QBasic has a system of data type suffixes that can be used with variable names to determine data type:  
  
% is INTEGER  
& is LONG  
! is SINGLE  
# is DOUBLE  
$ is variable length STRING  
  
Because you are not using any DEFxxx statements, the default data type is SINGLE. If you had used DEFINT A-Z, for example, then the default data type would be INTEGER.  
  
By adding % to the end of the variable name, you specify it to be an INTEGER. As Clippy pointed out, wsFileNum% and wsFileNum! are two different variables.  
  
Depending on personal preference, you might wish to avoid data type suffixes to the extent possible. It is considered by some people (including me) good practice to explicitly declare all of your variables:  
  
DIM wsFileNum AS INTEGER  
  
The # on the OPEN statement is optional. It is not a data type suffix, it is a reference to the fact that you are specifying a file number.  
  
To answer your question, yes, the first example is correct. The second example would only be correct if you had used DIM wsFileNum AS INTEGER, or something like DEFINT A-Z. Another functionally equivalent example is:  
  
DIM comPort AS STRING  
DIM wsFileNum AS INTEGER  
  
wsFileNum = FREEFILE  
CONST comMode = "9600,N,8,1,CS6000,DS6000"  
comPort = "COM1:"  
OPEN comPort + comMode FOR RANDOM AS wsFileNum  
  
On QB64, I'd possibly make wsFileNum a LONG instead of an INTEGER.  
  
Regards,  
Michael

**Introduction to QuickBASIC  
(IBM-PC and Mac versions)**

[RETURN](http://terpconnect.umd.edu/~toh/Chem498C/Chem498C.html) to the CHEM 498C Home Page

**BASIC**, which stands for Beginner's All-purpose Symbolic Instruction Code, is an easily-learned computer language that was developed in the 1960's. It is still important because of its adoption by the influential industry giant MicroSoft as the basis for their Visual Basic language and of the macro language in version 5 of their Excel spreadsheet (VBA, Visual Basic for Applications). The brief introduction here is based on a simple BASIC dialect called QuickBASIC.

Go through each of the commands and sample codes described here to verify that they work as you expect. Then do the assignment at the end and submit printed program listings.

**1. Control commands and editing.**  
**IBM-PC (DOS).** Run QuickBasic by typing qb at the DOS prompt and press ENTER. This takes you to the editor window. When you write a program from scratch, you type in program lines, using the cursor keys (or the mouse if there is one on your computer) to navigate around in the text, the delete key to erase text. To save your program, press ALT-F and then S, type a file name and press RETURN. To load an existing program, press ALT-F and then O. This presents a list of files in the QuickBasic subdirectory. Press Tab to move to the file list, use the cursor keys to select the desired file and then press RETURN to load it. To execute (run) the program press Shift F5. F4 toggles between output window and editor screen. F6 toggles between command window and editor screen.

**Mac.**Launch QuickBasic by double-clicking on the QuickBasic icon. The screen shows three windows: the output window (where program-generated output goes), the editor window (where you type and edit programs), and the command window (where you can type and execute single commands instantly). The editor window works just any Mac (or Windows) text editor, with **Open** and **Save**commands, **Copy** and **Paste**, etc. To run a program, hold down the apple key and press R. To stop a running program, hold down the command (apple) key and type a period. To bring up the editor window, press (apple)-L. To print the program listing, select **Print** from the**File** menu.

**2. Text Screen printing. PRINT** A prints the value of the variable A on the screen and moves the cursor to the next line. Putting a semicolon at the end of the line keeps the cursor on the same line. The default output screen has 24 lines and 80 columns. **CLS** clears the text screen. **LOCATE** y,x moves the cursor to the line y, character position x.

**3. Keyboard input**. **INPUT** A$ pauses the program, displays a ? prompt, waits for any string of characters to be typed in, then when is pressed, assigns the string variable A$ to the string typed in. To read the keyboard on the fly, without pausing the program, use key$=**INKEY$**, which assigns the last key pressed to the string variable key$. This is useful in real-time control applications, to allow a program to continue to run while it checks for keypresses.

**4. Data Types and labels**. Variables can be given alphanumeric names, i.e. pH, volume1,volume2, etc., but reserved words (the names of BASIC commands and functions) can not be used for variable names. String variables have a $ added (i.e. reply$) and are limited to 255 characters. Integer variables have a % (i.e. reading%) and must be between -32767 and 32767. Real (floating point) numeric variables nave no suffix and have a range of 10±38. Arrays of reals, integers, or strings are indicated by (). Thus **DIM** A%(1000) dimensions (allocates space for) an array of 1000 integer numbers. **LET** Z%=A%(K) sets Z% equal to the Kth element of the integer array A%(). Arrays of reals and strings are handled the same way.

