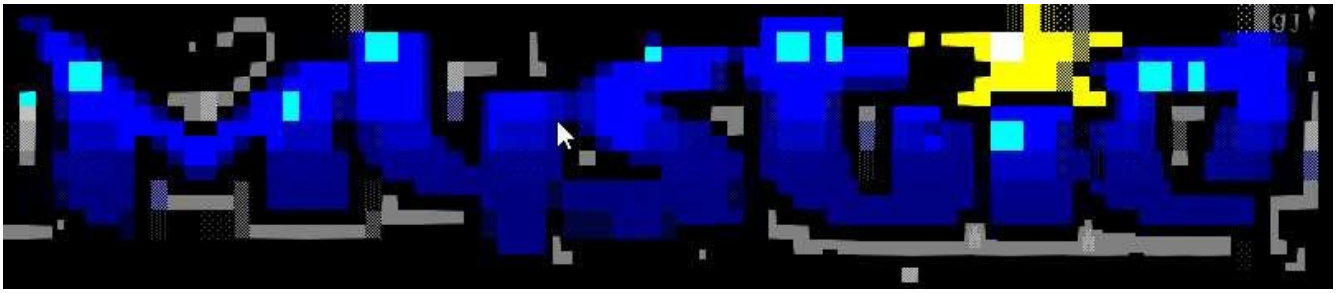

MPL Training
Lesson 18
Black Panther

-=-=-=-=-=-=-=-=-

Wow! I can't believe we're already on lesson 18. I hope these are benefiting all of you, and helping you to understand how MPL works.

In this lesson, we're going to take a closer look at some of the specific functions available with Mystic and can be used in MPL programs. These will deal more with either obtaining information to use in your MPL, or ways to write information to the screen.

-=-=-=-=-=-=-=-=-

MCI2STR (String): String

This function will take any of the MCI codes within Mystic, and return the string containing the information. For example:

```
Var BBSName : String

BBSName:=MCI2Str('BN')
WriteLn('The name of the BBS is '+BBSName)
```

Or, you could shorten it up, and write:

```
WriteLn('The name of the BBS is '+MCI2Str('BN'))
```

-=-=-=-=-=-=-=-=-

MCILength(String): Integer

This function will try to return the length of a string without counting the MCI codes.

```
Var B: Byte
B:=MCILength('|15Hello|UH')
WriteLn('Length should be 5: '+Int2Str(B))
```

-=-=-=-=-=-=-=-=-

ReadKey: Char

This function will read a single key from the user. It is also useful when you wish to have a pause in your MPL program, without having an actual pause prompt. It will wait for the user to hit a key.

```
Var Ch : Char
Repeat
  Ch:=ReadKey
Until Keypressed

WriteLn('You entered: '+Ch)
```

This example will wait for the user to hit a key, and then tell the user which key they hit on their keyboard.

If you wanted to use this as a pause, you could just put:

```
ReadKey
```

When the program got to this line, it would wait for the user to hit a key before proceeding.

-=-=-=-=-=-=-=-=-

StuffKey(S: String)

This procedure can be handy if you want to put text into the buffer for the user. A good example of this, is my Scrollz animated message banner. It will run until the user hits a key. I then use the StuffKey to put this into the buffer that is passed back to Mystic. This way, Mystic is able to act on whatever key was pressed.

```
Repeat
…
Until Keypressed
Ch:=ReadKey
StuffKey(Ch)
Halt
```

What this is doing, is running the Repeat/Until loop until the user hits a key. At that point, it will save the keystroke into Ch, and then stuffs that keystroke into the buffer, and exits. When Mystic takes control back from the MPL, it will open the menu, or command associated with that keystroke.

-=-=-=-=-=-=-=-=-

WhereX: Byte
WhereY: Byte

These are both very useful, if you want your MPL program to know where the cursor is located on the screen. I've used these to find out if a header file was displayed or not in some of my MPLs.

```
Var
  X : Byte
  Y : Byte

X:=WhereX
Y:=WhereY

WriteLn('Your cursor is located at '+Int2Str(X)+','+Int2Str(Y))
```

In my MPLs, I'll check where the Y position is. If it's not showing '1', then I know a header file has been displayed. Also, in my pause prompts, I can check to see if the X coordinate is at 1. If it's not, I can send a carriage return to the screen, so the cursor is at the next line down, and at X coordinate of '1'.

-=-=-=-=-=-=-=-=-

```
Write(Text)
```

This procedure is used to write information to the screen. Write will **NOT** send a carriage return at the end of the line. Sometimes this is what you need, but if you want the carriage return, use the WriteLn procedure, which will be next.

```
Write('Hello ')
Write('World')
```

This would display 'Hello World' on the screen.

```
WriteLn('Hello')
WriteLn('World')
```

Would display:
Hello
World

-=-=-=-=-=-=-=-=-

```
WriteRaw(Text)
WriteRawLn(Text)
```

These procedures will work basically the same as the Write and WriteLn procedures, but will **NOT** parse any pipe codes or MCI codes.

```
WriteRawLn('|11Hello there |UH')
```

Would display:

|11Hello there |UH

on the screen.

-=-=-=-=-=-=-=-=-

WriteXY(X, Y, Z: Byte; Str: String)

Now, I'm sure you've already seen the following lines in some MPLs.

