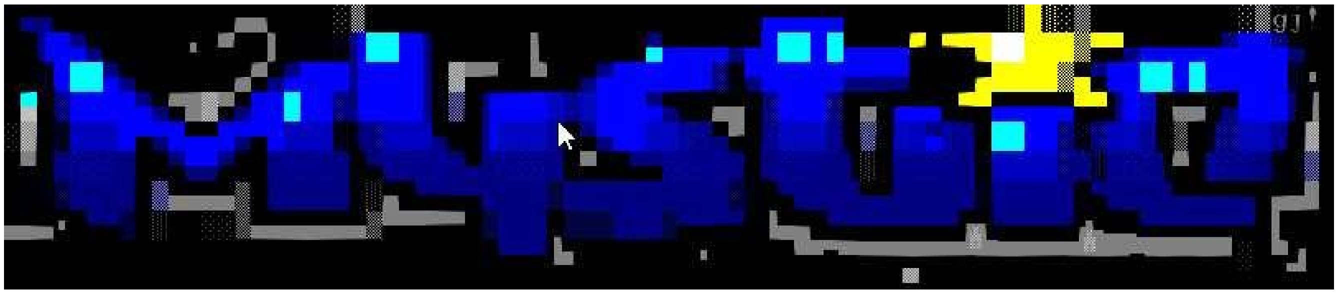


MPL Training
Lesson 13
Black Panther

Tips ‘n Tricks

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

Ok, so now that you know everything there is to know about MPL, I’d like to share some tips and tricks that I’ve developed. (Just kidding about knowing everything, but you’re on your way!)

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

First, and probably most useful tip. When you are typing a function into your IDE, you probably start with the procedure or function name, and any parameters it takes. Then you type in your variable names, and start working on the code block.

Well, when you start typing:

```
Function ReadCall:Boolean
Var
  Ret     : Boolean = False
  Fptr    : File
  Counter : Byte = 1
Begin
```

STOP!

As soon as you type in ‘Begin’, hit enter twice, and type in ‘End’.

```
Function ReadCall:Boolean
Var
  Ret     : Boolean = False
  Fptr    : File
  Counter : Byte = 1
Begin

End
```

That way, you already have the ‘End’ in place, and don’t have to worry about forgetting it. It also helps you ensure that your code blocks are indented properly as well. Now, just move your cursor back up onto the blank line, indent your code, and start typing again.

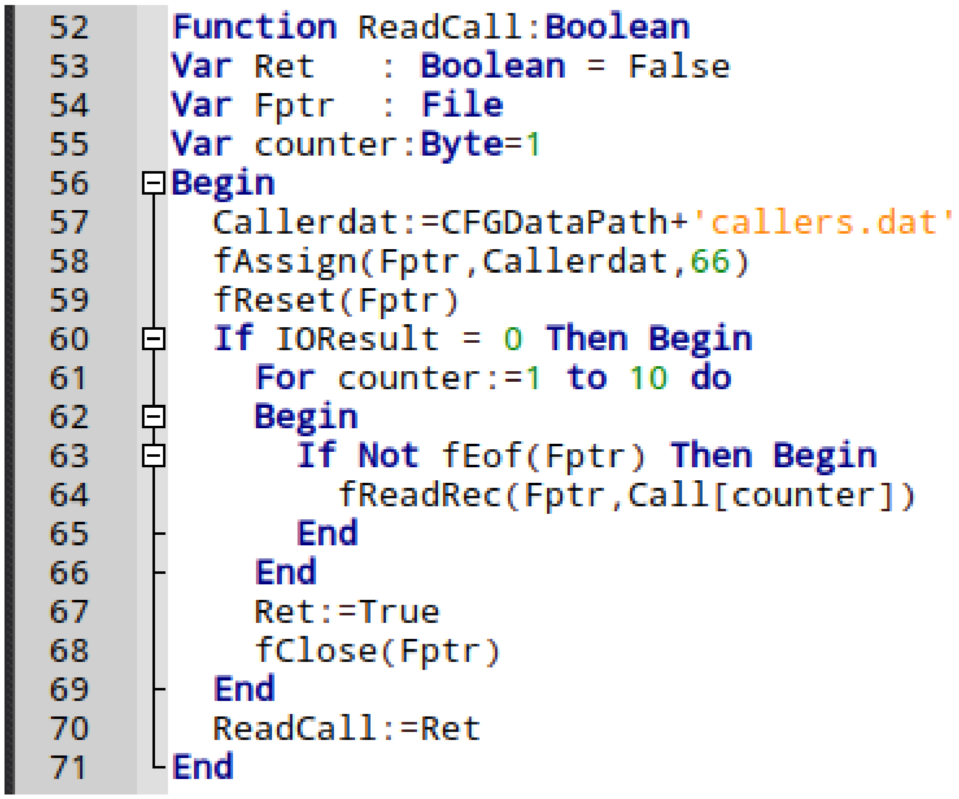
```
Function ReadCall:Boolean
Var
  Ret     : Boolean = False
  Fptr    : File
  Counter : Byte = 1
Begin
  Callerdat:=CFGDataPath+’callers.dat’
  fAssign(fptr, Callerdat, 66)
  fReset(fptr)
  If IoResult = 0 Then
  Begin

End
```

