QBasic Tutorial

INTRODUCTION TO QBASIC

BASIC stands for **B**eginner's **A**ll **P**urpose **S**ymbolic **I**nstruction **C**ode. It was invented in 1963, at Dartmouth College, by the mathematicians John George Kemeny and Tom Kurtzas.

BASIC is an interpreter which means it reads every line , translates it and lets the computer execute it before reading another. Each instruction starts with a line number.

FEATURES OF QBASIC

1. It is a user friendly language.
2. It is widely known and accepted programming language.
3. It is one of the most flexible languages, as modification can easily be done in already existing program.
4. Language is easy since the variables can be named easily and uses simple English phrases with mathematical expressions.

RULES OF QBASIC

Every programming language has a set of rules that have to be followed while writing a program, following are some rules of QBASIC language:

1. All QBasic programs are made up of series of statements, which are executed in the order in which they are written.
2. Every statement should have at least one QBasic command word. The words that BASIC recognizes are called keywords.
3. All the command words have to be written using some standard rules, which are called "**Syntax Rules**". Syntax is the grammar of writing the statement in a language. Syntax Errors are generated when improper syntax is detected.

QBASIC DATA

Data is a collection of facts and figures that is entered into the computer through the keyboard. Data is of two types:

1. CONSTANT: Data whose value does not change or remains fixed. There are two types of constants:

(a) NUMERIC CONSTANT: Numbers - negative or positive used for mathematical calculations
e.g. –10, 20, 0

(b) ALPHANUMERIC CONSTANT / STRING: Numbers or alphabets written within double quotes(inverted commas " ").
e.g. "Computer", "Operating System"

2. VARIABLE: Data whose value is not constant and may change due to some calculation during the program execution. It is a location in the computer's memory, which stores the values. Depending on what value is held, Variables are of two types:
    (a) NUMERIC VARIABLE: The variable that holds a Numeric Constant for arithmetic calculations (+, - ,*, /) is called a Numeric Variable.
    e.g. A = 50, here A is the Numeric Variable

    (b) ALPHANUMERIC VARIABLE: The variable that holds an Alphanumeric Constant, which cannot be used for arithmetic calculations, is called Alphanumeric Variable or String Variable. An Alphanumeric variable must end with a $ sign and the Alphanumeric constant must be enclosed in inverted commas.
    e.g. Name$ = "Akanksha", here Name$ is an Alphanumeric Variable

TYPES OF MODE IN QBASIC

Once QBASIC program is loaded into the computer memory, it displays Ok prompt. Ok means it ready to accept the commands. QBASIC can be made to translate your instructions in two modes:

- Direct Mode
- Program Mode

1. Direct Mode: The accepts single line instructions from the user and the output is viewed as soon as enter key is pressed. The instructions are not stored in the memory. This mode can be used to do quick calculation. They do not have line numbers. E.g.
   Print 3+4

   Print "This is the Direct mode in QBasic"

2. Program Mode: The mode is used to type a program which is stored in the memory. They have line numbers. We have to give the command to get the output.
   e.g.

```
10 Print 3+4
20 End

RUN
```

Programs are built up with set of instructions or commands. Every programming language has its own SYNTAX (rules) and COMMANDS.

COMMAND/KEYWORDS IN QBASIC AND THEIR FUNCTIONS:

The following commands do not need line number.

1. LIST - The command is used to list the program on the screen.
2. RUN - The command is used to execute the program.
3. LLIST - The command is used to list of program as a hardcopy.
4. LPRINT- The command is used to get the output of the program on the hard copy.
5. NEW - The command is used clear the memory of the existing program.
6. SYSTEM – The command is used to take you back to dos prompt.
7. PRINT AND CLS command can also be used without a line number. Print is used to view the display on the screen and CLS to clear the screen.
8. RME - The command is used to show the position of the mistake.
9. SAVE -The keyword is used to save the program.
   E.g. SAVE “PROGRAM1” QBasic will automatically add a period and an extension “bas” to the filename.
10. LOAD - The keyword is used to LOAD the program from the disk to the memory.
    E.g. LOAD”PROGRAM1”

QBASIC COMMANDS

1. CLS: This command is used to clear the screen.

2. PRINT: Print command is used to display the output on the screen.
   E.g. Print “HELLO WORLD!!!”

   Print 80 * 8

   Print – Only Print command will leave blank space.

   Print Tab(10) ”Navrachana” – will print Navrachana on 10 column.