After a program has been run once, you can interrogate the current value of any of its variables by switching to the command window and typing ? followed by the name of the variable. This is a useful technique in troubleshooting.

**5. Functions and math statements.**BASIC uses the same sort of "computer math" notation as most other languages and spreadsheets, etc, with +, -, \*, and /, parentheses for grouping, ^ for exponents. The principal math functions are **SQR**() (square root), **EXP**(), **LOG**() (natural log), **SIN**(), **COS**(), **TAN**(), **RND**() (a random number between 0 and 1).

**6. Loops and Decisions.**To execute a number of statements a fixed number of times, use a FOR/NEXT loop:

**FOR** X=1 **TO** 10

**PRINT** X, **SIN**(X)

**NEXT** X

which causes the line or lines between the **FOR** and the **NEXT** to be executed repeatedly for every value of X from 1 to 10. To create a loop that executes until some condition occurs or while some conditions exists, use the**WHILE... WEND** construct. For example, this is useful for reading a text file of unknown length until all data are read:

**OPEN** "data.txt" **FOR INPUT AS** #1

k = 0

**WHILE NOT EOF**(1)

**INPUT** #1, Y(k)

k = k +1

**WEND**

The statements between the **WHILE** and the**WEND** are executed repeatedly as long as the condition **NOT EOF**(1) is true. In this case the condition is the boolean function **EOF**, which returns a *true* when the end of the file is reached. Conditions can also be boolean expressions such as x=1 or a>b, containing any combination of variables, constants or expressions related by = (equal), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), <> (not equal to). You can also combine conditions using **AND** or**OR**: e.g., X > 0 **AND** X < 100.

For simple two-way decisions, use the IF clause:

**IF** y<0 **THEN**

**PRINT** "Y is negative"

**ELSE**

**PRINT** "Y is not negative"

**END IF**

For multiple choice situations, use the CASE structure:

**SELECT CASE** K$

**CASE** "A","a"

**PRINT** "K$ was A"

**CASE** "B","b"

**PRINT** "K$ was B"

**CASE ELSE**

**PRINT** "K$ was neither A nor B"

**END SELECT**

To jump directly to another line in a program, use a **GOTO** aaaa statement and label the line to jump to with aaaa: placed first in the line. For example:

TryAgain: **INPUT** "Pick a number between 1 and 10", A

**IF** A < 1 **OR** A > 10 **THEN GOTO** TryAgain

**7. Subroutines and subprograms.** Subroutines and subprograms are sections of a program code that are set apart from the main program so they can be called from several places in the main program and which, after they are finished, return to the next line in the main program. They are most commonly used for code which is re-used several times in one program. A subroutine begins with a labeled line and ends with a **RETURN** statement. It is called with a **GOSUB** command. All of the variables in a subroutine are global, that is, they are shared with the main program. A subprogram, on the other hand, begins with a **SUB** statement and ends with an **END SUB**; it is called with a**CALL** statement. All of the variables in a subprogram are local, that is, they are independent of the variables in the main program. Information is passed between the subprogram and the main program by means of an argument list.

**8. Memory access and Input and Output ports.** Laboratory interfacing using add-in cards (e.g. analog-to-digital and digital-to-analog converters) usually requires direct access to absolute memory locations. On the IBM-PC, access to add-in boards is by means of special input and output instructions that refer to **I/O ports**. To *output*to an I/O port, you use **OUT** port,datum, where port is the output port number (range: 0-65535) and datum the single-byte integer data value (range: 0-255). To get *input*from a port, you use the **INP** function:datum = **INP**(port), where port is the input port number (range: 0-65535) and datum a single-byte integer data value read from the port (range: 0-255). The port number is usually determined by switch settings on the add-in card.

