15 PRINT formatting and INPUT

PRINT formatting

You are already familiar with the **PRINT** statement; and how it is used to put characters on the screen. In this chapter you will find out how to use **PRINT** to position the output on the screen.

Press **BREAK** and then try the following program.

```
10 X = 6
20 PRINT X;X;X
30 PRINT X,X,X
40 PRINT X'X'X
>RUN
         666
         6          6          6
         6
         6
         6
```

From this you can see that

(i) Items in the **PRINT** instruction separated by semicolons are printed one after the other, with no spaces.

(ii) Items separated by commas are tabulated into columns. These columns are ten characters wide and are called fields.

(iii) Items separated by apostrophes are printed on a new line. Now try the following program:

```
10 A$ = "J"
20 PRINT A$;A$;A$;A$;A$
30 PRINT A$,A$,A$
40 PRINT A$'A$'A$
>RUN
```

```
JJJ
J          J          J
J
J
J
```

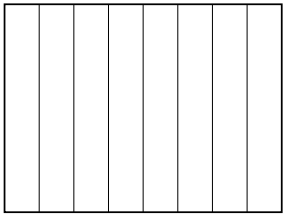
This is the same program as before, but using a string variable. Notice that the character is not printed in the same place as the number. Here is another program to demonstrate this:

```
10 X = 6
20 A$ = "J"
30 PRINT X, XX,X'A$,A$,A$
>RUN
          6          6          6
J          J          J
```

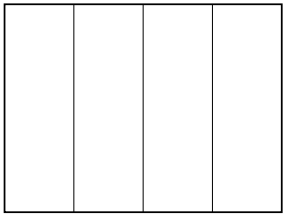
Field No 1 Field No 2 Field No 3

Each field of ten characters width is shown above. As you can see, numbers are printed out to the right of each field, characters to the left. This is done so that numbers line up in the units column, or the least significant decimal.

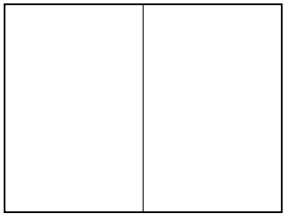
The width of these fields can be altered, as can the number of decimal places to which real numbers are printed. The Electron normally gives each field a width of ten characters. The number of fields across the screen depends upon which mode you are using. There are three different character sizes that are available, and these give either eight fields, four fields, or two fields, each of width ten.



80 character modes
(MODES 0 and 3)



40 character modes
(MODES 1,4,6)



20 character modes
(MODES 2 and 5)

If you now type

```
PRINT 6,9,7/3,57
```

you will see that the 9 and the decimal equivalent of 7 /3 have ran into each other:

```
6             92.33333333          57
```

To prevent this happening the field width and/or the number of decimal places can be altered using the integer variable @%.

If you type

```
@% = & 2040A
```

and then

```
PRINT 6,9,7/3,57
```

you can see the effect of reducing the number of decimal places to 4.

```
6.0000     9.0000     2.3333    57.0000
```

The assignment of the variable @% is made up of a number of parts:

& indicates that a hexadecimal number follows. Hexadecimal numbers are explained in chapter 6.

After the & sign, the first number of **@%** indicates the format of the print field. 2 means that the computer will print a fixed number of decimal places.

The next two figures indicate the number of decimal places which are required, in this case 04.

The final two figures give the field width, in this case 0A which is 10 in decimal.

So,

```
@% = &20105
```

will give each number printed to one decimal place, with a field width of five.

Some more points:

(i) The format, the first figure after the **&** symbol, can take three values:

0 is the normal configuration - the format which the computer uses when it is first switched on.

1 gives numbers in exponent form, that is an integer followed by a power of ten. So 0.0006 would be printed 6E-4. ('Six times ten to the power of minus four'.)

2, as just shown, gives numbers to a fixed number of decimal places.

(ii) When the computer is first switched on, **@%** = &0090A. This gives nine significant figures and a field width of ten.