3. REM: It stands for Remark. It gives an explanation of the program or of the statements in the program thereby making the program more understandable to the reader. The computer does not execute this statement since whatever is written after REM is ignored by the compiler. REM can be used anywhere and many times in a program.

4. LET: It assigns a value to a variable in a program. It stores a value in the memory location.

SYNTAX: Let <Variable>=<Constant / Variable or Expression>

e.g. Let A = 15…….. Assigning constant to a variable

Let A = B……Assigning variable to a variable

Let C = A + B…. Assigning an expression to a variable

Using Let with Numeric Variables: Let A = 50, here A is a Numeric Variable and '50' is a Numeric Constant and value '50' is assigned to A and stored in the computer's memory for further calculations.

Using Let with Alphanumeric Variable: Let N$ = "QBasic Program", here N$ is an Alphanumeric Variable and "QBasic Program" is the Alphanumeric Constant assigned to N$.

NOTE: A numeric data should be assigned to a Numeric variable and an alphanumeric data to an alphanumeric variable otherwise "TYPE MISMATCH" error is displayed.

5. END: This command is usually given at the end of the program. Statements written after end are not executed since the program terminates execution on reading this command.

6. INPUT: This statement allows the user to enter a value for the variable while running the program. A question mark (?) appears on the output screen waiting for the user to enter a relevant data and then press enter key. Once the Return key or Enter key is pressed the data is stored in the variable.
   SYNTAX : INPUT < VARIABLE >

   e.g. Input A……Enter Numeric constant

   Input N$…..Enter Alphanumeric constant

   Input "Enter name:";N$….Giving relevant message to avoid erroneous data input

7. **DELETE <LINE NO.>:** To delete a line number in a program . e.g.
   Delete 10 will delete line number 10

Delete 30-50 will delete all line numbers between 30 to 50.

Print with Semi-Colon (;) Semi-colon placed after the message to be displayed, leaves no space between two messages.

e.g. Print "This is an example";" of QBasic program"

output: This is an example of QBasic program

Print with Comma(,): The screen of the computer is made of 80 columns and 40 rows. The columns are divided into five (5) zones of 14 columns each. Comma placed after the message prints the message zone wise on the screen.

QBASIC REMINDER

A QBASIC program consists of lines containing

1. A line number
2. A QBASIC keyword like PRINT,END etc
3. Each program line begins with positive number.
4. No two lines should have same number.

RUNNING A PROGRAM

RUN is the command used to execute the program and get the output on the screen.

WRITING A NEW PROGRAM

It is possible to overwrite lines with the new statements, but if you want to write a totally new program use a NEW command.

EXITING QBASIC

In order to exit the QBASIC program SYSTEM command is used.

Chapter I-QBasic Commands

When you open QBasic, you see a blue screen where you can type your program. Let's begin with the QBasic commands that are important in any program.

PRINT

Command **PRINT** displays text or numbers on the screen.

The program line looks like this:

PRINT "My name is Nick."

Type the bolded text into QBasic and press F5 to run the program. On the screen you'll see

My name is Nick.

Note: you must put the text in quotes, like this – "*text*". The text in quotes is called a string.

If you put the PRINT alone, without any text, it will just put an empty line.

PRINT can also put numbers on the screen.

PRINT 57 will show the number 57. This command is useful for displaying the result of mathematical calculations. But for calculations, as well as for other things in the program, you need to use variables.

Variables

When you think, you keep words or numbers in your mind. This allows you to speak and to make calculations. QBasic also needs to keep words or numbers in its memory. To do this, you use **variables**, pieces of QBasic memory, which can keep information. A variable can be named with any letter, for example – a. It can also have a longer name, which can be almost any word. It is important to know that there are two main types of variables – that keep a number and that keep a word or a string of words.

- **Numeric variables.** It's basically variables named with just a letter or a word. You tell this variable to keep a number like this:

  a = 15

  In other words, you assigned the value 15 to the variable **a**.

  QBasic will now know that the variable named **a** keeps the number 15. Now, if you type

  PRINT a

  and run the program, the computer will show this number on the screen.

- **String variables** can keep so called "strings", which is basically any text or symbols (like % or £), which you put in the quotes " ". You can also put numbers in a string variable, but again, you must include them in quotes, and QBasic will think that those numbers are just a part of text. The string variables look like this – **a$**. The $ sign tells QBasic that this variable contains text.