**9. Data files.** It is often necessary to save and recall data (e.g, a spectrum, chromatogram, titration curve, etc.) as a disk file. Such data are often stored in the computer's memory as an array. The simplest way to store data on the disk is as a *sequential text* file, which stores the ASCII text as they would be printed out on the screen or printer. Each data number can take anywhere from 2 bytes (e.g. an single-digit number follower by a carriage return) to 15 bytes (e.g. -1.4354557E-10) for a single-precision real. For example, this program creates a sequential text file named "SEQFILE.DAT" containing 100 x-y pairs where x = 1 to 100 and y = square root of x:

**OPEN** "SEQFILE.TXT" **FOR OUTPUT AS** #1

**FOR** j = 1 **TO** 100

**WRITE** #1, j, **SQR**(j)

**NEXT** j

**CLOSE** #1

This program reads and prints out the contents of the file "SEQFILE.DAT":

**OPEN** "SEQFILE.TXT" **FOR INPUT AS** #1

**FOR** j = 1 **TO** 100

**INPUT** #1, x, y

**PRINT** x, y

**NEXT** j

**CLOSE** #1

**10. Printing to the network printer from with a program.** On the Macintosh, program output may be directed to the currently-selected printer on the LocalTalk network by opening the printer like it were a file. At the beginning of the program, you assign a file channel to the printer:

**OPEN** "LPT1:PROMPT" **FOR OUTPUT AS** #1

Later in the program when you are ready to print, say, the variable Z$, use **PRINT** #1, Z$. At the end of the program, **CLOSE** #1. Use a channel number different from other open files.

**11. Graphics.**On an IBM-PC you have to use the **SCREEN** command to specify the screen mode. The old PC-XTs have GCS graphics, which calls for the use of the **SCREEN** 1. The graphics screen coordinates are 0 to 320 on the horizontal (X) axis and 0 to 200 on the vertical (Y) axis. The newer models have VGA graphics; use **SCREEN** 12. The graphics screen coordinates are 0 to 640 on the horizontal (X) axis and 0 to 480 on the vertical (Y) axis.

On the Macintosh, the SCREEN command is not used; the graphic screen coordinates depend on the size of the output window - typically 0 to 620 on the horizontal (X) axis and 0 to 430 on the vertical (Y) axis.

The origin (0,0) is on the upper left. **PSET** (X,Y) plots a point at X,Y. **LINE** (X1,Y1) - (X2,Y2) draws a line from coordinates X1,Y1 to X2,Y2. In DOS, attempting to plot outside of the allowed coordinate range results in an error message (but not on the Mac).

Axis labels and other text notations to a graphic screen are easily added by using the **LOCATE** and **PRINT** commands.

**12. String operations.** BASIC was the first language to have a set of string manipulation functions. Try these out so you understand their function.

**LEN** A$ returns the length of A$.

**LEFT$**(A$,1) returns the leftmost character of A$.

**RIGHT$**(A$,1) returns the rightmost character of A$.

**MID$**(A$,n,m) returns m characters of A$ starting with the the nth character. For example, if A$="TEMP 1200 C", **MID$**(A$,6,4) equals "1200".

**LET** A$=B$+C$ assigns A$ to the fusion of B$ and C$.

**STR$**(A) returns the decimal string equivalent of the numeric variable A.

**VAL**(A$) interprets A$ as a decimal number string and converts it into its numeric value.

**ASC**(A$) returns the ASCII value of the single character A$.

**CHR$**(A) returns the single character string whose ASCII value is A.

**HEX$**(A) returns a string containing the hexadecimal representation of the numeric variable A.

These functions are very useful in interfacing applications, for example to construct command strings for the automation of an RS-232-interfaced instrument, to process received data strings from such an instrument, to construct algorithmically generated filenames, and to process user input from the keyboard. Here's an example of the last application. Often, within a program, it's necessary to give the operator some way of changing specific variables. The following example displays the current value of the variable X1 and allows the user to change it by typing in a new value and pressing or to leave it alone by simply pressing .

**PRINT** "Value of variable X1 (currently ";X1;")";

**INPUT** A$

**IF** A$<>"" **THEN LET** X1=**VAL**(A$)

From a user interface point of view, it is better to enter numeric data this way, rather than using **INPUT** X1. The latter will give an error message and abort the program if any non-numeric entry is made, whereas **INPUT** A$ will accept any string, even an "empty" RETURN.**VAL** simply returns a zero for any non-numeric input, but this can be easily checked for by the program. The **VAL** function also strips off any trailing alphabetic characters (e.g. units, etc.)

**Assignment**

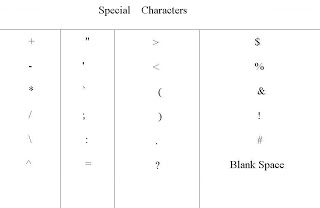
1. Write a program that calculates one complete cycle of a sine wave (from 0 to 2¹) and saves it as a sequential text file in x,y format (on the boot disk). Use a text editor to open and inspect the resulting data file. Submit a printed program listing.