STOP! Enter in the ‘End’ right away!

```
Function ReadCall:Boolean
Var
  Ret     : Boolean = False
  Fptr    : File
  Counter : Byte = 1
Begin
  Callerdat:=CFGDataPath+’callers.dat’
  fAssign(fptr, Callerdat, 66)
  fReset(fptr)
  If IoResult = 0 Then
  Begin
    For Counter:=1 to 10 Do
    Begin
      If Not fEof(fptr) Then
      Begin
        fReadRec(fptr,Call[counter])
      End
    End
  End
End
```

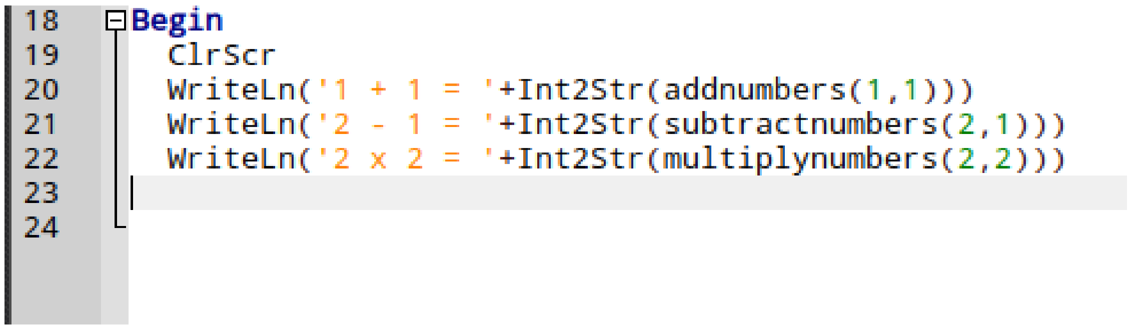
-=-=-=-=-=-=-=-=-

Also, depending on the IDE you use to type in the code, there will be lines to the left of the line numbers that indicate where your code blocks start and end.



```
Function ReadCall:Boolean
Var Ret    : Boolean = False
Var Fptr   : File
Var counter:Byte=1
Begin
  Callerdat:=CFGDataPath+'callers.dat'
  fAssign(Fptr,Callerdat,66)
  fReset(Fptr)
  If IOResult = 0 Then Begin
    For counter:=1 to 10 do
    Begin
      If Not fEof(Fptr) Then Begin
        fReadRec(Fptr,Call[counter])
      End
    End
    Ret:=True
    fClose(Fptr)
  End
  ReadCall:=Ret
End
```

If you notice the black lines between the line numbers and the code, you will see these correspond to the different code blocks. Keep an eye on these lines, as your typing your code. If you see something like:

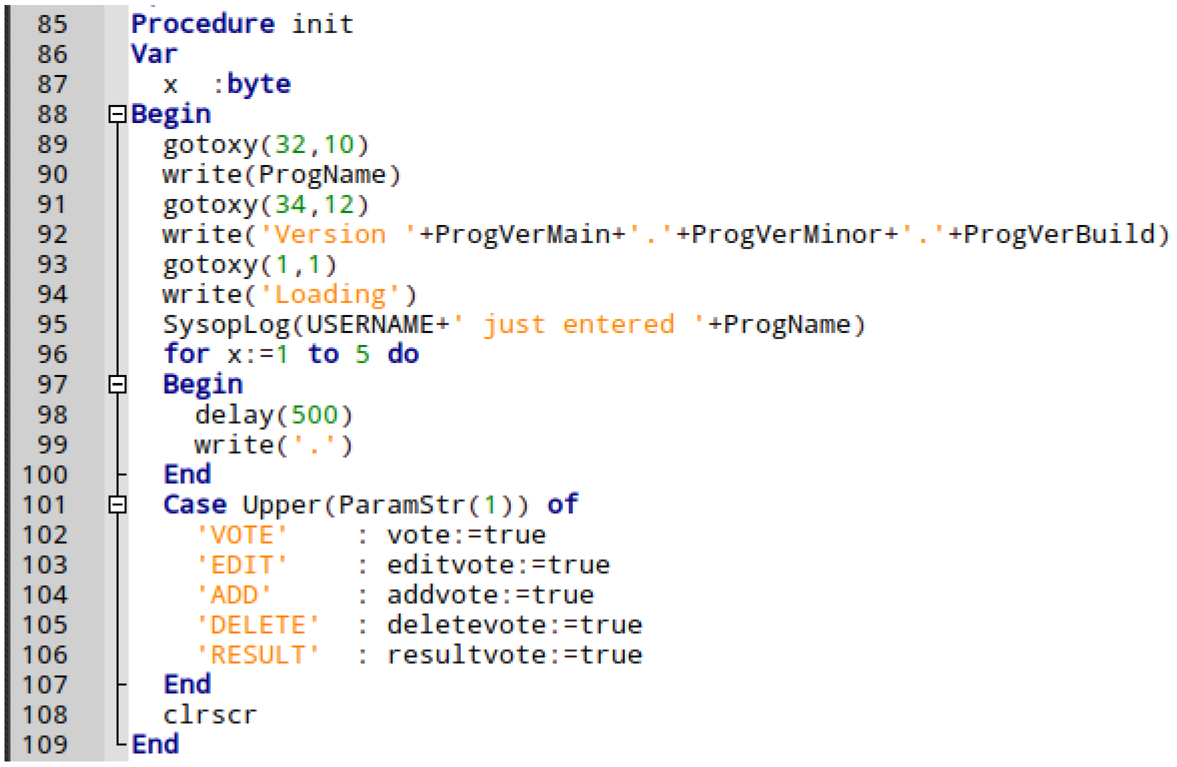


```
Begin
  ClrScr
  WriteLn('1 + 1 = '+Int2Str(addnumbers(1,1)))
  WriteLn('2 - 1 = '+Int2Str(subtractnumbers(2,1)))
  WriteLn('2 x 2 = '+Int2Str(multiplynumbers(2,2)))
```

...this, you know something is missing… Each of those lines should have a designated start and end point, and this one does not. This will help some of your compile errors.

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

When you first start typing a function or a new MPL program, as soon as you start using a variable, go up and declare that variable.



```
Procedure init
Var
  x   :byte
Begin
  gotoxy(32,10)
  write(ProgName)
  gotoxy(34,12)
  write('Version '+ProgVerMain+'.'+ProgVerMinor+'.'+ProgVerBuild)
  gotoxy(1,1)
  write('Loading')
  SysopLog(USERNAME+' just entered '+ProgName)
  for x:=1 to 5 do
  Begin
    delay(500)
    write('.')
  End
  Case Upper(ParamStr(1)) of
    'VOTE'     : vote:=true
    'EDIT'     : editvote:=true
    'ADD'      : addvote:=true
    'DELETE'   : deletevote:=true
    'RESULT'   : resultvote:=true
  End
  clrscr
End
```