  Example:

  a$ = "It is nice to see you"

  PRINT a$

  On the screen you'll see:

  It is nice to see you

The PRINT command can print more that one string on the line. To do this, put the **;** sign between the variables. For example, you have two variables – **name$**, which contains name Rob, and **age**, which contains the number 34. Then, to print both name and age, you type:

PRINT "Name - "; name$; ". Age - "; age

> As you can see, the name of a variable can be more than just one letter – it can be a short word which describes what sort of information does this variable keep.

What you see on the screen when you run the program will look like this:

Name – Rob. Age – 34

Or, you can type the program like that:

PRINT "Name - "; name$

PRINT "Age - "; age

The result is:

Name – Rob

Age - 34

INPUT

INPUT is a command that allows you or anybody else who runs the program to enter the information (text or number) when the program is already running. This command waits for the user to enter the information and then assigns this information to a variable. Since there are two types of variables, the INPUT command may look like this – **INPUT a** (for a number), or **INPUT a$** (for a string).

Example (Type this program into QBasic and run it by pressing F5)

PRINT "What is your name?"

INPUT name$

PRINT "Hi, "; name$; ", nice to see you!"

PRINT "How old are you?"

INPUT age

PRINT "So you are "; age; " years old!"

END

Note: The END command tells QBasic that the program ends here.

You don't have to use PRINT to ask the user to enter the information. Instead, you can use

INPUT "Enter your name"; name$

and the result will be the same.

GOTO

Quite often you don't want the program to run exactly in the order you put the lines, from the first to the last. Sometimes you want the program to jump to a particular line. For example, your program asks the user to guess a particular number:

~ ~ ~ ~ *'some of the program here*

INPUT "Guess the number"; n

~ ~ ~ ~ *'some of the program there*

The program then checks if the entered number is correct. But if the user gives the wrong answer, you may want to let him try again. So you use the command GOTO, which moves the program back to the line where the question is asked. But first, to show QBasic where to go, you must "label" that line with a number:

1 INPUT "Guess the number"; n *'this line is labelled with number 1*

Then, when you want the program to return to that line, you type

GOTO 1

You can use GOTO to jump not only back but also forward, to any line you want. Always remember to label that line. You can have more than one label, but in that case they should be different.

Chapter II
Mathematical Calculations

QBasic was obviously created for us to have fun, play games, draw nice graphics and even make sounds.

But, as you might guess, nothing good comes without a bit of effort that has to be put in it. In the most QBasic programs a bit of math has to be done.

The math… Doh!

If you hate mathematics, don't worry. QBasic will do it all for you, you just need to know how to tell QBasic to do that.

QBasic can perform the following mathematical operations:

Operator	*What it does*	*Example*	*Result*
+	Add	**7 + 2**	**9**
-	Subtract	**7 – 2**	**5**
*	Multiply	**7 * 2**	**14**
/	Divide	**7 / 2**	**3.5**

Examples:

1. 1. **a = 15 / 4 + 3**

   PRINT a

   Result on the screen – 6

2. **PRINT "Enter the first number"**

   INPUT a

   PRINT "Enter the second number"

   INPUT b

   c = a + b

   d = a * b

   PRINT a; "+"; b; "="; c

   PRINT a; "*"; b; "="; d

   END

When you run this program it goes like this:

Computer: Enter the first number

You: 22

Computer: Enter the second number

You: 18

Computer: 22 + 18 = 40

22 * 18 = 396

Advanced operations:

Operator	*What it does*	*Example*	*Result*
\	divides and turns the result into an integer (the whole number)	**7 \ 2**	**3**
^	Raises a number to the power of another number	**3 ^ 4** (means: 3 * 3 * 3 * 3) **2.5 ^ 3** (means:2.5 * 2.5 * 2.5)	**243** **15.625**
SQR	Calculates the square root of a number	**SQR(9)** **SQR(16)** **SQR(5)**	**3** (because: 3 ^ 2 = 9) **4** (because: 4 ^ 2 = 16) **2.236**
MOD	Divides two numbers, and if the result is not an integer (for example - 3.25), finds out how much to subtract from the first number in order to get the integer result.	**17 MOD 5**	**2** (because: 17 / 5 = 3.4 17 – 2 = 15 15 / 5 = 3)