2. Write a program that reads a sequential text file in X,Y format (up to 100 data pairs in length), determines the number of data pairs and the range of X-values and Y-values, and plots all of the X,Y pairs as points on the graphic screen so that Xmin is near the left of the screen, Xmax is near the right of the screen, Ymin is near the bottom of the screen, and Ymax is near the top of the screen. Draw and label the X- and Y axes. Test this program on the file generated in part 1. Submit a printed program listing.

This page is maintained by [Tom O'Haver](http://www.wam.umd.edu/~toh/toh.html), Department of Chemistry and Biochemistry, The University of Maryland at College Park. Comments, suggestions and questions should be directed to Prof. O'Haver at [to2@umail.umd.edu](mailto:to2@umail.umd.edu).

### [elements of qbasic](http://worldcomputerarticle.blogspot.com/2010/05/elements-of-qbasic.html)

7:56 PM | Posted by prabin | |

**INTRODUCTION**  
Every programming language consists of some basic elements which are required to make a program. The element required to construct a QBASIC program consists of a set of *characters, keywords, constants, variables, operators and expressions.*  
  
**1. CHARACTER SET**  
A set of characters that are allowed to use in QBASIC is known as the QBASIC Character Set. The QBASIC Character Set consists of alphabets (both small and capital), numbers (0 to 9) and special characters. These special characters have their own meaning and function. The table below shows the special characters used in QBASIC.  
  
[](http://3.bp.blogspot.com/_oYlT7Zn6ivI/S-oZ65bX0UI/AAAAAAAAAH0/nM0lIXu7hxQ/s1600/Special+Character+Set.JPG)  
QBASIC keywords and variables are formed by using the characters defined in the QBASIC Character Set.  
  
**2. KEYWORD**  
Keywords are those words which have special meanings in QBASIC. Keywords are formed by using characters of QBASIC Characters Set. Keywords are statements, commands, functions (built in functions) and names of operators. The keywords are also called Reserved Words. Some reserved words are CLS, REM, INPUT, LET, PRINT, FOR, DO, SELECT, MID$, ASC, SQR, LEN, LEFT$, TIME$ and INT.  
  
**3. CONSTANTS**  
Constants are the data or the values in a program that cannot be changed during the program execution. The data may be a letter, words, numbers, or special characters. A constant can be stored in a variable when it is required to use in more than one statement or expression. In QBASIC, these data/constants are grouped into two main categories. They are:  
a. Sting Constant  
b. Numeric Constant  
  
**a. String Constant:**  
Sting Constant is a letter, words, numbers, combination of letters with numbers or special characters enclosed in double quotes. Mathematical operations cannot be performed on String Constants.  
“B”, “APPLE”, “SYMBOL NO:10205”, “!!! Welcome to QBASIC World !!!”, etc. are some examples of Sting Constants.  
  
**b. Numeric Constant:**  
Numeric Constant refers to a number. A number with or without decimal point is a numeric constant. Thousand separators are not allowed to use in numeric constant. Numeric data should not be enclosed in double quotes. Mathematical operations and logical operations can be performed on the numeric constants. 101, 105.50, 720, 45603, etc. are some examples of numeric constants.  
  
Numeric Constants may be integer, long integer, single precision or double precision.

1. *Integer:* Integer is whole number between -32768 to 32767.
2. *Long Integer:* Long Integer is a large range of whole number.
3. *Single Precision:* Single Precision is seven digit or less than seven digit positive or negative number that contains decimal point. Single Precision can be in the exponential form using E or with a trailing exclamation point. (!). 564, 78.65, 1.2 E-06 and 12345.678! are some examples of Single Precision Constants.
4. *Double Precision****:*** Double Precision is 17 digit or less than 17 digit positive or negative numbers that contains decimal point. Double Precision can be in the exponential form using D or with trailing hash sing (#). 9999.99D-12, 2345.786# and 3456.78 are some examples of Double Precision Constants.

**4. VARIABLE**  
A program is written to perform certain tasks. A program needs data to produce information. A program can use data only when data are stored in the computer memory (RAM). In a computer memory, there may be many data. So, you need to tell the computer to use only those data which you want to use in a program. This is possible if you assign a name to the place where you have stored a data. In QBASIC, you can perform this task by using a variable. A variable is a place in the computer memory which has a name and stores data temporarily. Simply, you can say, a variable is an entity that stores data needed to be used in a program. Each program defines different number of variables. A value of a variable can be change during the execution of the program.  
  
