

# So you want to code in Fenix?

A Beginner's Tutorial by EvilDragon with a lot of help by Josebita

## 1. What is this?

It's a **tutorial** if you want to **begin to code in Fenix**. Fenix is a programming language specially designed to easily **code 2D games**. An interpreter exists for various platforms (e.g. DC, PC, GP32). This tutorial is mainly aimed at GP32 users, but can also be used to learn Fenix if you want to code for a different platform.

While writing this Tutorial, I am myself trying to learn Fenix – so basically it's a WIP Tutorial which will grow while I learn :)

## Chapter 2 – First Coding Steps

Hey, guess what?!

Finally, after setting up the environment and knowing some basics about processes (which can have multiple instances) and variables, we finally start to code! Yay!

### 1. „Hello World!“ - Explained

Remember the program we used for testing our installation? The scrolling „Hello World“?

```

Hello World.prg
01 PROGRAM Hello;
02 Begin
03     x = 0;
04     while(x < 320)
05         delete_text (0) ;
06         x += 2;
07         write (0, x, 100, 1, "Hello World!") ;
08         FRAME;
09     End;
10 End;
```

Let's take a closer look at that:

In Line number 1 we have a **PROGRAM Hello;**

This is the name of the program. It should always be the first line. Note the ; at the end of the line?

Each command concludes with ;

**Remember that. It's essential.**

Most bugs in a program are a missing ;

Onto line number 2. Here, we have **Begin**

Each block begins with a **Begin**. Note that there is **NO ;** here. Why not, you ask?

Well, the Begin marks the beginning of a block. The block ends with an **End;**

That's why the **;** is **only at the END** of the block. Same thing goes for the while-command in line 4.

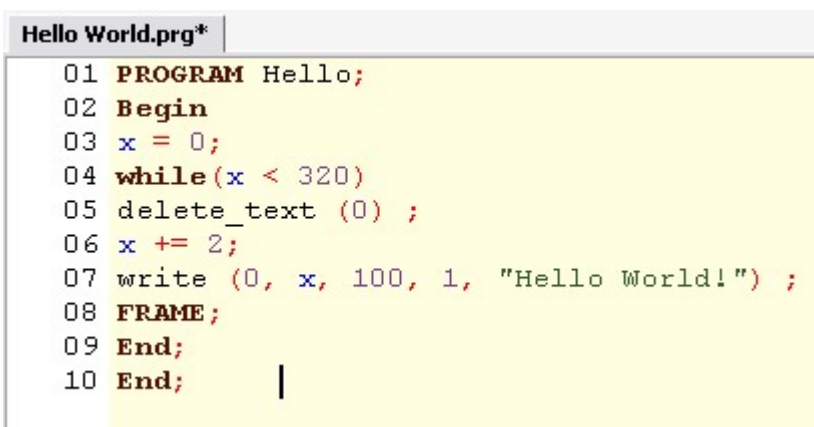
The block structure is always like this:

```
BEGIN (Block 1)
  code
  code
  BEGIN (Block 2)
    code
    code
  END (Block 2)
  code
  code
END (Block 1)
```

So, the **FIRST beginning ALWAYS concludes with the last END;** the other blocks are inbetween.

**Don't forget that!**

**Coder's tip:** Always use the tab-key to mark your blocks. It gets really messy if you don't. Look at this picture:



```
Hello World.prg*
01 PROGRAM Hello;
02 Begin
03 x = 0;
04 while (x < 320)
05 delete_text (0) ;
06 x += 2;
07 write (0, x, 100, 1, "Hello World!");
08 FRAME;
09 End;
10 End; |
```

It's the same code as above – only without tabs. You don't see with one look where the block begins and where it ends. Not much of a problem in this small program, but when your programs get longer and have a lot more blocks, it's messy. Believe me.

Okay, next: **Line 03.**

This line just sets the variable x to 0.

The command ends with this line, so there is an **;**

**Line 04:**

while (x < 320)

There is **no ;** here – because the while-command **ends with line 09.**

This means: While the variable x is lower than the number 320, line 04 – 08 repeat themselves (a new block starts with the „while“ in line 04 and ends with the „End;“ in line 09).

