string variable, followed again by another constant string. Since no semicolon is at the end of the print statement, the cursor moves down to the next line. But of course this makes no difference because the program is over. Here is a disection of that `INPUT` statement:

```
INPUT "What is your name"; yourname
  ^           ^          ^        ^
command     prompt     {; | ,}   variable
```

To input a number is just as easy as a string:

```
INPUT "Your age please (rude as it is to ask)>", age%
```

And of course you can use those variables just like any string:

```
PRINT "You're"; age%;" years old!  My my my... to be young again..."
```

(TRY IT) Make a program that inputs the user's name, hair color, and a favorite music group then prints out sentences including what the user has typed in. You decide variable arrangement and program layout.

My output looks like this:

```
What's your name buster? Neil
Hmm, so what color is your hair? Strawberry Blond
Enough!  What's one of your favorite music groups? Faith No More

Okay Mr. Neil!  So you say your hair is Strawberry Blond, eh?  I hope you're
right, oh faithful listener of Faith No More.
```

Section 6: Decisions

Decisions are a very important task for a program to accomplish. If your program accepts data, it should usually make decisions to make sure the data is correct. Specifically when I speak of decisions in QBasic I speak of the `IF THEN` statement and its counterparts `ELSE` and `END IF`.

The **`IF`** Statement:

Syntax:

```
IF [condition] THEN
      [code to execute if condition is true]
ELSE
      [code to execute if condition is false]
END IF
```

What is the `ELSE` and `END IF` and the stuff in between? Well, everything between the first line and `ELSE` and between `END IF` and `ELSE` is code. Code meaning you can put anything you've already learned in there. You can put `PRINT` statements, `INPUT` statements, `CLS`'s, etc. When and why is the code actually used by QBasic? Well, if the condition specified is TRUE then the code immediately following `THEN` is executed (or run; used by QBasic). If the condition is not true or FALSE then the code immediatly following the `ELSE` is executed.

Here is your introduction of conditions. In programming languages a condition that works out or is logical is TRUE. For example "4 = 2 + 2" is a condition that is TRUE. Why? Well, because as we all know (hopefully), 2 plus 2 IS four. If we had said "4 = 2 + 1" then *that* condition would be FALSE. Why would we ever have a condition like that? Well, for one thing we wouldn't have it in constants, we would use variables. And since variables don't always contain the same data, the condition has the possibility of being FALSE. Take for example, the following:

```
var1% = 4
var2% = 4

PRINT "Variable 1 ("; var1%; ") and Variable 2 ("; var2%; ") are ";

IF var1% = var2% THEN
      PRINT "EQUAL!"
ELSE
      PRINT "NOT EQUAL!"
END IF
```

Which will yield an output of:

```
Variable 1 ( 4 ) and Variable 2 ( 4 ) are EQUAL!
```

If you copied the exact code as it is, you would always get this output. But say for example you set `var1%` to 8 and `var2%` to 15, then the program's output would be otherwise, wouldn't it? But these all imply that the data is pre-determined. What if we use an `INPUT` statement to get the values of `var1%` and `var2%`, then we wouldn't know the output until we ran the program and typed in the numbers.

Of course this program wouldn't be very useful, as we always know visually how to compare to numbers. But if during the course of the program you needed to compare two variables this could prove very useful. Of course conditions don't always mean two values equalling each other. You also have the choice of greater than (>), less than (<), greater than or equal (>=), or less than or equal (<=). Intuitively you probably already know how to use these. You may recall from some distant math class having to add in one of these signs between to numbers of different size.

How do I use these? Well, use them just as you would an equal sign. We all know that 2 is less than 4 (`2 < 4`), therefore you could type the following and the condition would always be true:

```
IF 2 < 4 THEN PRINT "TWO IS LESS THAN FOUR!"
```

Ah, you noticed I only used a single line. I could have done the same thing in more lines:

```
IF 2 < 4 THEN
      PRINT "TWO IS LESS THAN FOUR!"
ELSE
ENDIF
```

But, notice all the wasted space. Since we don't even have code to execute if the condition is false we can omit it, therefore use something like:

```
IF 2 < 4 THEN
      PRINT "TWO IS LESS THAN FOUR!"
END IF
```

Again, there is three lines for something which really only does two things. So since we only have a single command we can just put it on the same line as the actual `IF` statement (following the `THEN`) and get rid of the `END IF`.

Many times you will not have a need for code if the condition is false. In these cases you can simply omit the `ELSE`. And many times you will have a need for only one command if the condition is TRUE. In these cases you can simply use one line. Also, if you have only one command for code if a condition is TRUE, you can combine that into one line (as long as the code for if the condition is TRUE is also one line):

```
IF var1% < var2% THEN
      PRINT "LESS THAN"
ELSE
      PRINT "GREATER THAN"
END IF
```

Can be put on one line as so:

```
IF var1% < var2% THEN PRINT "LESS THAN" ELSE PRINT "GREATER THAN"
```

And remember that if you have an `IF` statement that takes up a single line, do NOT add the `END IF`.

(TRY IT) Make a program that gets two numbers from the user (via `INPUT`) and prints which one is the larger and which one is the smaller. Have it print its answer on a single line (ooh tricky). This little program should encompass everything this tutorial has taught so far, good luck =).

Strings in Decisions:

And how could you compare strings? Well, the

Section 7: LABELs and GOTOs
Section 8: SUB-Programs
Section I: Example Programs

This section includes some example programs that perform a specific deed. They increase in complexity the farther through this section you go.

Title: Basic Program
Description: Clears the screen, prints a constant string, and ends.

```
CLS
      PRINT "Hello World!"
END
```

Title: Print Attachments
Description: Demonstrates the use of semicolons and commas, by using two print commands that use each.

```
CLS
      PRINT "Hello"; "World!"
      PRINT "Hello", "World!"
END
```

Section II: Common Problems

This section is dedicated to common problems so that you can figure out most of your "What did I do wrong!?"s. If you can answer these HELP ME's without reading the ANSWER and REMEMBER then you definitely know what you're doing!

HELP ME: It gives me a "Expected: Variable = Expression" error when I type this in:

```
my%var = 10
```

ANSWER: You have a percent sign in your variable name: remove it, or if you wish to make this an integer, put the percent sign at the end for the integer indentifier.

REMEMBER: Variable names CANNOT include special characters unless they are indentifiers. Identifiers MUST come at the end of the variable name.

Section III: Important Notes

- Everything in quotes will be used "AS IS", or WYSIWYG as some people like to say (what you see is what you get). If you wish to use a variable in any way do NOT put quotes around the variable name (QBasic will see `"myname$"` as the actual text *myname$*, while it will see `myname$` (no quotes) as whatever the content of that variable name is (`Neil Obremski` for example)).
- You can't put numerical values into constants. You can't, for example, say `4 = 2 + 2`. You have to either output the values or put them into a variable.