There are mainly two types of variables. They are:  
i. String Variable  
ii. Numeric Variable  
  
A string variable stores sting data. Its types declaration sign is dollar ($). A numeric variable stores numeric data. A numeric variable can be Integer, Long Integer, Single Precision or Double Precision variables.

* An Integer variable can store only an integer number. Its type declaration sign is percentage (%).
* A Long integer variable can store a large range of whole number. Its type declaration sign is ampersand (&0).
* A Single Precision variable can store a whole number as well as number with decimal. Its type declaration sign is exclamation sign (!). You can also use it without declaration sign. It is the default numeric variable.
* A Double Precision variable also stores a whole number as well as number with decimal point. Its type declaration sign is hash (#).

**Rules for naming a variable**  
a. Variable names can have maximum of 40 characters.  
b. Variable names can have alphabets, numbers and decimal point.  
c. A Variable name must begin with a letter.  
d. A Variable name cannot begin with fn or FN alphabets. For example, fnames$, fnumetc.  
e. Variable names cannot be reserved words.  
f. Variable names may be ended with type declaration characters like $, %, &, !, and #.  
  
Naam$, Address$, Bookname$, GameName$, etc., are examples of Sting Varibales.  
Salary!, Age%, Mark, Number1, Number2, FirstNum, RollNumber, etc., are examples of Numeric Variables.  
  
**5. OPERATOR**  
Operators are symbols that indicate the type of operation QBASIC has to perform on the data or on the values of variables.  
  
There are four types of operators in QBASIC. They are Arithmetic Operators, Relational Operators, Logical Operators and Sting Operator.  
  
**a. Arithmetic Operators**  
Arithmetic Operators are used to perform mathematical calculations like addition, subtraction, division, multiplication and exponential. The following table shows arithmetic operators used in QBASIC.  
  
**Operation ------------ Operator ---------------- Example ------------------ Result**  
i. Addition ----------------- + ----------------------- 5+8 -------------------------- 13  
ii. Subtraction ----------- - ---------------------- 8-6 --------------------------- 2  
iii. Multiplication -------- \* ---------------------- 5\*4 --------------------------- 20  
iv. Division ---------------- / ------------------------ 8/2 -------------------------- 4  
v. Integer Division -------- \ ----------------------- 9\2 --------------------------- 4  
vi. Exponential ----------- ^ ----------------------- 4^3 ------------------------- 64  
vii. Modular Division --- Mod --------------------- 7 mod 3 ------------------------ 1  
  
  
**b. Relational Operators**  
Relational Operators are use to perform comparisons on two values of same type. A comparison of sting data with numeric data cannot be done. The comparison of sting data is done on the basis of ASCII value. The result of comparison is either true (non zero) or false (zero).  
The following table shows the relational operators used in QBASIC.  
**Operator ------------- Relation ------------------------------- Example**  
i. = --------------------- Equal to -------------------------------- A = B, A$ = B$  
ii. > -------------------- Greater than --------------------------- A > B, “CAT”>”RAT”  
iii. < ------------------- Less than ------------------------------- A < B, "cat" < "cat"  
iv. > = ---------------- Greater than or equal to ---------------- A > = B, X$ > = Y$  
v. < = ----------------- Less than or equal to ------------------- A < = B, X$ < = Y$  
vi. < > ---------------- Not equal ------------------------------ A$ < > B$, X <> Y.  
  
**c. Logical Operators**  
Logical Operators combine two or more relational expressions to evaluate a single value as True (Non Zero) or False (Zero). The result of evaluation is used to make decisions about the program flow. The commonly used logical operators in QBASIC are AND, OR and NOT.  
  
**i. AND Operator:**  
AND operator returns ‘True’ when all the results returned from individual relational expressions are ‘True’ otherwise it returns ‘False’.  
The AND Truth Table is given shown below.  
**Condition1 (P) ------------- Condition2 (Q) -------------- Result (P AND Q)**  
F ------------------------------- T ----------------------------------- F  
T ------------------------------- F ----------------------------------- F  
F ------------------------------- F ----------------------------------- F  
T ------------------------------- T ----------------------------------- T  
  
Note: A ‘T’ indicates a true value and a ‘F’ indicates a false value.  
  
