

Beginning Programming With KPL

By Jon Schwartz

Document updated November 16, 2005

website: www.kidsprogramminglanguage.com

Beginning Programming With KPL

Table of Contents

| | |
|--|-----------|
| Introduction: What is Programming?..... | 3 |
| Why Should I Learn to Program With KPL? | 3 |
| How Should I Use This Tutorial? | 3 |
| OK, Show Me a Program!..... | 4 |
| What about Computer Graphics?! | 6 |
| Game Graphics Using Sprites | 11 |
| Using Variables and Loops in KPL | 13 |
| Using KPL on Your Computer | 16 |
| Next Steps After the Tutorial | 23 |

Introduction: What is Programming?

Computer programming is, simply put: **giving instructions to a computer.**

Computers are very good at following instructions. They do exactly what we tell them to do. But they have **no imagination!** And so when we write programs to give computers instructions, we must tell them very precisely what we want them to do.

Why Should I Learn to Program With KPL?

Different computer languages give you, the programmer, different ways to tell your computer what you want it to do. Kid's Programming Language, or KPL, has several important advantages for you as a beginner:

- KPL was carefully designed to make it **as easy as possible** for a beginner to learn.
- KPL was carefully designed to make it **as fun as possible** for you to learn
- KPL, unlike other learning languages, is also carefully designed to make it **as similar as possible** to the languages which are used by professional programmers today

People keep repeating wise old sayings because they are true. The wise old saying to keep in mind about KPL is: **we learn to walk before we run.** Programming with KPL is learning to walk. After you learn KPL, you'll have a **much** easier time learning to run – whether you decide to run with Java, Python, Visual Basic or C#.

How Should I Use This Tutorial?

Very Important Point: This tutorial is written based on improvements to KPL which are in KPL versions released on or after October 10th. If you downloaded KPL before October 10th, or if the examples don't work as described when you type them in, please download and install the latest version of KPL from <http://www.kidsprogramminglanguage.com/download.htm>.

If you are truly a beginner, and have never done any programming, the best way to use this tutorial is to read and study this tutorial section by section, in order, making sure you understand each section before moving on to the next. It will probably be best to study the tutorial without using KPL on your computer, until you get to the section of the tutorial **"Using KPL on Your Computer"**. Studying and learning KPL this way will help you avoid being distracted by details which you will get to at the end of the tutorial.

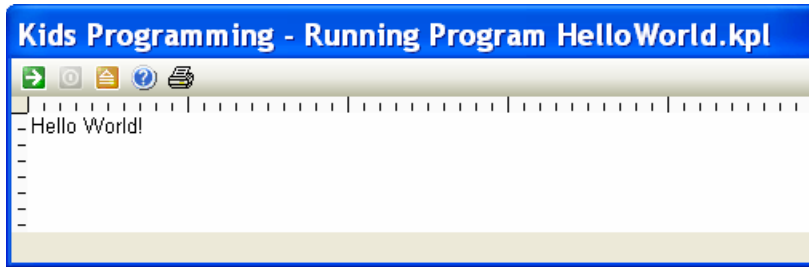
Computer programming involves learning to think in ways people do not normally think. Computers require us to be much more logical, orderly and precise than we normally need to be! Learning this may be difficult at first, but you can do it! And once you get the hang of it, it really does become fairly easy.

One of the best ways to learn that new way of thinking is to ask questions of or get explanations from someone who already understands computer programming. Can you think of anyone who could help you with questions and answers as you study programming with this KPL tutorial?

OK, Show Me a Program!

```
1 Program HelloWorld
2 Method Main()
3     Print("Hello World!")
4 End Method
5 End Program
```

Well, that's it! When you run that KPL program, this is what you will see:



The 5 lines of KPL code you see above are a screenshot of what the code looks like in KPL, so you can see it here exactly as you will see it when you are programming in KPL.

The first thing I should mention is that the line numbers you see to the left of each line of code are not actually used by KPL – they are just there for your own information as you read and work with your KPL programs. Older languages, such as GWBASIC, actually used line numbers to process code in the order desired by the programmer. KPL does not do this. KPL programs are normally processed one instruction at a time, starting at the top and moving downward through the program. There are exceptions to this, which we won't discuss in this beginner's tutorial.

One important rule for programming in KPL is that each line of KPL code needs to be on a separate line in your program file. For instance, this KPL program has the same code in it, but not on separate lines, and so this program will not work:

```
1 Program HelloWorld Method Main() Print("Hello World!") End Method End Program
```

All computer languages have some rules which the programmer has to follow, so that the computer will understand the programmer's instructions. Person-to-person communication has lots of rules, too, that serve the same purpose – it's just that we are so used to those rules that we follow them without having to think about them. For instance, we don't say "Goodbye" when we pick up the phone! And we don't say "Hello" when we hang up the phone! Sure, it's a silly example, but it's very relevant, since KPL also requires a specific way for beginning and ending a program. All KPL programs must begin with a line like **Program HelloWorld** as shown in line 1. And all KPL programs must end with **End Program** as shown in line 5:

```
1 Program HelloWorld
2 Method Main()
3     Print("Hello World!")
4 End Method
5 End Program
```

You can name your program anything you want, but it's best to name it something which describes what the program is doing. In this case **HelloWorld** is the name I chose. You could just as easily use a different name, like **Program MyFirstProgram**.

Method Main() is the next important thing to know about KPL programming. All KPL programs start by processing the first instruction in **Method Main()**. Take a look below and you will see that instruction is **Print("Hello World!")**.

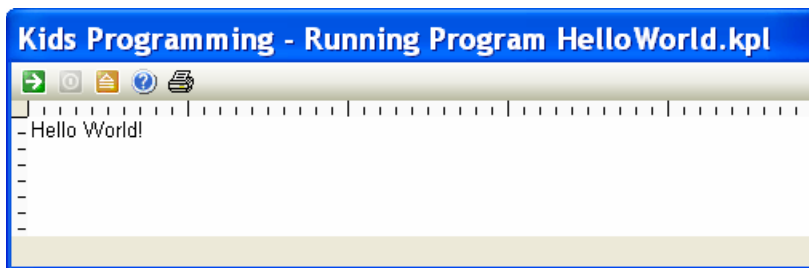
Method Main() is a bit of an arbitrary way to define where the program begins processing instructions – but it is, in fact, based on the way all modern programming languages work. As you might expect logically, **End Method** matches and declares the end of **Method Main()**.

We won't explain Methods further in this tutorial, but for now, it will hopefully make sense when you look at the code below that there is only one instruction "in" **Method Main()**, and that is the line **Print("Hello World!")**.

All this was a detailed explanation of why this particular program contains only one actual instruction, the one which tells the computer to **Print("Hello World!")**. So, let's summarize: Lines 1 and 5 tell the computer that they are the beginning and the end of this KPL program. And lines 2 and 4 tell the computer that they are the beginning and end of **Method Main()**.

```
1 Program HelloWorld
2   Method Main()
3       Print("Hello World!")
4   End Method
5 End Program
```

Now let's look again at the window which appears when you run this program. Notice that "inside" the window, the computer has indeed done only one thing, the thing we told it to do with this KPL program. The computer has displayed the words **Hello World!**