**Line 05:**

This **deletes a text** written with **WRITE-command** in this instance of the process. **The (0)** tells, that ALL texts will be deleted. You can also use an identifier to delete only a specific text. If you remove the line, the „Hello World“ won't scroll – but you will have a „Hello World“-Trail. Try it out, if you want.

**Line 06:**

```
x +=2
```

This **adds 2** to the variable **x**. (so, if it was 0, it'll be 2, if it was 2, it'll be 4, etc.)  
*Remember: If x is bigger than 320, the block ends (as we defined that with the WHILE-command. That's why the program ends as soon as the „Hello World“ reaches the end of the screen)*

**Line 07:**

```
write (0, x, 100, 1, "Hello World!" ) ;
```

This **writes the text** on the screen.  
 The command looks like this:

```
write (text id, x, y, alignment, „Text!“ ) ;
```

The **text id** is used to identify the text. If you set it to 0, it will use an internal counter. The first written text can be referred to as 1, the second as 2, etc.

Then we have the coordinates, x and y. This defines the position of the text on the screen. The **GP32 resolution is 320\*240**. 0,0 is top left corner, 320,240 is lower right corner. We have **x, 100** in our line. The **x** in our program is a **variable** which increases (that's why the text moves) until it reaches **320** (*the right end of the GP32 screen*), the **y** is 100, which should be a bit above the middle.

Next is **„alignment“**. This tells *Fenix* how to **align the text** (*align left, right, center, etc.*)  
 You can lookup the different possibilities in the **Fenix Reference**.

**Line 08:**

```
FRAME;
```

A **VERY IMPORTANT COMMAND** in *Fenix*! Remove it and start the program. NOTHING will happen. Why?

Well, **ALL commands which show something on the screen**, ONLY get executed when the FRAME-command is run!

With the **WRITE-Command** in Line 07 we define **WHAT AND WHERE the text should be** – but **without a FRAME-command**, it **won't be shown!**

It's the same for graphics and sprites, **DON'T EVER FORGET THAT!**

**Line 09:**

This marks **the end of the „While“-block**. After the **x >= 320**, the block ends and the program continues with the next line (line 10)  
 While **x < 320**, the „while“-block loops.

**Line 10:**

This marks the end of the „Begin“-block and thus the end of the program.

Okay, so in conclusion, **this is what our program basically does:**

It **writes a text** at the coordinates **x** (variable) and **100** on the screen.

The **x-Variable** is 0 at first (*left side of the screen*) and **increases by 2 pixel** with every loop in the **while-block**.

The **text gets deleted everytime** and **written again** with a new x-coordinate.

That's why the text moves.

When **x = 320** (*the right of the screen*), the **loop ends and the program ends**.

Nice, eh?

Well, as nice as the program is, **it has one problem:** The „Hello World“ **flickers**. That's because the **text gets deleted** and **newly written everytime**.

Now we're gonna change that!

If you lookup the „**WRITE**“-command in the **Fenix reference**, you'll find a crosslink to **MOVE\_TEXT**

This sounds like it could be used for moving text, eh?

Let's look how the command works:

**MOVE\_TEXT ( TEXTID, X, Y )**

Okay, *text id* is the value for **selecting the text** (*if you use the internal counter, the first write-command gets nr. 1, the second nr. 2, etc.*). **X, Y** are the new coordinates (*in our case x, 100*)

So **instead of deleting and rewriting** the text, we can **move** it.

This should remove the flickering and saves resources.

**Your task is: Change the code so that we use the MOVE\_TEXT-command instead of WRITE and DELETE.**

Can you do it? It's not THAT hard – but you also need to **think a bit**.

Check the next page after you did it! :)

Did you manage to do it? The code should look like this:

```

Hello World.prg* | HELP: WRITE | HELP: WRITE | HELP: MOVE_TEXT | FENIX HELP
01 PROGRAM Hello;
02 Begin
03   x = 0;
04   write (0, 0, 100, 1, "Hello World!") ;
05   while(x < 320)
06     x += 2;
07     move_text (1, x, 100);
08     FRAME;
09   End;
10 End;
```

What did I change? Well, the **write**-command is now **BEFORE the loop**, as we only need to write the text **ONCE** (we're moving it later).

