

# Formatted Output

---

## Introduction

---

Since DarkBASIC Pro is primarily a games programming language, it doesn't give a great deal of thought to such boring things as allowing the programmer to display real numbers in a given format. So although we might want to display an amount of money as

```
$12.34
```

we're much more likely to see

```
$12.3400001526
```

This inaccuracy is caused by rounding errors as the original decimal number is converted to binary for storage in the computer and then converted back to be displayed on the screen.

### The STR\$ Statement (Update)

The original STR\$ simply turned a number into a string as shown in the following code snippet:

```
result$ = STR$(v#)  
PRINT result$
```

#### Activity Sup001

Use the code given above to create a complete program which allows 6 numbers to be read (use READ and DATA) and displays the string equivalent of each number.

Use the following numbers as input when running the program:

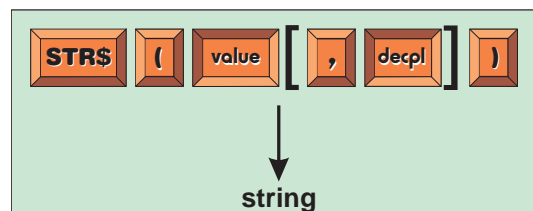
```
1289.345, 5.0, 2, 0.000007, 0.0, -12.3
```

As you can see from the result of your program, the numbers are hardly arranged in a satisfying tabular style.

However, the latest version of DarkBASIC Pro has added a new parameter to the STR\$ which allows us to specify the number of decimal places to be given in the string created. The updated version of the STR\$ statement has the format shown in FIG-Sup02.

FIG-Sup02

The STR\$ Statement



In the diagram:

*value*

is any numeric value.

*decpl*

is an integer specifying the number of decimal places to be given in the string.

Examples of expressions and the resulting string are given in TABLE-Sup01.

**TABLE -Sup01**

STR\$ Examples

Expression	String Created
STR\$(12)	"12"
STR\$(0)	"0"
STR\$(-3)	"-3"
STR\$(9.5)	"9.5"
STR\$(9.0)	"9"
STR\$(1.3)	"1.29999995232"
STR\$(12,2)	"12.00"
STR\$(0,2)	"0.00"
STR\$(-3,2)	"-3.00"
STR\$(9.5,2)	"9.50"
STR\$(1.3,2)	"1.30"
STR\$(12.69,1)	"12.7"
STR\$(12.69,0)	"13"

There are a few examples to note from the table:

- STR\$(9.0) drops the .0 from the string
- STR\$(1.3) gives a string which shows the inaccuracy due to conversion
- STR\$(1.3,2) gives a string which is correct
- STR\$(12.69,1) rounds to the nearest value
- STR\$(12.60,0) rounds to the nearest integer

**Activity Sup002**

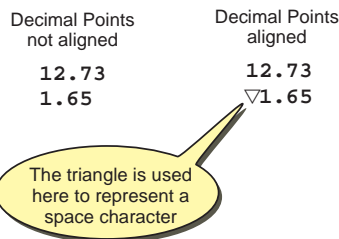
Modify your previous program to display each of the values read to two decimal places.

Notice that the list of numbers isn't decimal-point aligned. To achieve this we'll need to write our own function.

The reason the numbers don't align is because they have varying numbers of digits in the integral part of the value (that is, the digits to the left of the decimal point). To fix this we need to pad-out shorter numbers with spaces (see FIG-Sup03).

**FIG-Sup003**

Aligning the Decimal Point



This technique only works properly if a mono-spaced font is being used when the string is displayed, since every character in a mono-spaced font has exactly the same width.

---

## The Format\$() Function

---

In order to achieve this degree of control over the string, we'll create a new routine called *Format\$* which converts a number to a string with a specified number of characters before and after the decimal point.

As we've already seen in Hands On DarkBASIC Pro Volume 1, in a professional company, the details of a new routine are often stated in the form of a mini-spec, which gives exact details of the function. Below is the mini-spec for our new routine.