The following program explains MOD. Type this program (except my comments) into QBasic accurately and run it to see how MOD works.

```
1 CLS                       this command clears the screen, so it's empty
  INPUT "Enter the first number "; a
  INPUT "Enter the second number "; b
  IF b = 0 THEN             checks if the second number is zero, because you can't divide by zero
     PRINT "the second number cannot be 0.  Try again."
     DO: LOOP WHILE INKEY$ = ""     waits for you to press a key to continue
     GOTO 1                 then sends you back to line 1
  END IF
  CLS                       clear the screen again
  c = a MOD b
  d = a / b
  e = a - c
  f = e / b
  PRINT a; "MOD"; b; "="; c
  IF c = 0 THEN             this checks if the result of a MOD b = 0, because it means that
                            the result of a / b is integer
    PRINT "because"; a; "/"; b; "="; d; " - integer.  Try again."
    DO: LOOP WHILE INKEY$ = ""          waits for you to press a key to continue
    GOTO 1                  then sends you back to the line 1
  END IF
  PRINT "because"; a; "/"; b; "="; d; " -not integer"   The rest of the program executes if the
  PRINT "but"; a; "-"; c; "="; e                        result of a / b is not integer
  PRINT "and"; e; "/"; b; "="; f; " - integer"
  END
```

This program may look very complicated for you, but don't worry. QBasic is a very easy language to learn and soon you'll be having fun with it. I promise you!

Chapter III

How QBasic decides what to do

From the previous chapters you have learned how to create a simple program with INPUT, GOTO and PRINT commands. In such a program, you are asked to type the information; QBasic processes it and then shows the result on the screen. In many programs (for example - games), the user has a choice of what to enter. In this case, QBasic has to check what the user has typed, and to react accordingly. This can be done with the IF...THEN command.

IF…THEN…ELSE

This command checks if an argument involving a variable is true. An argument may look like this: **IF a = 15 THEN**... If the argument is true (and **a** really equals to 15), then QBasic executes the command you put after the IF...THEN.

Example:

IF a = 15 THEN PRINT "OK"

If the argument is not true (if **a** is not equal to 15), QBasic bypasses this line and goes to the next. In some cases, you can use the ELSE command, which tells QBasic exactly what to do if the argument is not true.

IF a = 15 THEN PRINT "OK" ELSE PRINT "It's not 15"

This example means that if **a** equals to 15, the computer will show OK on the screen. But if **a** is not equal to 15, you'll see

It's not 15

To check the argument in IF…THEN command, you can use any of these mathematical operators:

operator	*meaning*	*example*
=	Equal to	IF a = 15 THEN…
<>	Not equal to	IF a <> 15 THEN…
<	Less than	IF a < 15 THEN…
<=	Less or equal to	IF a <= 15 THEN
>	More than	IF a > 15 THEN…
>=	More or equal to	IF a >= 15 THEN…

You can make QBasic to execute more than one command if the argument is true. To do this, put those commands after IF…THEN and divide them with : symbol.

IF a = 15 THEN PRINT "OK": GOTO 1

This example means that if **a** equals to 15, QBasic will first print OK and then will go to the line labelled 1. Here is an example of full program (a simple game):

```
1 CLS
score = 0
PRINT "How many days are there in a week?"
INPUT a
IF a = 7 THEN GOTO 2
PRINT "Wrong answer!"
PRINT "To try again – press 'y'."
INPUT a$
IF a$ = "y" THEN GOTO 1 ELSE END
2 score = 10
PRINT "It's the right answer!"
PRINT "Your score is now"; score; "!"
PRINT "Thanks for playing."
END
```

Let's analyse how this program works.
The first command, CLS, clears the screen so it's empty. Then QBasic makes the variable **score** to be equal to 0. Then computer shows the question "How many days there are in a week?" You type your answer (a number) and QBasic puts it in the variable **a**. Then QBasic checks if the number in this variable equals to 7 (because there are 7 days in a week). If it equals to 7, the program goes to the line 2, where the variable **score** gets equal to 10. You get the message "It's the right answer! Your score is now 10! Thanks for playing." and then the program ends. But if you gave the wrong answer (that is, the number in the variable **a** is not 7), QBasic bypasses the line with IF…THEN, and shows the message "Wrong answer! To try again – press 'y'." You can then press the key 'y' to try again or press any other key to end the game. The value of the key you pressed goes to the variable **a$**, which, if you remember, is a string variable (because of the **$** symbol), and can contain only strings (letters, words or symbols). So the program checks if the key you pressed is really "y". If it is, the program takes you back to the line labelled 1, where the screen is cleared and the question is asked again. But if the key you pressed is some other key (not "y"), the program ends.