**ii. OR Operator:**  
OR Operator return ‘True’ if any one of the relational expressions returns ‘True’. If all the relational expressions returns ‘False’ then only the combined result returned by OR operator will be ‘False’.  
  
The OR Truth table is as given below.  
**Condition 1 (A) ------------------ Condition2 (Q) --------------- Result (A or B)**  
F ------------------------------------------- T ---------------------------------- T  
T ------------------------------------------- F ---------------------------------- T  
T ------------------------------------------- T ---------------------------------- T  
F -------------------------------------------- F ---------------------------------- F  
  
**iii. NOT Operator:**  
NOT Operator operates on one operand and returns ‘True’ if the logical operation returns ‘False’. The NOT truth table is as given below.  
**Condition1 (A) --------------------------- Result (NOT A)**  
F ----------------------------------------------- T  
T ----------------------------------------------- F  
  
**d. String Operator**  
String Operator joins two or more than two sting data. The plus sign (+) is used as the String operator. The act of combining two stings is called concatenation. The following table shows the use of Sting Operator.  
**String Data (A$) -------------------- Sting data (B$) ----------------- A$ + B$**  
“Ram” ---------------------------------- “Thapa” ------------------------ Ram Thapa  
“50” ------------------------------------- “45” ----------------------------- 5045  
  
  
**6. EXPRESSIONS**  
An expression is the combination of operators, constants and variables that is evaluated to get a result. The result of the expression is either string data, numeric data or logical value (true or false) and can be stored in a variable. For example, the following are expressions in QBASIC.  
(A + B) > C  
A > = B + C  
u\* t + ½\*a\*t^2  
  
An arithmetic expression may contain more than one operator. While evaluating such expressions, a hierarchy is followed. The hierarchy in arithmetic operations is listed as given below:  
a. Exponentiation (^)  
b. Negation (-)  
c. Multiplication and division  
d. Integer division  
e. Modular division  
f. Addition and Subtraction  
  
The hierarchy in relational operations are =, >, <, <>, < =, and > = respectively. The hierarchy in logical operations are NOT, AND and OR.  
  
*NOTE:*

* When parenthesis is used, it changes the order of hierarchy. The operators inside the parenthesis are evaluated first. So, you can say QBASIC expression follows rule of PEDMAS where P, E, D, M, A and S stand for parenthesis, Exponentiation, Division, Multiplication, Addition, and Subtraction respectively.
* Algebraic expression cannot be used directly in programming. It must be converted into QBASIC expression.

**Algebraic Expression --------------------------------- BASIC Expression**  
A = L × B ------------------------------------------------- A = L \* B  
P = 2(L + B) ---------------------------------------------- P = 2\*(L + B)  
I = (P × T × R)/100 --------------------------------------- I = (P \* T \* R)/100  
V = 4/3 pi R^3 ------------------------------------------- V = 4/3 \* PI \* R^3

#### 4 comments:

**[http://img2.blogblog.com/img/b16-rounded.gif](http://www.blogger.com/profile/08443409464317672470)**

[**Chandra Panday**](http://www.blogger.com/profile/08443409464317672470)**said...**

integer = %  
String = &  
single = %  
double = &

[July 11, 2012 at 10:11 PM](http://worldcomputerarticle.blogspot.com/2010/05/elements-of-qbasic.html?showComment=1342069914409#c4899990958820194501)

**[http://img2.blogblog.com/img/b16-rounded.gif](http://www.blogger.com/profile/08443409464317672470)**

[**Chandra Panday**](http://www.blogger.com/profile/08443409464317672470)**said...**

hello  
  
please   
  
send me a Complete Qbasic Note if any body with at least on project

[July 11, 2012 at 10:13 PM](http://worldcomputerarticle.blogspot.com/2010/05/elements-of-qbasic.html?showComment=1342070011693#c5800870145038349187)

**[Debaditya Mandal](http://www.blogger.com/profile/14748006143919341547) said...**

how to tell  
A number if it is a even or a odd no.  
means like   
if A = multiple of 2 then  
print A" is a even no. "

[July 3, 2013 at 5:06 AM](http://worldcomputerarticle.blogspot.com/2010/05/elements-of-qbasic.html?showComment=1372853177102#c3807994702950980569)

**[Sriram Pithani](http://www.blogger.com/profile/02812669334585627495) said...**

what r all the loop statements....plz tell

[August 12, 2013 at 12:19 PM](http://worldcomputerarticle.blogspot.com/2010/05/elements-of-qbasic.html?showComment=1376335198154#c5089003463254837340)