The **DELETE\_TEXT**-command is gone – as **we don't need to delete the text anymore**.

In the loop, we now have a **MOVE\_TEXT** command with the **text id** (1, because, like I said, the first **WRITE**-command can be referenced as 1, the second as 2, etc.) and the new coordinates.

*BTW: If you want to smooth the moving, only add 1 to x in the loop (x +=1;)  
This makes it smoother but slower – but you can change the speed if you add a value behind the **FRAME**-command, i.e. **FRAME 50**;  
Try it :)*

**Understood everything? Good :)**

Well, now I want you to **ADD another text** – it should be somewhere in the **top right corner** of the screen and **unlike the first text, it should NOT move**.

The text can be anything you like. If you want, you can also give it a different color. (how to do that? Just follow the cross-links from the **WRITE**-command. EVERY command used in combination with can be found there!)

**See a new version of our script at the next page!**

```
Hello World.prg | HELP: WRITE | HELP: WRITE | HELP: MOVE_TEXT | FENIX HELP | I
01 PROGRAM Hello;
02 Begin
03   x = 0;
04   write (0, 0, 100, 1, "Hello World!") ;
05   SET_TEXT_COLOR (25);
06   write (0, 250, 10, 1, "GP32 rules!") ;
07   while (x < 320)
08     x += 1;
09     move_text (1, x, 100);
10     FRAME 25;
11   End;
12 End;
```

That was an easy task, wasn't it? Well, let's look at our changes:

**Line 05:**

SET\_TEXT\_COLOR (25);

This changes the text color of **ALL following write-commands** to color nr. 25 (*you can define these colors using a FNT-File. We will learn that later!*)

**Line 06:**

This writes the text „GP32 rules!“ at the top right corner of the screen.  
(*Note: the internal text id for this text is 2 – that's why this text won't be affected by the move\_text-command, which only moves texts with id 1*)

**LINE 08 and LINE 10:**

Now we're only adding 1 instead of 2 to the x-variable per loop, making the text move slower. The 25 behind the FRAME command makes it run a bit faster again.

**Okay, your last task for today is:**

Make the „GP32 Rules!“ - Text **move from right to left**, same speed as we move „Hello World!“ from **left to right**.

(*Note: „GP3 Rules!“ can start off-screen. This makes it pretty easy*).

Try it – and find the **easiest solution at the next page**.

Got it? Well, here's my easy solution:

```
HELLO World.prg | HELP: WRITE | HELP: WRITE | HELP: MOVE_TEXT | FENIX HELP | F
01 PROGRAM Hello;
02 Begin
03   x = 0;
04   write (0, 0, 100, 1, "Hello World!") ;
05   SET_TEXT_COLOR (25);
06   write (0, 320, 10, 1, "GP32 rules!") ;
07   while(x < 320)
08     x += 1;
09     move_text (1, x, 100);
10     move_text (2, 320-x, 10);
11     FRAME 25;
12   End;
13 End; |
```

### I changed Line 06:

Only the coordinate gets changed, so that „GP32 rules!“ starts at x-coordinate 320 (rightmost off the screen).

### I added line 10:

This line moves text with id 2 (the second write-command).

As we need to move from **right to left** instead of **left to right**, we simply take the **maximum value** (320) and subtract **variable x** from it.

When  $x = 0$ , „Hello World!“ will be at 0,100 and „GP32 rules!“ at 320,10.

When  $x = 100$ , „Hello World!“ will be at 100,100 and „GP32 rules!“ at 220,10

When  $x = 320$ , „Hello World!“ will be at 320,100 and „GP32 rules!“ at 0,10

Easy, wasn't it?

Well, **this ends our lesson for today**. You can fiddle around a bit, move texts around as you want. The more you try, the better you will get :)

What should you have learned in this chapter?

**a) What blocks are.**

**b) Each command ends with ;**

**c) USE THE FENIX REFERENCE! You'll find a lot of useful commands there!**

That's it! See you soon,

**EvilDragon.**