Sometimes you may want QBasic to execute more than two or three commands if the argument is true. Instead of putting all of them on one line, you can make an IF…THEN block:

```
IF a$ = "y" THEN
PRINT "OK, let's try again."
score = 0
GOTO 1
END IF
```

Note the END IF command at the end of this example. It tells QBasic that the commands, which should be executed if the argument is true, end here. This is important to separate the IF..THEN block from the rest of the program by putting END IF.
If you want QBasic to check more than one argument at once, use such words as AND and OR. For example – you want QBasic to execute commands in IF…THEN block if a is more than 12 but less than 50, somewhere in between. To program that, you can type:

```
IF a > 12 AND a < 50 THEN
```

Or, if you want commands to be executed if a equals either 6 or 12, you type:

```
IF a = 6 OR a = 12 THEN
```

Wow! So much said about that simple IF…THEN command in QBasic. It is indeed simple. IF you practise using this command in your programs, THEN you'll get the hang of it :-)

Chapter IV

Looping with QBasic

LOOPS

When a set of instructions are repeatedly executed a fixed number of times it is termed as Loop.

To make interesting and efficient programs, you can make QBasic to execute a part of a program more than once. This is called looping, when QBasic goes through a part of a program over and over again. This can be done with the GOTO command, but in QBasic there are some good ways to loop the program. One of them is FOR...NEXT command.

When using Counters, we have to follow the following points:

1. Initialize the counter
2. Increment or Decrement the counter
3. Check for the maximum limit

This process may become very tedious, if the program is very long with several counters. In that case we use loop and there are different types of loops like For-Next.

FOR...NEXT

This command allows you to execute a part of a program a certain number of times. It looks like this:

```
FOR i = 1 TO 4
PRINT "I am looping!"
NEXT i
```

This little stupid program will print on the screen:

```
I am looping!
I am looping!
I am looping!
I am looping!
```

The letter **i** can be anyother letter, **c** for example. It is actually a variable, which changes its value each time the program loops (in this example - from 1 to 3). So, if you make a program like this:

```
FOR a = 1 TO 5
PRINT "This is loop number"; a
NEXT a
```

This will print:

```
This is loop number 1
This is loop number 2
This is loop number 3
This is loop number 4
This is loop number 5
```

With FOR...NEXT you can use the STEP command, which tells QBasic how to count from one number to another. If you type:

FOR j = 0 TO 12 STEP 2
~ ~ ~
NEXT j

it will count by two:
0, 2, 4, 6, 8, 10, 12

FOR j = 0 TO 6 STEP 1.5
~ ~ ~
NEXT j

This will count:
0, 1.5, 3, 4.5, 6

You can also count backwards:

FOR d = 10 TO 1 STEP -1
~ ~ ~
NEXT d

When you want QBasic to count backwards, always put STEP -1 (or -*whatever*)!

FOR….NEXT: It can be referred to be as a self incrementing loop.

Purpose: To execute a series of instructions a specified number of times in a loop.

Syntax: **FOR var = x TO y [STEP z]**

Instruction series…

NEXT [var]

Where

- var is a variable. It is a counter and is called the 'Control Variable'.
- x is the start value
- y is the final or ending value
- x and y may be numeric constants, variables or expressions
- var is assigned all values starting with x terminating with y in steps of z

For e.g.

1. WAP to print numbers 10 to 15 using For…Next loop

```
10      Let N = 15
20      FOR M = 10 TO N
30      Print M;
40      Next M
```

Output as displayed on screen:

10 11 12 13 14 15

2. WAP to print numbers 40 to 50 in reverse order using For…Next loop

```
10      FOR A = 50 TO 40 STEP –1
20      Print A;
30      Next A
```

Output as displayed on screen:

50 49 48 47 46 45 44 43 42 41 40

3. WAP TO print first 10 multiple of any number input.

```
10 Input “Enter any number “,n

20 for I= 1 to 10

30 Print n; ”x”  ;I “=”  n* i

40 next

50 end

Run
```

Output

```
10      x       1       = 10

10      x       2       = 20

10      x       3       = 30

10      x       4       = 40

…….
```

IMPORTANT POINTS TO REMEMBER:

1. var increments or decrements depending on whether the number specified in STEP is positive or negative
2. If STEP is not given then BASIC assumes the increment to be 1
3. The loop will execute only once if the start and end values are equal.
4. The loop will not execute if the start value is less than the end value and the STEP is negative.
5. The loop will not execute if the start value is more than the end value and the STEP is positive.