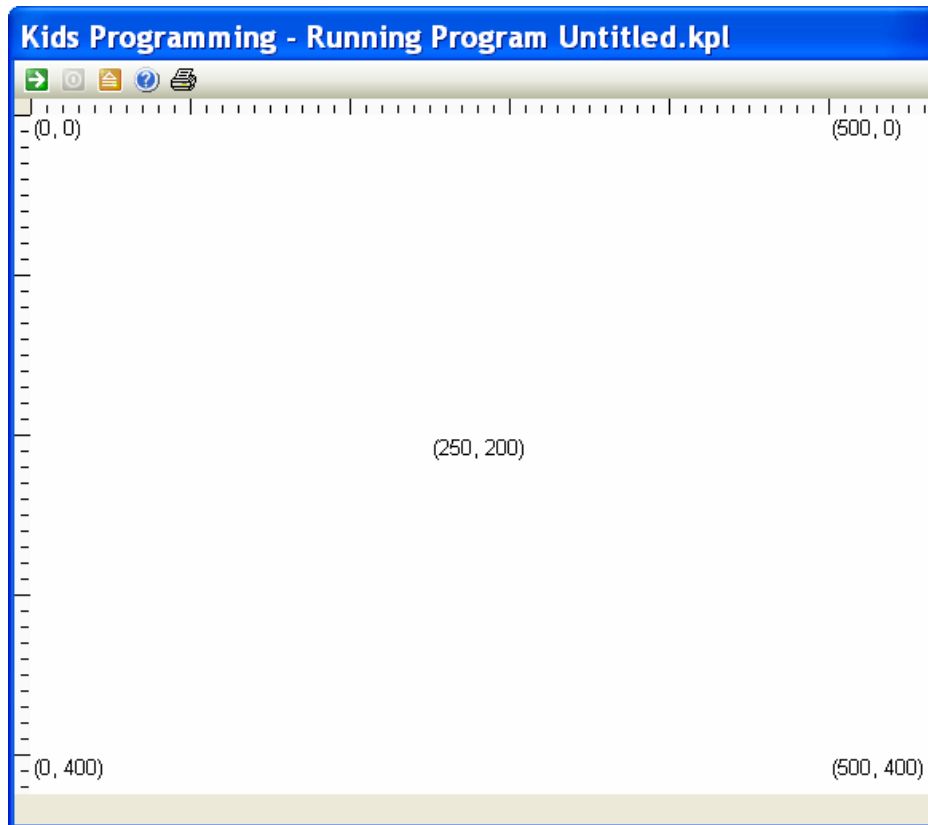
What about Computer Graphics?!

Hello World! is a classic first program – but it's not very exciting! So let's talk about graphics.

An explanation and example of computer graphics has to start with where computers place graphics on the screen. Computers use a **coordinate system** which is different from the **algebra** coordinate system which we all learn in school. But actually, the computer coordinate system can be easier to work with, particularly because it makes it simpler to work with locations on the computer screen.

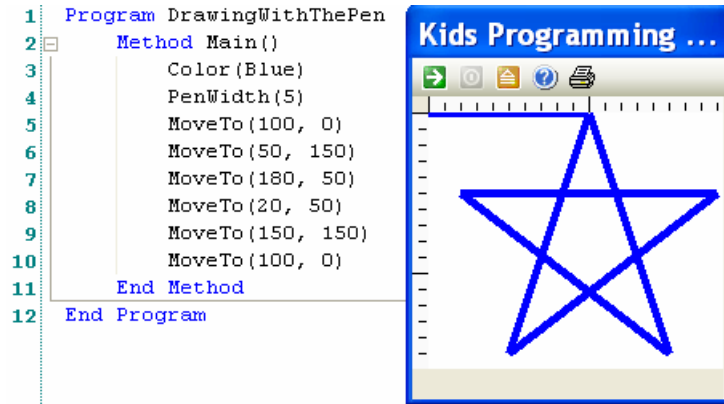
Computers use an **(X, Y)** coordinate system for locations on the computer screen in which **the left edge of the screen defines $X = 0$** , and **the top of the screen defines $Y = 0$** . This means that the **origin**, where **$X = 0$** and **$Y = 0$** , is the **upper left corner** of the screen. Moving to the right across the screen increases the X value, and moving down the screen increases the Y value.

If you are not used to thinking about coordinate systems, this is a complicated explanation, so here is a picture which shows where several **(X, Y)** locations are displayed by a KPL program:



Please take a moment to examine the numbers in the picture above. The **first value** of each pair is the **X value** of that location on the screen – and as you can see, the X value increases as you move from the left to the right. The **second value** of each pair is the **Y value** of the location on the screen – and as you can see, the Y value increases as you move from the top to the bottom.

Let's write our first KPL graphics program, and in the process see exactly how you use the graphics coordinates system to make cool graphics on the screen with KPL. We'll start by showing the full KPL program, as well as the graphics it creates when you run it. And then we will step through the KPL program in detail, to explain exactly how it works:

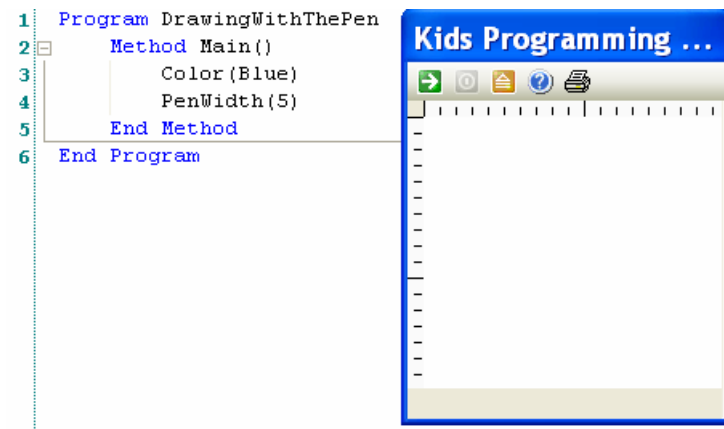


Notice there are now 8 KPL instructions in **Method Main()**, from line 3 to line 10. That's a lot more than our first example, but hopefully you agree it's cool that KPL can draw a blue star on the computer screen like that with only 8 instructions!

Notice that the program begins and ends in the same way our first program did, except that we have named this program **DrawingWithThePen**:

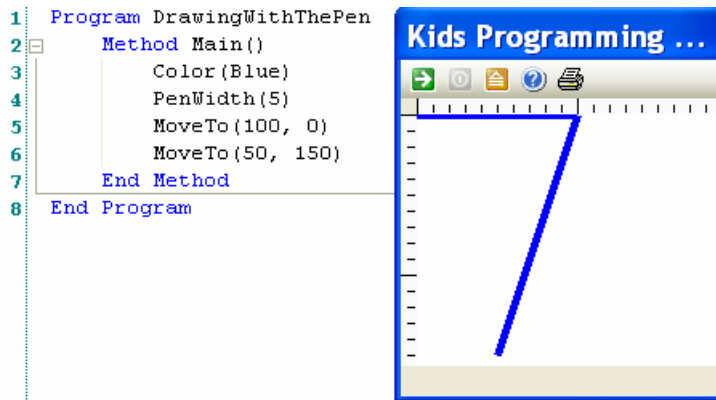
```
1 Program DrawingWithThePen
2   Method Main()
3
4   End Method
5 End Program
```

Now let's start adding our instructions to **Method Main()**, to examine what they each do:



Our first two instructions have been added to our program, but as you see, the program doesn't display any graphics yet. **Color(Blue)** tells KPL that we want it to draw using the color Blue. **PenWidth(5)** tells KPL that when we draw with the Pen, we want KPL to draw a line which is 5 pixels wide. You could, of course, use a different value for PenWidth. **PenWidth(2)** would draw a thinner line, and **PenWidth(10)** would draw a thicker line. So far, we have told KPL

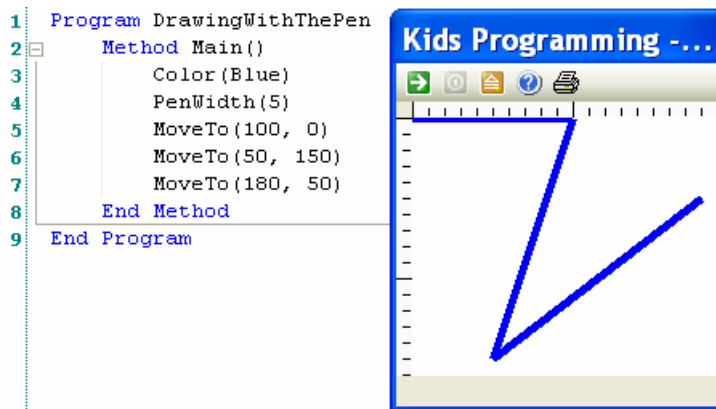
about how it will draw, but we haven't told it to draw anything yet. Now, let's draw our first line in the star:



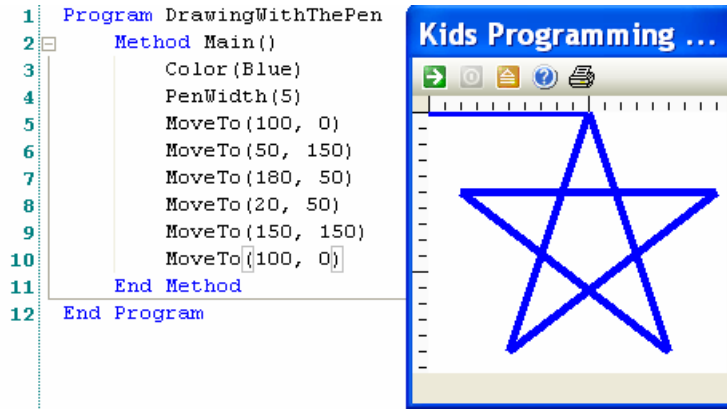
The first instruction we added is **MoveTo(100, 0)**. This tells KPL to move the pen to location (100, 0) on the screen, which is the top of our star. Since KPL always starts with the pen at location (0, 0) – the upper left corner – moving the pen to location (100, 0) draws the horizontal line you see at the top as it moves.