(iii) The computer will not print more than ten significant figures. The ten significant figure format is obtained by setting **@%** to, for example, &00A0C. This will give ten significant figures and a field width of 12. Alternatively, typing **@% = 12** will give the same result, because the number of significant figures is assumed to be ten if it is not specified.

Y on can make the print instruction convert decimal numbers or variables into hexadecimal by using the ~ character.

```
PRINT ~10
```

will give A (decimal 10 in hex).

```
PRINT ~LENGTH
```

will print out the contents of the variable **LENGTH** in hex.

The position on the screen at which **PRINT** prints is controllable by the **TAB** instruction.

```
PRINT TAB(16); "J"
```

will print the character J 16 spaces across the screen.

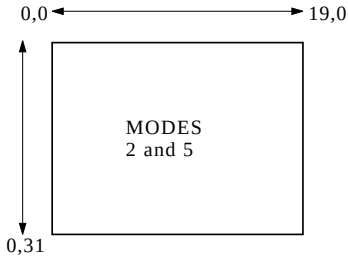
As is usual with functions, the number in the brackets may be a variable, or any arithmetic expression.

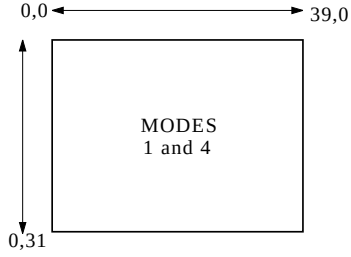
TAB can also be used with two parameters (numbers in the brackets). If you imagine the screen to have coordinates, the number of columns going across the top, and the number of rows down the side, then

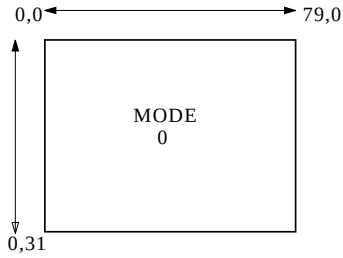
```
PRINT TAB(16,22); "J"
```

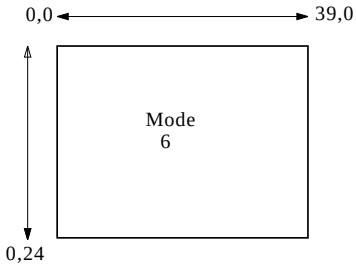
will print the character J 16 spaces across the screen and 22 spaces down.

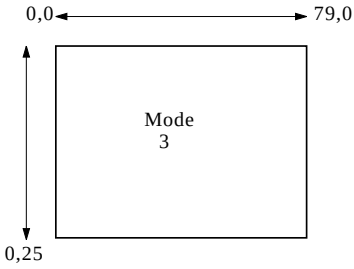
These text coordinates vary depending upon which mode you are using.











TAB instructions have the effect of moving the cursor around the screen.

```
PRINT TAB(0,0)
```

will always move the cursor to the top left of the screen in any mode.

If at any time you wish to turn the cursor off, you can do so by typing

```
VDU 23,1,0;0;0;0;
```

It will still exist of course, but it will not be printed on the screen.

```
VDU 23,1,1;0;0;0;
```

will return the cursor to the screen once more.

INPUT

So far, the only form of input that you' vemade to the computer is the typing of commands and programs. Often you will need to give the computer information while the program is running.

```
10 PRINT "GIVE ME A NUMBER AND I'LL DOUBLE
IT";
20 INPUT X
30 PRINT "DOUBLE ";X;" IS ";X*2
>RUN
GIVE ME A NUMBER AND I'LL DOUBLE IT
?16
DOUBLE 16 IS 32
```

When you mn this program, line 20 prints a question mark on the screen. This question mark invites you to type in some data. When you press **RETURN** the number that you typed is put in X. If you don' t type a number, or you type letters or symbols instead, X becomes zero.