GotoXY(1,15)
Write('This is fancy text displayed on the screen')

This can actually be accomplished with one procedure.

WriteXY(1,15,11,'This is fancy text displayed on the screen')

The X and Y are the coordinates to display the text.

The Z is the color code you would like the text displayed in. In the above example, it was displayed in cyan. (11)

This function can be used any time you want to have text displayed in a particular position on the screen. It cannot, however, parse any MCI codes. So if you have |15 to display in white in the text, it will display the |15 on the screen.

-=-=-=-=-=-=-=-=-

WriteXYPipe(X, Y, Z: Byte; Len: Integer; Str: String)

This procedure is similar to WriteXY, but will allow you to tell it the length of the text, and it will pad the output to fit that space.

WriteXYPipe(1, 10, 8, 50, 'This pads to 50 chars at location 1, 10')

-=-=-=-=-=-=-=-=-

CHR(B: Byte): Char

This function can be useful in obtaining information from the user. It will take the ASCII code number, and change it to the actual character.

WriteLn('Hello'+Chr(32)+'World')

This will actually output:
Hello World
This is because ASCII 32 is the number for a space. This function can handle all 255 characters within the ASCII character set.

-=-=-=-=-=-=-=-=-

ORD(C: Char): Byte

This is pretty much the opposite of the CHR function, as this will change a character into it's ASCII character code.

```
WriteLn(Ord(' '))
```

This will display '32', as that's the ASCII code for a space.

-=-=-=-=-=-=-=-=-

Copy(S: String, Index: Byte, Count: Byte): String

Have you had a string in a program, where you only want to work with one part of that string? Let's say you have:

```
Str:='Hello World'
```

You only want to know what the first 5 characters are of that string.

```
WriteLn(Copy(Str, 1, 5))
```

This would display Hello to the screen. You could also copy this into a different string variable as well.

```
Str1:=Copy(Str, 1, 5)
```

Now, Str would contain 'Hello World' and Str1 would contain 'Hello'.

The Index is where you would like it to start copying from, and the Count is how many characters you want it to copy.

-=-=-=-=-=-=-=-=-

Delete(S: String, Index: Byte, Count: Byte)

This is actually similar to the Copy procedure, but will actually delete the text from the current string.

```
Str:='Hello World'
Delete(Str, 6, 6)
WriteLn(Str)
```

This would give you 'Hello' printed to the screen. It starts from the Index character position, and removes Count number of characters.

-=-=-=-=-=-=-=-=-

Insert(Source: String, Target: String, Index: Byte)

Now that we've removed text from a string variable, let's see how we can insert text into the string variaible.

Source is the text you would like to insert into another string.

Target is the string you want the text inserted into.

Index is where you would like it inserted.

```
Str : String='RCS Team'
Insert('Development ', Str, 5)
WriteLn(Str)
```

This would display 'RCS Development Team' on the screen.
-=-=-=-=-=-=-=-=-

Replace(Str1, Str2, Str3: String): String

We've talked about inserting text into a string, and deleting text in a string. Now we'll get crazy, and talk about how to replace text in a string. :)

This function will replace ALL occurrences of the text you indicate, with the text you want it to be.

Str1 is the given text string you are working with.
Str2 is the text you want to replace within Str1
Str3 is what you want to replace Str2 with

```
Var Str : String='Hello Hello Hello'
Str:=Replace(Str, 'Hello', 'World')
```

At this point, Str would contain the text 'World World World'.

-=-=-=-=-=-=-=-=-

Int2Str(L: Integer): String

You will see this function used a lot in WriteLn statements. What this function does, is convert, or typecast, an integer variable into a string. This is necessary when using WriteLn, as it can only write strings. So, whenever you want to display a byte, integer, longint, etc, it will need to be typecast.

```
WriteLn('I see that your age is '+Int2Str(age))
```

-=-=-=-=-=-=-=-=-

Str2Int(S: String): Integer

This is one that can be very useful, but can also cause people a lot of issues. What it does, is take a string, and will attempt to convert it (typecast) as an integer.

```
Var
  Str: String='29'
  Age: Byte
```

```
Age:=Str2Int(Str)
WriteLn(Age)
```

This would output ‘29’ on the screen.

The reason this one causes issues for some people, is the string has to be checked beforehand, to make sure the string does contain a valid integer. If you have a string variable that contains ‘AGE29’, and you try to run Str2Int on it, it will crash your program, as it contains characters that are not numeric.

-=-=-=-=-=-=-=-=-

One more, we’ll call this one a bonus. :)

StrComma(gold: LongInt): String

This function will take an integer, and return a string that contains commas as the thousand separators.

```
WriteLn(StrComma(1000000))
```

This would output 1,000,000 on the user display. It’s much easier to read, than trying to count the number of zeros. ;)

-=-=-=-=-=-=-=-=-

This should give you a better understanding of obtaining input from the user, and displaying information to the display. While this is usually only a small part of the code that we write, it is also the areas that can take the longest to write. We focus so much on how things look to the user, and make sure we are getting the right information from the users, that it can be the most important parts of our programs. After all, the input and output are the only things the user has to judge our programs. Most of the time, they won’t look at everything else that is happening in our code, in the background.