The second instruction we added is **MoveTo(50, 150)**. This instruction actually draws the first line in our star. Consider the (X, Y) values for these two points for a moment. Since this instruction tells KPL to move the pen from X = 100 to X = 50, the line moves to the left. And since we move from Y = 0 to Y = 150, the line moves down the screen.

Now let's add one more instruction to KPL, which adds the next line to the star:



The instruction we added is **MoveTo(180, 50)**. KPL continues moving the pen from the previous point, which was (50, 150). Have you ever seen an Etch-A-Sketch – those cool red toys which allow you to draw things on the screen by turning two knobs? KPL's pen is a lot like a computerized Etch-A-Sketch. Let's add the remaining lines which will finish the star now:

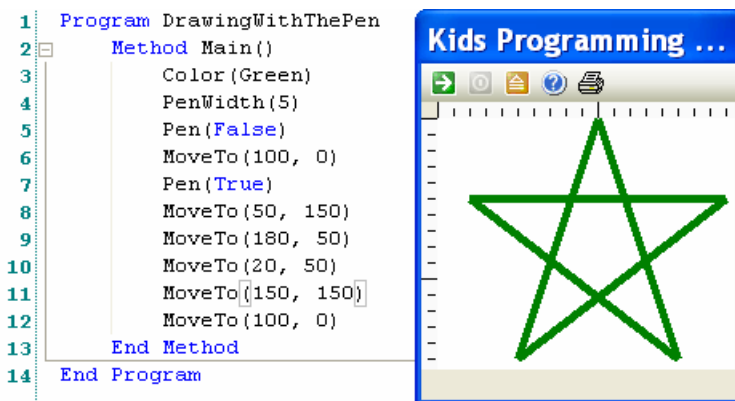


As you can see, three more KPL instructions cause KPL to draw three more lines. In total, we have moved the Pen 6 times, drawn six lines, and the result is a star, drawn according to the instructions in our KPL program.

The KPL program required to do this is small – only 8 KPL instructions. The hardest thing about this is getting used to the way the (X, Y) coordinate system works in KPL. At this point, if (X, Y) values aren't clear to you yet, you probably should return to the beginning of this section and read through it one more time. If after you do that (X, Y) coordinates still don't make sense, you may want to type your own version of this KPL program into KPL on your computer, and experiment with it by drawing lines to and from different locations on the screen. Perhaps you could practice by drawing a square, and a triangle? If you do that with KPL on your computer, you can skip to the section **Using KPL on Your Computer**, for help using KPL on your computer. After you are comfortable with (X, Y) coordinates, you can return to this point in the tutorial.

It's important to spend enough time and effort so that you are very comfortable with how KPL handles (X, Y) coordinates, since these are the basis of all graphics programming. Other languages handle graphics this same way, so when you learn this with KPL, you are learning the most fundamental concept needed for computer graphics programming in any language.

Let's add a final detail to our example, which also shows you an extra bit of control that you have over KPL's Pen. This is something you can't do this with an Etch-A-Sketch! You also can't control the color of an Etch-A-Sketch, so let's show how easy KPL makes that, too:

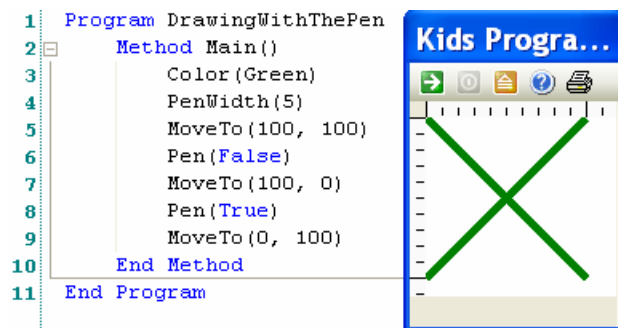


As you can see, we just changed **Color(Blue)** to **Color(Green)**. Easy! Also, the first statement we have added is **Pen(False)**, just before **MoveTo(100, 0)**. **Pen(False)** simply

tells KPL that it is not supposed to draw with the pen when it moves it. So now, when the pen moves from (0, 0) to (100, 0), it no longer draws the horizontal line to the top of our star – and our star looks better!

Remember that computers require very specific instructions, and when we said **Pen(False)**, KPL stopped drawing when the pen moves. So what do we have to do now that we want KPL to draw the lines which make up our star? We need to give KPL the instruction: **Pen(True)**, so it starts drawing again as it moves the pen.

To summarize, **Pen(False)** tells KPL to turn drawing “off” as it moves the Pen, and **Pen(True)** tells KPL to turn drawing “on” as it moves the Pen. Etch-A-Sketch can’t do that! Turning the pen off and on as you move it allows you to draw all kinds of objects, and to draw multiple objects. Here’s a very simple example which can best be drawn if you turn the Pen off, move it, and then turn the Pen back on again:




Game Graphics Using Sprites

The point of the drawing example was mostly about explaining how (X, Y) coordinates are used for displaying graphics. But game graphics aren't normally drawn. They are loaded from an image file, like an image of a UFO or an asteroid or an elf with a bow. KPL uses Sprites as a way to make this **very** easy to do!

As we did with the last program, we are going to start by showing you the finished example, and then explain in detail how this program works, so that you can write programs like this yourself. Here is the full KPL program – which if you look inside **Method Main()** has exactly 8 instructions, just like the last example. These 8 instructions, though, tell KPL to display a UFO as shown at the top of the window, and move it slowly down the screen to where you see it below:

```
1 Program UFO
2
3 Method Main()
4
5     LoadSprite( "UFO", "UFO.GIF" )
6     MoveSpriteToPoint( "UFO", 50, 0 )
7     ShowSprite( "UFO" )
8
9     Define ufoY As Int
10    For ufoY = 1 To 150
11        Delay(10)
12        MoveSpriteToPoint( "UFO", 50, ufoY )
13    Next
14
15 End Method
16
17 End Program
```



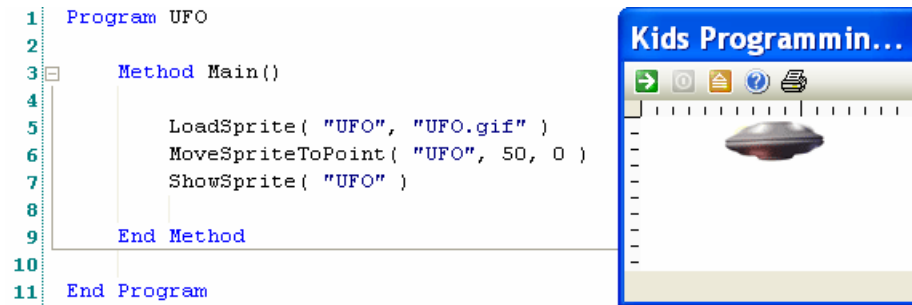
Notice that this program begins and ends the same way as our previous examples – and as all KPL programs have to – except that we have named this one **UFO**:

```
1 Program UFO
2
3 Method Main()
4
5 End Method
6
7 End Program
```

Let me show you some examples of image or picture files which are included with KPL. You can use **ANY** image file you want to use with KPL, including ones you add or create – these are just a few of the 65 images which are included with KPL:



Now that you've seen some examples of the kinds of graphics you can use with KPL, let's start by adding the KPL instructions which cause the UFO to be displayed at the top of the screen:



The first instruction we added is `LoadSprite("UFO", "UFO.gif")`. As I mentioned before, KPL has some rules about how we give it instructions. The rules about `LoadSprite` are not difficult, but they are exact! `LoadSprite` requires two "values" for it to work. The first value must be the name by which you will refer to the Sprite – in our case, `"UFO"`. And the second value must be the name of the image file which KPL will use for this sprite – in our case, `"UFO.gif"`. These values must be surrounded by quotation marks, as shown. And you must separate them with a comma, as shown.