Name	:	Format
Parameters		
IN	:	value : numeric intsize : integer decpl : integer
OUT	:	result : string
IN/OUT	:	None
Pre-condition	:	$1 \leq \text{intsize} \leq 15$ AND $0 \leq \text{decpl} \leq 8$
Description	:	value is converted to a string. The string will have <i>intsize</i> digits to the left of the decimal point ( including any sign character), and <i>decpl</i> characters after the decimal point. If <i>decpl</i> is zero then the decimal point itself will not be included in the string. If <i>intsize</i> is less than the digits required to show all the digits in the integral part of value, then <i>intsize</i> will be ignored and the string will be expanded to whatever size necessary to hold an accurate representation of value. The final string is held in <i>result</i> .
Outline Logic:		Convert value to string with required decimal places Pad out integral part with spaces as required

This is the type of information we would give to the programmer who would be responsible for creating the routine.

A few things to note:

- The parameter, *value*, is specified as numeric, meaning it can be either an integer or real value.
- Neither the routine nor the returned value contain a \$ character in their name. This is because a mini-spec should be independent of the language to be used. Other languages (such as C++ or Java) would not use a \$ character in the names.
- The pre-condition specifies any restrictions on the parameters' possible values. If the parameters fall outside these values, the routine will not return the expected result.
- The description details how to perform a normal conversion, but also how to handle exceptional cases.

Now we're ready to start producing code using the details given in the mini-spec.

The first line of the routine is

```
FUNCTION Format$(value#,intsize,decpl)
```

Notes:

- All these details came from the mini-spec.
- The function name needed to be changed slightly since DarkBASIC Pro requires functions that return strings to end with a \$ sign.
- The parameter *value* has been renamed *value#* to allow both real and integer values to be passed (remember an integer value can be copied into a real variable).

Next we check to see if any of the pre-conditions have not been met:

```
IF intsize<1 OR intsize>15 OR decpl<0 OR decpl>8
```

and, if so, exiting from the function. But, since this function is designed to return a string, we must return one - in this case we'll return an empty string:

```
    return ""
ENDIF
```

Now we must write code to perform each line of the Outline Logic. All you have to do is look at each line of the outline logic in turn and write the DarkBASIC Pro equivalent (see TABLE-Sup02).

**TABLE -Sup02** Converting Outline Logic to DBPro

Structured English	DarkBASIC Pro
Convert value to string with required decimal places	result\$ = STR\$(value#,decpl)
Pad out integral part with spaces as required	result\$ = SPACE\$((intsize+decpl+1)-LEN(result\$))+result\$

The complete routine has the following code:

```
FUNCTION Format$(value#,intsize,decpl)
  REM *** If pre-condition not met, exit ***
  IF intsize<1 OR intsize>15 OR decpl<0 OR decpl>8
    EXITFUNCTION ""
  ENDIF
  REM *** Convert value to string with required dp ***
  result$ = STR$(value#,decpl)
  REM *** Pad string to required size ***
  result$ = SPACE$((intsize+decpl+1)-LEN(result$))+result$
ENDFUNCTION result$
```

### Activity Sup003

Use this function in your previous program to display the set of numbers in the DATA statement with 5 digits before and 2 after the decimal point.

Are the values decimal-point aligned?

# Solutions

## Activity Sup001

```
DATA 1289.345, 5.0, 2, 0.000007, 0.0, -12.3
FOR c = 1 TO 6
  READ v#
  PRINT STR$(v#)
NEXT c
WAIT KEY
END
```

## Activity Sup002

```
DATA 1289.345, 5.0, 2, 0.000007, 0.0, -12.3
FOR c = 1 TO 6
  READ v#
  PRINT STR$(v#,2)
NEXT c
WAIT KEY
END
```

## Activity Sup003

```
DATA 1289.345, 5.0, 2, 0.000007, 0.0, -12.3
FOR c = 1 TO 6
  READ v#
  PRINT Format$(v#,5,2)
NEXT c
WAIT KEY
END
```

```
FUNCTION Format$(value#,intsize,decpl)
  REM *** If pre-cond not met, exit ***
  IF intsize<1 OR intsize>15 OR decpl<0
    ↵OR decpl>8
    EXITFUNCTION ""
  ENDIF
  REM *** Value to string ***
  result$ = STR$(value#,decpl)
  REM *** Pad string to required size ***
  result$ =
  ↵SPACE$((intsize+decpl+1)-LEN(result$))
  ↵+result$
ENDFUNCTION result$
```