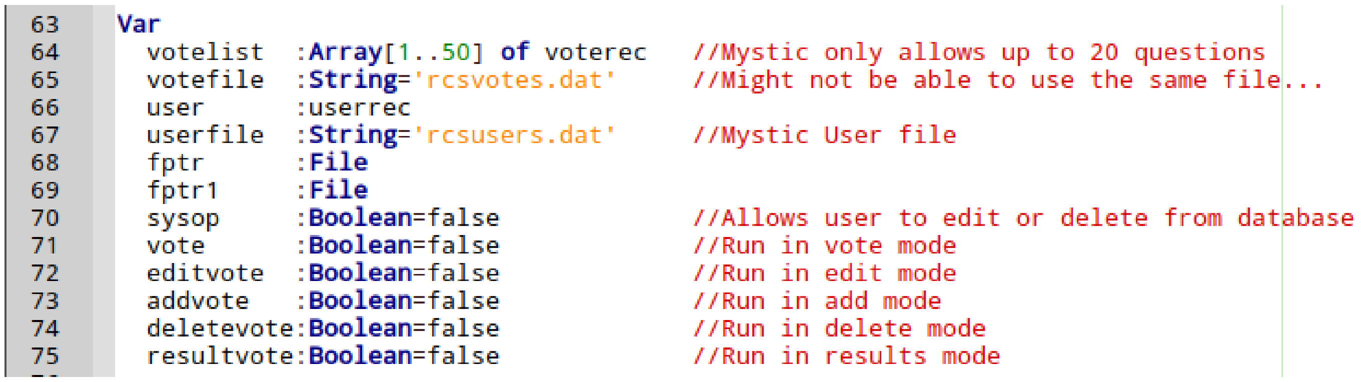
In this example, when I started typing this procedure, I got to ‘ProgName’, and went to the top and set up my ‘Const’ variable. When I got to ‘ProgVerMain’, I did the same. Each variable, as soon as I type it into a function or procedure, I immediately go up and declare the variable. That way you don’t have to worry about forgetting until you try to compile it.

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

I know I’ve mentioned this before, but I’m going to repeat it again. Always, always, always, comment your code!

While you are actively working on a project, you are going to remember what you were doing, and how you were doing it. But, once you start working on another project, and have to go back to fix a bug… I mean, and ‘undocumented feature’, you are not going to remember what or how your code works.

Here is an example of a program that I’ve been working on. This is my list of variables:



```
Var
  votelist  :Array[1..50] of voterec   //Mystic only allows up to 20 questions
  votefile  :String='rcsvotes.dat'     //Might not be able to use the same file...
  user      :userrec
  userfile  :String='rcsusers.dat'     //Mystic User file
  fptr      :File
  fptr1     :File
  sysop     :Boolean=false             //Allows user to edit or delete from database
  vote      :Boolean=false             //Run in vote mode
  editvote  :Boolean=false             //Run in edit mode
  addvote   :Boolean=false             //Run in add mode
  deletevote:Boolean=false             //Run in delete mode
  resultvote:Boolean=false             //Run in results mode
```

Even with variable names that tell you what they are going to do, it is a good idea to make yourself a note to remind you. It is always better to have too many comments in your code, than not enough. It can be very difficult to figure things out later.

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

Develop your own programming style. Find out what works best for you, and for debugging your code later on. For me, I found that using a two space indent, setting up programs in a specific way, will really help you. Also, if your style is a little different than other peoples, you will know when they copy and paste your code into their programs. :)

Remember, there really is no wrong way to program something. There may be better ways to do it, but if your way works, go with it! There will always be code that you go back to at a later date, and will be able to optimize and clean up the code.

Don’t try to emulate someone else when coding. Just because someone does a particular function a certain way, doesn’t always mean it’s the best way of doing it. There may be reasons why a programmer changes the way to accomplish a specific task. Perhaps the data file they’re reading is set up differently. Find your way!