Here are a few examples which show **incorrect** ways of telling KPL to load this Sprite. Let me repeat: **each of the four instructions below will not work** because they do not follow KPL's rules about calling `LoadSprite`. Can you figure out what to change to make these work?

```
LoadSprite( 'UFO', 'UFO.gif' )
LoadSprite( UFO, UFO.gif )
LoadSprite( "UFO" . "UFO.gif" )
LoadSprite( "UFO" )
```

Summary: `LoadSprite("UFO", "UFO.gif")` tells KPL to make a new sprite using the picture from file `"UFO.gif"`, and to name the sprite `"UFO"`.

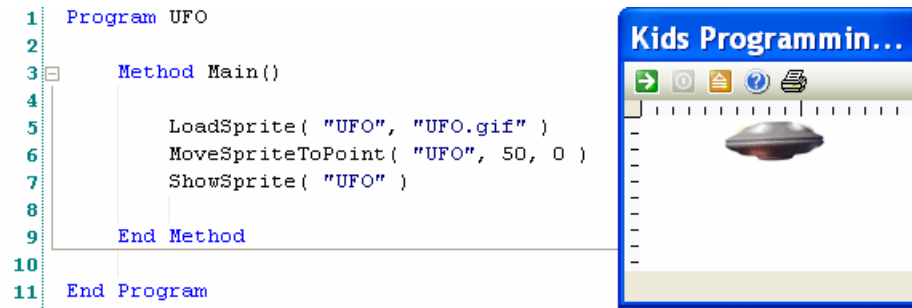
KPL now needs to know where to put the sprite on the screen, so we give it the instruction `MoveSpriteToPoint("UFO", 50, 0)`. `MoveSpriteToPoint` requires three values. First is the name of the sprite, which we know is `"UFO"` because that's what we named it with `LoadSprite`. Second is the **X axis location** for the sprite, which we want to be 50. Third is the **Y axis location** for the sprite, which we want to be 0.

An important detail to notice is that **there are no quotation marks around the numeric values**. In general, KPL **requires** quotation marks around values that are **words**, and KPL **does not require** quotation marks around values that are **numbers**. See the **KPL User Guide** section on Data Types for more detailed information about this.

Summary: `MoveSpriteToPoint("UFO", 50, 0)` tells KPL to move the sprite named `"UFO"` to the screen location (50, 0).

All that remains is to tell KPL to display the sprite. That's the easiest instruction yet: `ShowSprite("UFO")`. That's it! Our UFO is on the screen where we told KPL to put it!

That was a very detailed explanation, so let's get back to showing how simple it actually is in KPL. Here's the KPL program, and what it looks like when we run it:



Using Variables and Loops in KPL

Now we want our UFO to "land" by moving down the screen to the bottom of the window. To do that we are going to define and use our first ever "**variable**," so that you can see how variables work in a KPL program.

The term variable is very descriptive – it means that the value can change over time. The value changing over time is the real power of using a variable, as you will see in this example. Variables should be defined with a name that makes sense given how we are going to use them. In this case, our variable is going to be used to change the Y-axis location of the UFO, to make the UFO move down the screen and "land," so we will logically call the variable **ufoY**.

Here is the line of code defining our variable:

```
Define ufoY As Int
```

Define is a KPL keyword that lets KPL know you are defining a new variable for KPL to use. **ufoY** is the name we are going to use for the variable. **As Int** tells KPL that **ufoY** is being defined as an Integer variable. **Int** is an abbreviation for Integer. Integer variables hold numeric values, like **-1** or **0** or **43**. Integer values are whole numbers only.

We know that our UFO starts at location **(50, 0)** on the screen, because that is the point where we told KPL to move it. How do we make it move down the screen? We do that by increasing its **Y-axis location** – to **(50, 1)** then **(50, 2)** then **(50, 3)** then **(50, 4)**, etc... It's up to us how far we want it to go, so let's decide that the UFO will stop when it gets to **(50, 150)**.

Do you see the pattern there? The **X** value of the sprite's location is always **50**. The **Y** value of the sprite's location is going up by **1** pixel at a time, from **0** all the way to **150**. It's important for you to see the pattern before we show you how to use that pattern in KPL code – so if the pattern doesn't make sense, please read the last paragraph again.

Here is the KPL code which tells KPL to increase the value of the variable **ufoY** from 1 to 150, 1 step at a time:

```
For ufoY = 1 To 150
    MoveSpriteToPoint( "UFO", 50, ufoY )
Next
```

This is your first “**loop**.” Loops are another of those new concepts which you will need to learn if you have not programmed before – but once you understand the concept, loops are easy.

This loop starts with the value of **ufoY** = 1. Since we also said **To 150**, we know the loop will stop after it gets to the value of **ufoY** = 150. Notice the keyword **Next**, which defines the end of the loop. When KPL gets to the keyword **Next**, KPL knows that it is time **increase the value of ufoY** to the next value – from 1 to 2, or from 2 to 3, or from 3 to 4, etc..., all the way to from 149 to 150.

Basically, we are telling KPL to count from 1 to 150, and we want KPL to use our variable ufoY to keep track of the value as it counts.

And what do we want KPL to do while it is counting? We want it to move the UFO down the screen, right? That’s what **MoveSpriteToPoint("UFO", 50, ufoY)** tells KPL to do.

Since this instruction is “inside” of the **For** loop, KPL will perform this instruction **each time it counts** from 1 to 150. This is the real power of a loop, of course. Just counting from 1 to 150 is not so interesting, but if you can **do something useful each time you count**, now **that** lets us do all kinds of cool things, including move our UFO down the screen! Let’s examine what is happening:

```
For ufoY = 1 To 150
    MoveSpriteToPoint( "UFO", 50, ufoY )
Next
```

We have already seen how **MoveSpriteToPoint** works: KPL moves the sprite **"UFO"** to the (X, Y) location we give it. What’s different this time? Before, we moved the UFO to (50, 0). What happens when we do this in the loop, and we instead use our variable **ufoY**? Remember that the first time through the loop, the value of **ufoY** = 1, and the second time **ufoY** = 2, then **ufoY** = 3, then **ufoY** = 4, etc... all the way to **ufoY** = 150. So, the first time through the loop, KPL uses the value of **ufoY=1** to move the sprite, so that the instruction KPL performs is effectively:

```
MoveSpriteToPoint( "UFO", 50, 1 )
```

And the next time through the loop, **ufoY** = 2, so KPL performs:

```
MoveSpriteToPoint( "UFO", 50, 2 )
```

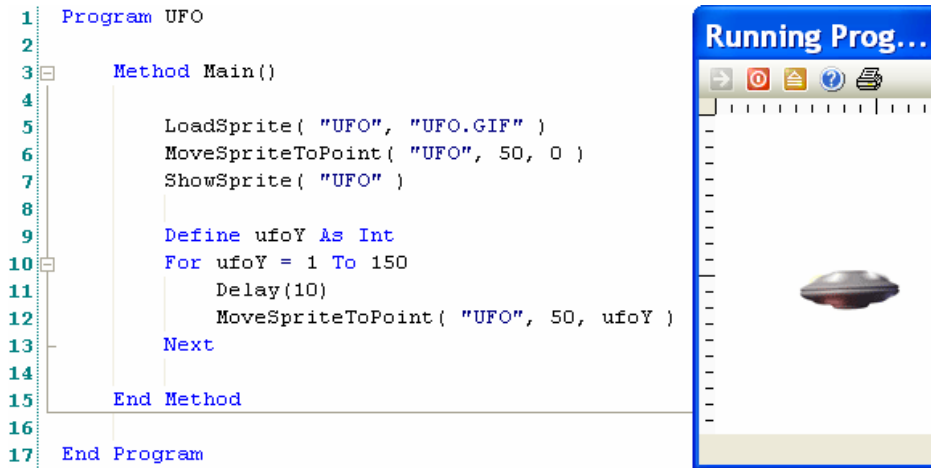
And so on like this, until KPL has used the loop to count all the way to 150:

```
MoveSpriteToPoint( "UFO", 50, 3 )
MoveSpriteToPoint( "UFO", 50, 4 )
MoveSpriteToPoint( "UFO", 50, 5 )
...
MoveSpriteToPoint( "UFO", 50, 150 )
```

Each time the sprite moves, it moves 1 pixel down the screen – just like we want it to!