Writing the control variable after NEXT is not essential by default it is taken as 1.

DO...LOOP

Imagine that you have a program that works like an ordinary calculator: you enter numbers, QBasic calculates and shows the result, and the program ends. The program may be good, but one problem is that you have to run the program each time you want to calculate!
That's where the handy DO...LOOP comes in. It's a block of comands, where the program doesn't have to loop a certain number of times, like in FOR...NEXT. It can loop indefinitely, while the condition is met (and when it's not met, the loop stops), or until the condition is met (so, when it's met, the loop stops). Condition is basically the same as an argument, for example **f < 20**

Here is an example:

```
DO
PRINT "Enter a number."
PRINT "When you want to quit, press 0."
INPUT n
r = n / 2
PRINT n; "/2 ="; r
LOOP WHILE n > 0
END
```

When you run this program, you can enter numbers and get the result as many times as you like. The program loops while numbers you enter are more than 0. Once you've entered 0, the program ends. The condition **WHILE n > 0** is put by the **LOOP** command but you can stick it to the **DO** command, like that:

```
DO WHILE n > 0
~~~
LOOP
Or you can use the word UNTIL instead, and put it either by DO or LOOP, like that:
DO UNTIL n = 0
~~~
LOOP
```

All these examples have the same effect: the program loops while numbers you enter are more than 0 (or, you can say - until the number you've entered is 0). Then QBasic stops looping and goes to execute commands you put after the DO...LOOP block (if it's END command, the program just ends).

Chapter V

More about Variables

So far you know that there are string variables (for holding text) and numeric variables (for holding numbers). But numbers can be very different, and in QBasic there are some different types of numeric variables:

Type of a variable	*The number it can hold*	*Example of a number*	*Example of a variable*
INTEGER	A whole number, from -32,767 to 32,767	**5**	**a%**
LONG INTEGER	A whole number, from more than -2 billion to more than 2 billion	**92,345**	**a&**
SINGLE PRESICION	A number with up to 6 digits after the decimal point.	**3.725**	**a!**
DOUBLE PRESICION	A number with up to 15 digits after the decimal point	**3.1417583294**	**a#**

As you see, those types of variables have a special symbol as part of their names:

% INTEGER

& LONG

! SINGLE

DOUBLE

The SINGLE type of variable is the most widely used in QBasic, and you don't have to stick the **!** symbol to it. If you put a variable without any symbol, QBasic will know that this is a SINGLE PRESICION variable.

QBASIC LIBRARY FUNCTIONS

At times we have to make very lengthy and complex calculations of the mathematical problems. Here, library functions come to our rescue, they provide a quick and easy way to evaluate many mathematical functions, carry out logical operations and also text operations.

MATHEMATICAL LIBRARY FUNCTIONS:

- ABS : Full form is Absolute value. It returns the absolute value of a number.
  Syntax : <Numeric Variable> = ABS <Numeric Variable>

  e.g.
  ```
  10  cls
  20  let A = -12
  30  B = ABS(A)
  40  print b
  50  end
  ```

  The output of the above program would be 12.

- SQR: Full form is Square Root. It calculates the square root of the value represented by the variable.
  Syntax : <Numeric Variable> = SQR <Numeric Variable>

  e.g.
  ```
  10  cls
  20  let a = 16
  30  b = SQR(a)
  40  print b
  50  end
  ```

  The output of the above program would be 4.

  NOTE : The computer will only give square root of positive number else it will give the error : ILLEGAL FUNCTION CALL.

- INT: Full form is Integer. It returns only the integer portion of a real number. The real number can be positive or negative or decimal number. INT only accepts the real number but ignores its the decimal part.
  Syntax : <Numeric Variable> = INT <Numeric Variable>

  e.g.
  ```
  10  cls
  20  let a = 13.5
  30  let b = -90.9
  40  c = INT(a)
  ```

```
50      d = INT(b)

60      print c, d

70      end
```

The output of the above program would be 13 and 90 respectively.