INPUT may also be used with string and integer variables.

```
10 PRINT "WHAT IS YOUR NAME"
20 INPUT A$
30 PRINT "NICE TO MEET YOU, ";A$
```

```
>RUN
WHAT IS YOUR NAME
?DOBBIN
NICE TO MEET YOU, DOBBIN
```

Line 10 of the above two programs have been used to print a message on the screen. This message can be incorporated into the **INPUT** instruction.

```
10 INPUT "WHAT IS YOUR NAME ",A$
20 PRINT "ARE YOU SURE ABOUT THAT, " A$;"?"
>RUN
WHAT IS YOUR NAME ?EINSTEIN
ARE YOU SURE ABOUT THAT, EINSTEIN?
```

Notice the comma in line 10 of this program. It tells the computer to print a question mark when it wants input from the keyboard. If you leave out the comma, the question mark will not be printed. A semi-colon may be used, and has exactly the same effect as the comma.

When the program is running, the **INPUT** instruction requires you to press **RETURN** when on wish to send what you have typed to the computer. Up until **RETURN** is pressed you can delete all or part of what you have typed using **CTRL** U or the **DELETE** key.

When you are inputting a string, the computer will ignore any leading spaces, and anything after a comma, unless you put the whole string inside quotation marks.

```
10 INPUT A$
20 PRINT A$
>RUN
?BUS, CAR
BUS
>RUN
?"BUS, CAR"
BUS, CAR
```

Alternatively, you can use INPUT LINE, and inverted commas will not be needed.

```
10 INPUT LINE A$
20 PRINT A$
>RUN
?FISH, AND CHIPS
FISH, AND CHIPS
```

Several inputs can be requested at one time.

```
10 INPUT A,B,C$
20 PRINT A,B,C$
>RUN
?20.3, -16, INCHES
      20.3        -16INCHES
```

This time you must use commas to separate each piece of data which you type.

Another way of entering data is to use **GET$**. This reads the key which you press.

```
10 PRINT "PRESS A KEY"
20 A$ = GET$
30 PRINT "THE KEY YOU PRESSED WAS "; A$
```

The program waits at line 20 until you press a key. As soon as you do, the character which that key represents is placed in **A$**.

A similar instruction to **GET$** is **INKEY$**.

```
10 PRINT "PRESS A KEY IN THE NEXT SECOND"
20 A$ = INKEY$100
30 IF A$ = "" THEN PRINT "YOU WERE TOO SLOW"
ELSE PRINT "THE KEY YOU PRESSED WAS ";A$
```

Using this program, line 20 waits one second for you to press a key. If no key is pressed in that time then the program moves on. The **INKEY$** instruction has a number after it which is hundredths of a second. The larger the number, the longer the computer will wait for you to press a key. If you don' t press a key in time, INKEY$ will give the value -1.

Line 30 of this program shows an **IF** statement **IF** statements are discussed in detail in chapter 16.

There are two more input instructions, **GET** and **INKEY**. These are exactly the same in operation as **GET$** and **INKEY$**, but their effect is to give the ASCII code of the key which you press. **GET** and **INKEY** give a number, and must therefore be assigned to number variables.

```
10 X = GET
20 Y = INKEY500
```

If you use either **INKEY** or **INKEY$** with its time delay set to 0, the computer will not wait for you to press a key, but will merely glance at the keyboard to see if any key has already been pressed. This is useful when you come to write games, as the rest of the program is not held up waiting for a key to be pressed.

It is important to know that **INKEY**, **INKEY$**, **GET** and **GET$** will notice not just the key you happen to be pressing at the time, but also any key you have pressed since the last time the computer asked for input. Every time you press a key it is placed in the keyboard buffer (a buffer is a short term memory), and it is this buffer which **GET** and the others actually look at.

However, if you want to ignore any keys previously pressed, and just look at the keyboard directly, there' sa version of **INKEY** to do this. This is done by giving **INKEY** a negative number, each key having a corresponding value which you can use. For example, **INKEY(-70)** only operates when the J key is pressed. This use of **INKEY** is discussed in more detail in chapter 25, including a table of all the key codes.