That’s your first variable, and your first loop, and performing instruction inside a loop – these are all **very important** programming concepts! If they are not clear enough yet, please go back to the beginning of this section and reread it again.

Let's add one final detail to finish our program:



The only new instruction we have added is **Delay (10)**, inside the **For** loop. Why do we have to do that? **Because computers count very very very fast!** For a computer to count from 1 to 150 takes less time that it takes you or me to blink our eyes. Really! Computers are **that** fast! If we want to actually watch our UFO move down the screen, we have to slow the computer down a little while it counts. **Delay (10)** simply tells KPL to take a very short break before it moves the UFO each time.

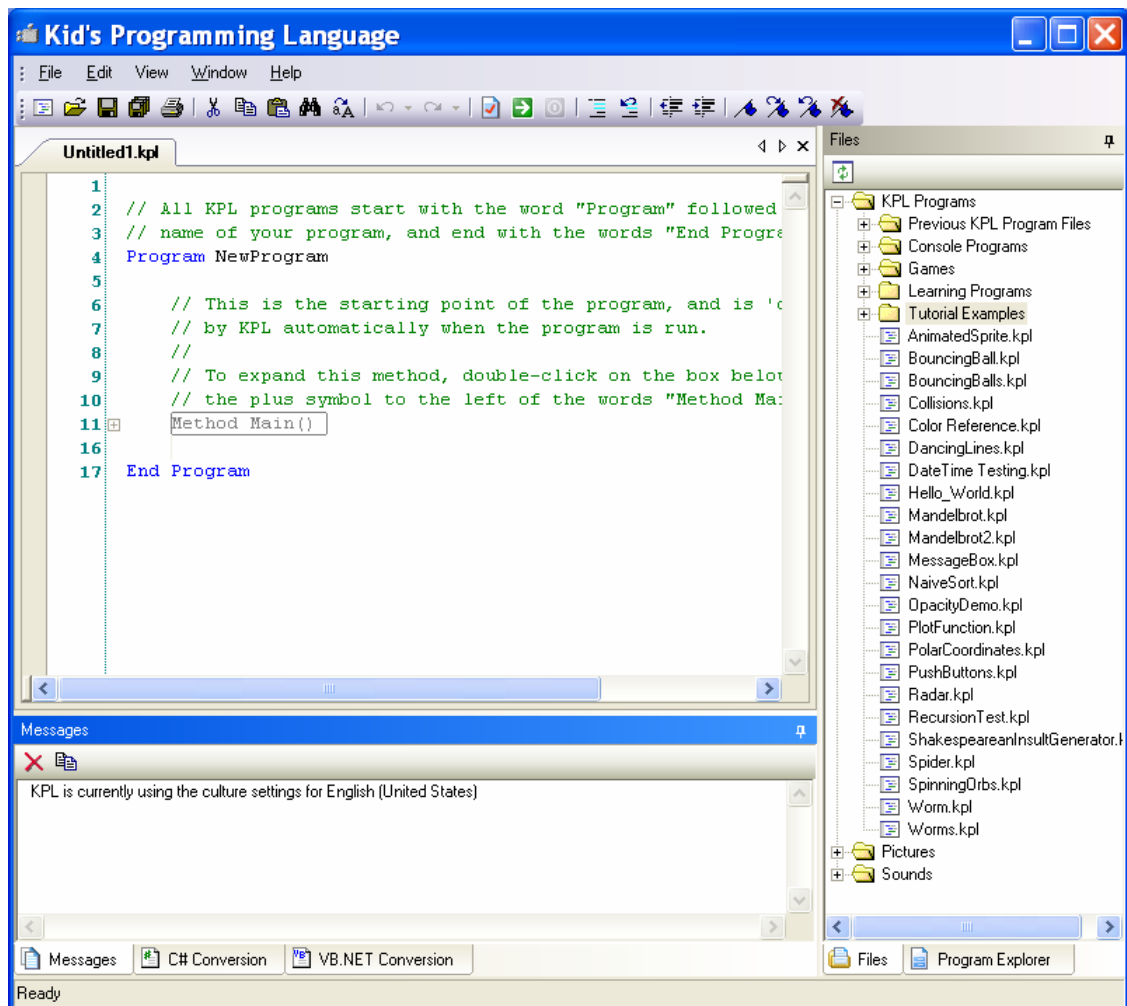
You know how when we want to count seconds like a clock, we instead say "one thousand one, one thousand two, one thousand three" to slow our counting down a little? This is exactly like that. With **Delay (10)** we are slowing down KPL's counting. When you work on this program yourself in KPL, try changing the **10** to other values and then run the program. Try **Delay (2)** and then try **Delay (100)**. Makes a big difference in how the UFO moves, doesn't it?

We spent several pages explaining this program in detail, which makes it seem more complicated than it is. But once you get the hang of programming in KPL, these 8 instructions won't take long to type – and hopefully you agree that just a few KPL instructions make it easy to do cool graphics with KPL!

Using KPL on Your Computer

Very Important Point: This tutorial is written based on improvements to KPL which are in KPL versions released on or after October 10th. If you downloaded KPL before October 10th, or if the examples don't work as described when you type them in, please download and install the latest version of KPL from <http://www.kidsprogramminglanguage.com/download.htm>.

This section assumes KPL is already installed, and focuses on getting you started with using KPL. Look for and launch KPL from the Kids Programming Language item on your Start menu in Windows. When you launch KPL, it will look something like this:



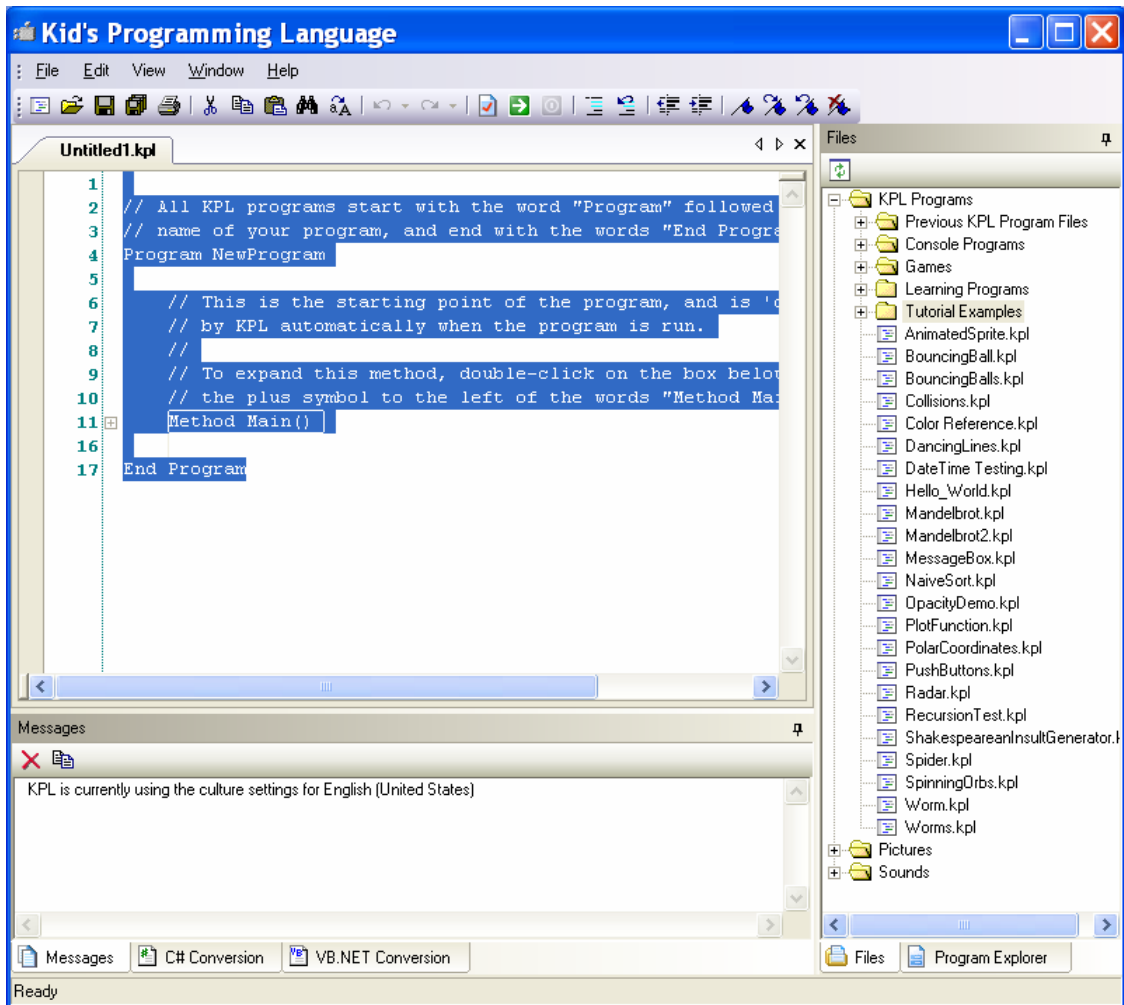
KPL is showing a new KPL program file called `Untitled1.kpl` in the Code Editor pane. The green lines you see in the code editor are “comments” which present information to the reader of the program, but which do not actually contain KPL instructions. Comments are prefixed by two forward-leaning slash marks: `//`