- **MOD :** It returns the integer remainder .
  Syntax :

  <Numeric variable>= <integer variable/constant> MOD <integer variable/constant>

  e.g.

```
10 cls

20 Let a=13

30 c= a MOD 2

40 print c

50 end
```

Output is :1(for 13 divided by 2 gives an integer remainder 1)

TEXT LIBRARY FUNCTIONS

- **LEN** : Returns the length of the string.
  Syntax : L = Len (string);

  Where, L = the variable storing the no of characters the string has.

  String = the string whose length has to be found.

e.g

```
10      cls

20      s$ = “Navrachana”
30      m = len(s$)
40      print m
50      end
```

The above program would give output = 10, as even space is considered as a character.

- **LEFT$:** Extracts number of characters specified from left of the string.

  Syntax : L$ = Left$(string, n)

  Where, L$ = the extracted left-most string.

  n = number of characters to be extracted.

  By default the number n = 1.

  e.g
  ```
  10    cls

  20    s$ = "Navrachana"
  21    let m$ = left$(s$, 7)
  22    print m$
  23    end
  ```
  The output of the above program would be "Navrachi", since it has extracted 7 characters from left from the string.

- **RIGHT$** : Extracts number of characters specified from the right of the string.

  Syntax : L$ = Right$ (string, n)

  Where, L$ = the extracted right most string.

  N = number of characters to be extracted.

  By default the number n = 1.

  e.g.
  ```
  10    cls

  20    s$ = "Monitor"
  21    Let m$ = Right$ (s$, 3)
  22    Print m$
  23    end
  ```
  The output of the above program would be "tor".

- **MID$** : Extracts specified part of a string

  Syntax : sa$ = Mid$(s$, I, j)

  Where, sa$ = the extracted part of the string

  s$ = the original string

  I = the position from which the character has to be extracted

  j = the number of characters to be extracted.

  e.g.
  ```
  10    cls

  20    s$ = "Mumbai"
  21    sub$ = mid$(s$, 1, 3)
  22    print sub$
  23    end
  ```

The output of the above program is "Mum".

- **ASC :** This function is used for converting the characters to their corresponding ASCII codes.
  Syntax : ASC(Alphanumeric Variable)

e.g.
```
10  cls
20  s$ = "A"
21  print ASC(s$)
22  end
```
The output of the above program would be 65.

- **CHR$:** This function is used for converting an ASCII code into its corresponding character.
  Syntax : CHR$(Numeric Variable)

  E.g.
```
10  cls

20  A = 66
21  Print CHR$(A)
22  End
```
The output of the above program would be 'B'.

RANDOM NUMBERS

- RND : Returns a random number between 1 and 0.
  e.g. Generate a random number between 100 and 150.

  Formula : N = first number + Int (Rnd * (Second number – First Number))

```
10  cls
11  N = 100 + Int (Rnd * (150 – 100))
12  Print N
13  End
```

Write the program in QBASIC for the following in your notebook.

A.

1. The interest on “P” rupees is given by SI = PRT / 100, where R is the rate and T is the time in year. Write a program to calculate the interest.

2. WAP with Let to store 2 numbers and:
   (a) Add and store result (b)Subtract and store result (c) Multiply and store result
   (d) Divide and store result

3. *
   * *

   * * *

   * * * *

B. Correct the following statements where necessary.

1. -10 REM this is my first program.

2. 20 ?this is all wrong

3. 10? “ This is QBasic programming”

4. 30? Rem this is a remark

5. 10 point “looks right or wrong?”

SAMPLE PROGRAMS WITH THEIR OUTPUT WITH THEIR PROGRAMS. TRY OUT THE FOLLOWING PROGRAMS ON THE SCREEN

A.

```
10    cls

20    s$ = “MUMBAI”
21    m = len(s$)
22    for I = 1 to m
23    sub$ = left$(s$, I)
24    print sub$
25    next I
26    end
```

```
M

M U

M U M

M U M B

M U M B A

M U M B A I
```

B.

```
10    cls

20    s$ = “MUMBAI”
21    m = len(s$)
22    for I = 1 to m
23    sub$ = right$(s$, I)
24    print sub$
25    next I
26    end
```

```
I

A I

B A I

M B A I

U M B A I

M U M B A I
```

C.

```
10    cls

20    s$ = “MUMBAI”
21    m = len(s$)
22    sub$=””
23    for I = 1 to m
24    sub$ = mid$(s$, I, 1) + sub$
25    print sub$
26    next I
27    end
```

```
M

U M

M U M

B M U M

A B M U M

I  A B  M U M
```