You will see many comments in the KPL example programs, and over time will learn to use comments yourself, as you write your own KPL programs. Comments help others to understand

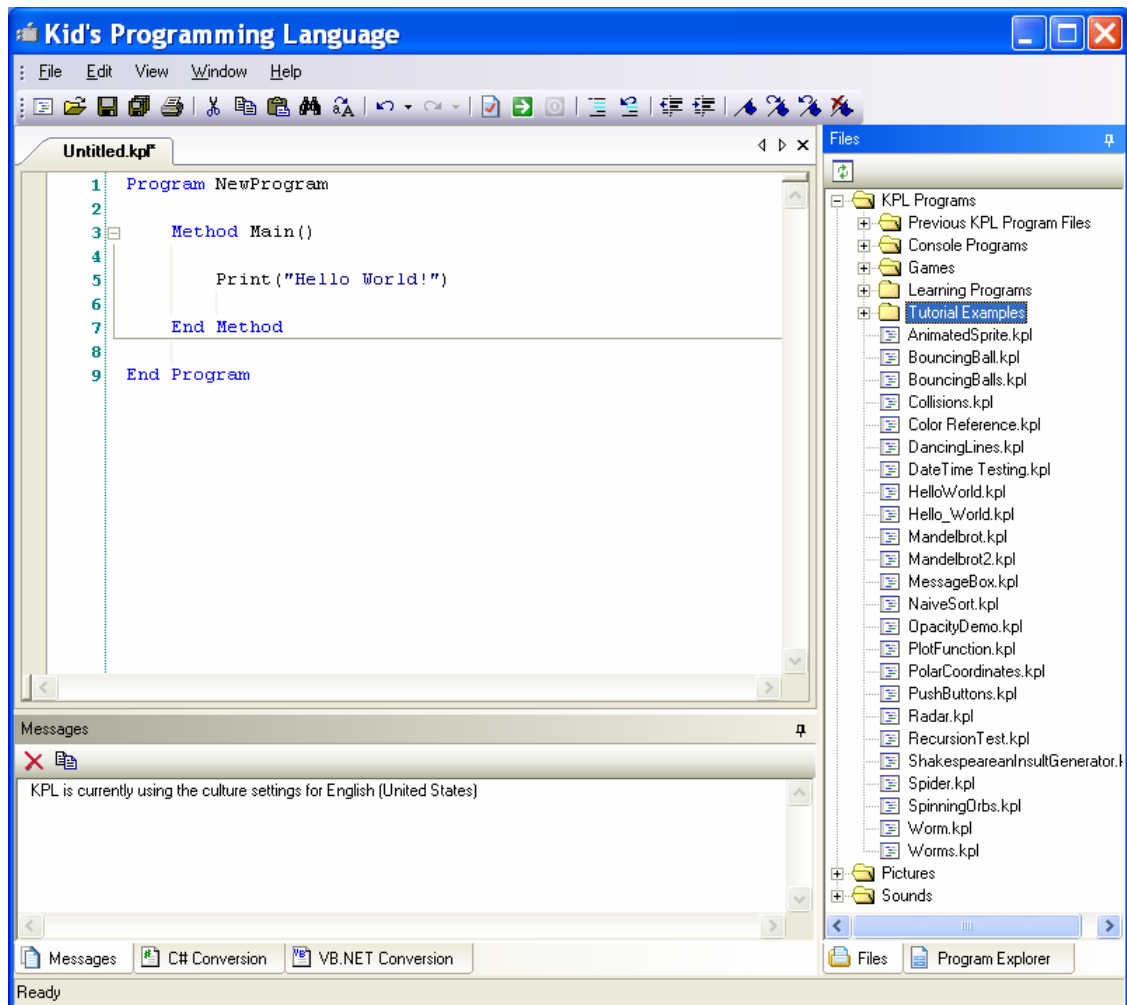
what the KPL program is doing – and can even help you remember when you look back at a program which you wrote long ago!

KPL's code editor has many of the same features as your word processor or email program. Take a moment to examine the menu items and the toolbar. If you hover the mouse pointer over the toolbar icons, you will see tooltips identifying them.


We want to focus on recreating the three example programs which we created in this tutorial, so let's start by clearing out the code which is shown in the program file Untitled1.kpl. To do this, click the **Edit** Menu, and the **Select All** item on that menu. You will see that all code in the code editor is now highlighted as shown below. With all of the code highlighted this way, you can either press the **"Delete"** key on your keyboard, or select the **Edit** menu's **Cut** command, and all the existing code will be cleared from the KPL code editor.

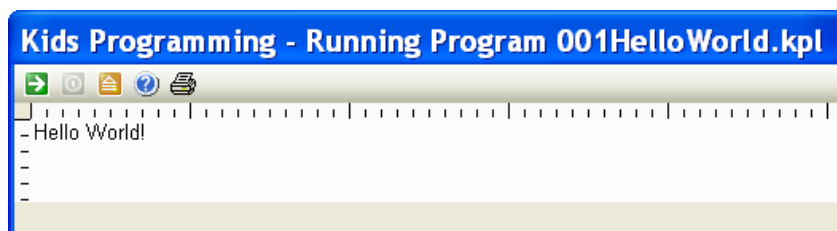


Once the previous code is deleted, go ahead and type in our first KPL program example, as shown here:



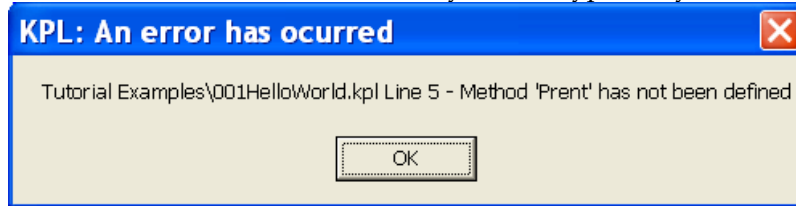
Remember that computers are very precise – so please try to type the KPL program exactly as shown above. It may or may not work properly if you type it differently. You can use the **TAB** key to indent your KPL code as shown. And you can hit **ENTER** on a line to leave it blank as shown. Using indentation and blank lines is not required, but will help you and others read and understand your KPL program more easily.

After you have typed in the program, click on the green arrow icon  or press the **F5** key to run your KPL program. If your KPL code was entered correctly, you should see a window pop up that looks like this, except it's a bit larger:



That's it! You are now a computer programmer! Way to go! Woohoo! Okay, okay, it's a very small and simple program, but still – you typed that in and made it run!

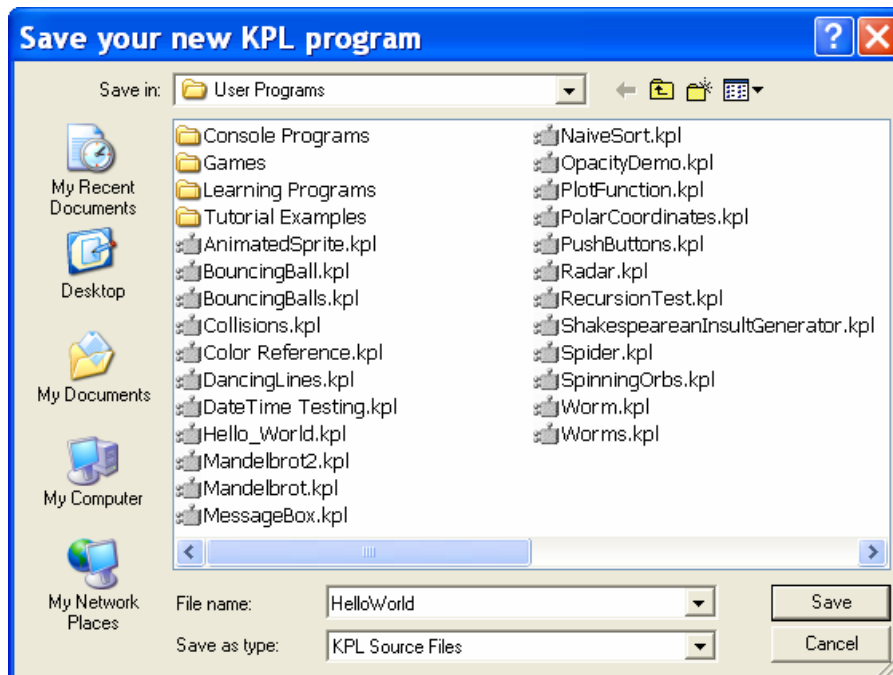
If KPL does not understand the code you have typed in, you will see an Error dialog like this one:



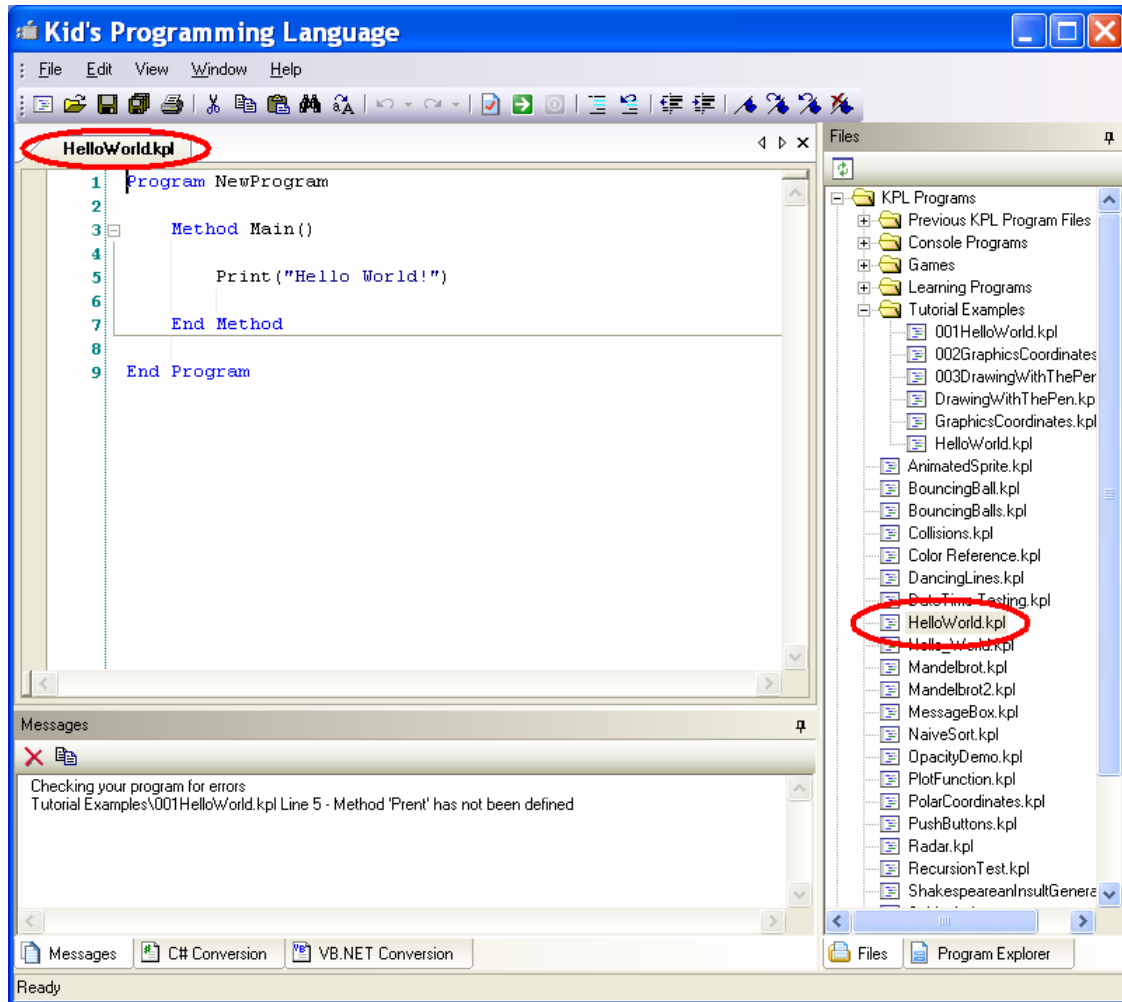
If that happens, please click OK, and carefully compare the code you typed against the code shown in the screenshot above. When you find the difference in your code and change it to match the example, you will be able to run the program.

All computer programmers sometimes have errors in their code, so don't feel bad about it when you do. None of us are perfect! When that happens with your code, just try to stay calm and patient and examine your code carefully – calm and patient and careful is the best way for you to find what's wrong and fix it.

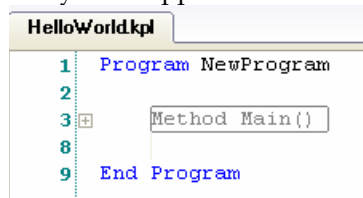
After you have the HelloWorld example working, click the **File** menu, **Save** option, and give your program the file name HelloWorld as shown:



After you save your KPL program you will see two places that prove that you did:

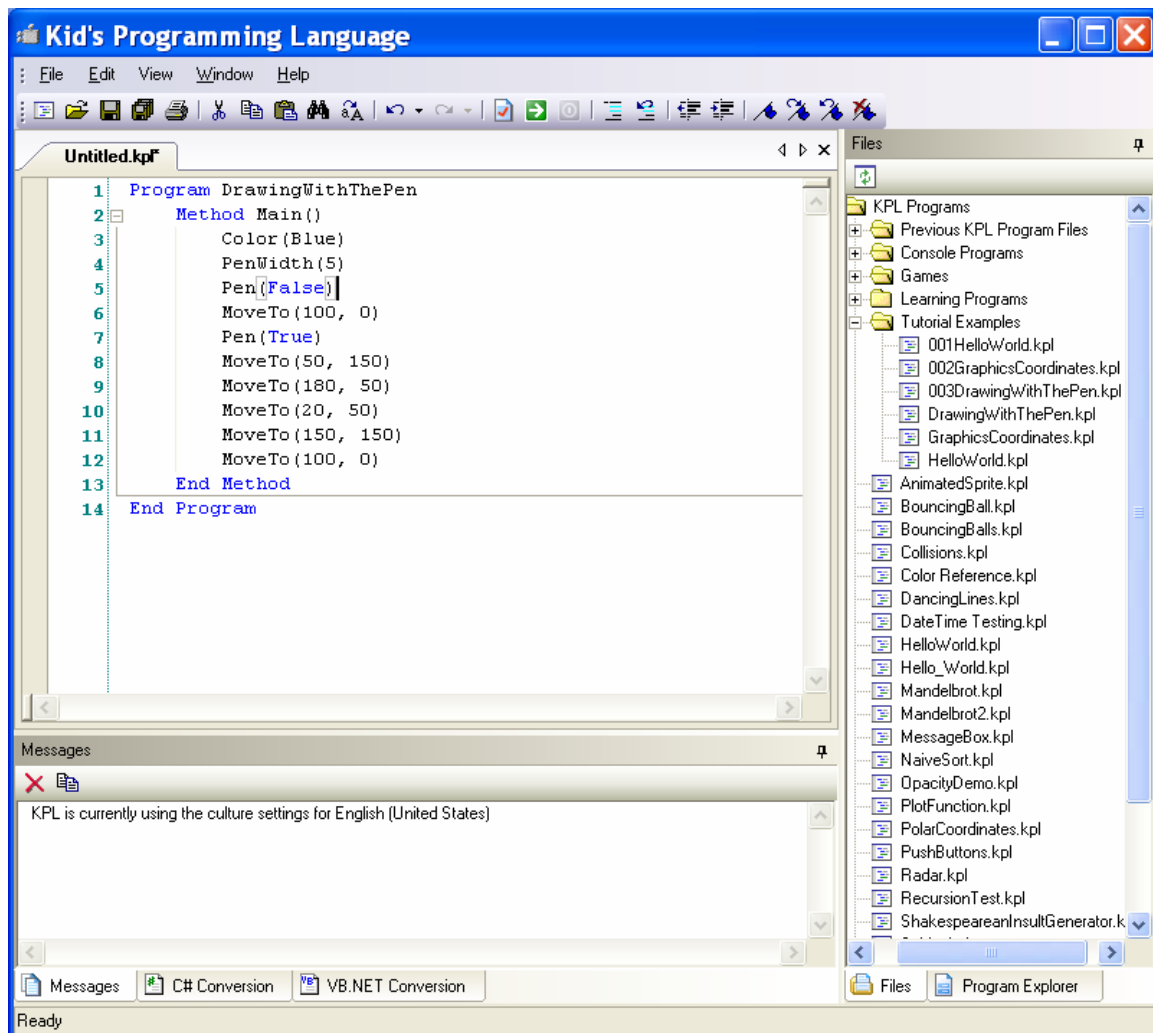


Now that your program is saved to disk, you can re-open it any time by double-clicking on it in the **Files** explorer on the right side of KPL. When you re-open your KPL program later, you will likely see it appear as shown below – don't panic, all your code is still there!



Click on the little + icon shown on line 3, and all the code within Method Main will “expand” so you can see it. Click the - icon, and the code will collapse again. This is not a useful feature for such a simple program, but when you get to writing much larger programs, you will see that hiding and showing code this way can be very convenient for you as a programmer.

After saving **HelloWorld**, click the **File** menu, **New Document Window**, and you will see a new **Untitled.kpl** program displayed, like the one you started with. Delete all of the code from it as you did before, and then enter the code for our second example:



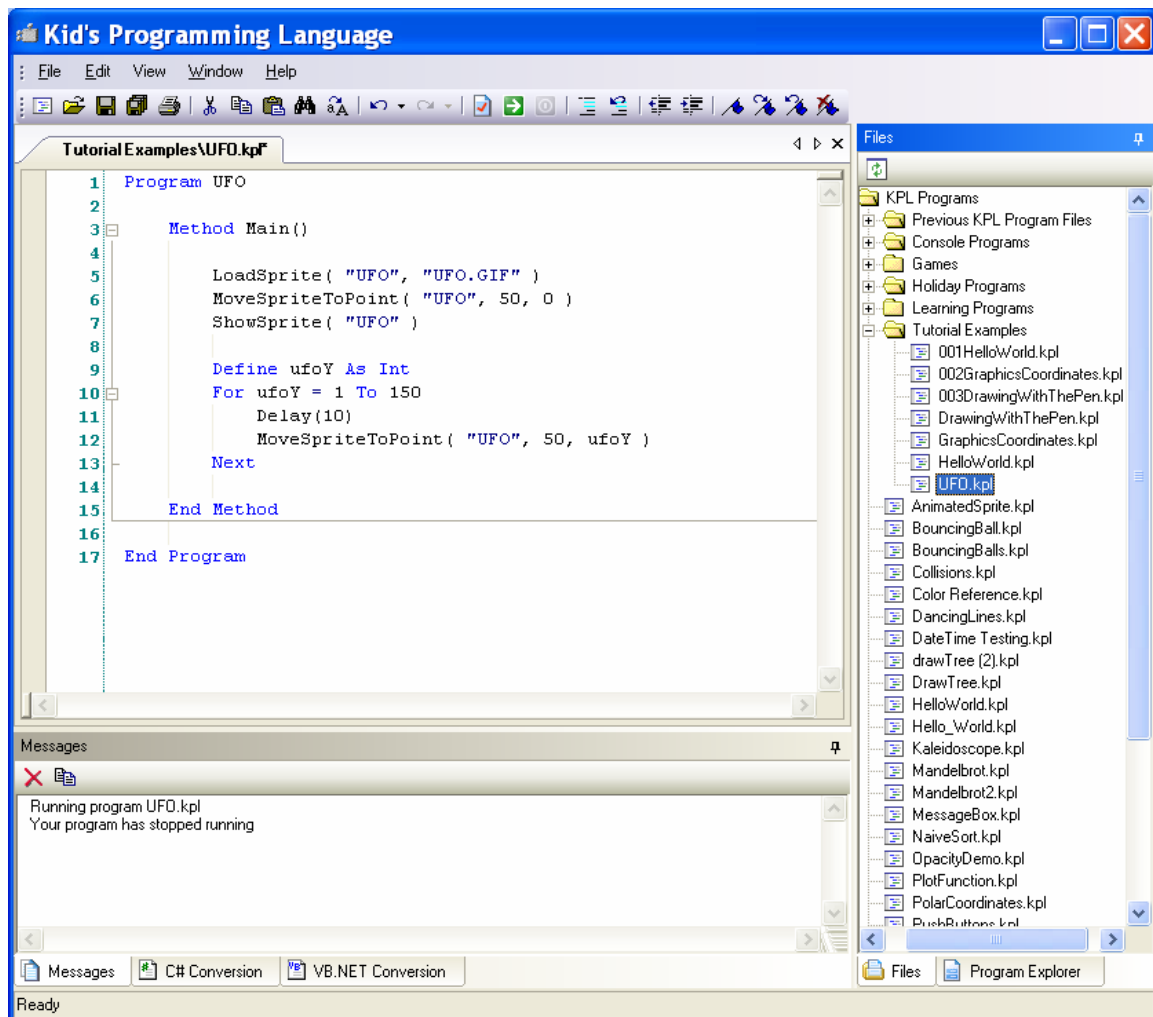
If you enter in all the KPL code exactly as shown here and run it, you will see the blue star, and you will be a computer graphics programmer! **Way to go!**

Remember if you encounter an error, just try to stay calm and patient and carefully look for the difference between your program and this example. When you change your code to match this exactly, it will run fine.

Don't forget to save this program after you have it working!

This is also a fun program for you to experiment with. What other colors look good? How does it look if you tell the pen to draw thicker or thinner lines? How would you draw a square or a triangle instead of a star? Here's a very difficult one: how would you make the star larger or smaller?

After saving **DrawingWithThePen**, click the **File** menu, **New Document Window**, and you will see a new **Untitled.kpl** program displayed. Delete all of the code from it as you did before, and then enter the code for our UFO example:



If you enter in all the KPL code exactly as shown here and run it, you will see the UFO fly. **Way to go!**

Remember if you encounter an error, just try to stay calm and patient and carefully look for the difference between your program and this example. When you change your code to match this exactly, it will run fine.

Don't forget to save this program after you have it working!

Try these fun exercises if you want some more good programming practice:

- Can you make the UFO move farther?
- How about making it fly from left to right instead of from the top down?
- Can you make the UFO fly down **and** to the right **at the same time**?

Next Steps After this Tutorial

The first thing to do after you get to this point in the tutorial is to look in KPL for the **Learning Programs** folder. There are 6 KPL programs there which you can learn from next – it's best to study them in order. In fact, the first 2 of the 6 will be very similar to programs you have already done in this tutorial! It's possible that new Learning Programs will be available by the time you read this – those would either already be part of your KPL installation, or you can find them on the Downloads page at www.kidsprogramminglanguage.com.

You might also want to download the **KPL User Guide for Teachers**, which is available from the same download page. It is not intended to be a tutorial for beginners, but it might be a useful reference for you as you move beyond this tutorial and work on your own KPL programs.

After you are comfortable with all the Learning Programs, there are many other example programs included in KPL, and you can open and examine and learn from them. **Kplong.kpl** and **NumberGuess.kpl** are particularly good programs to look at next. You will find those in the **Games** folder. They are much larger programs – and fully functional games! – but when you understand the tutorial and all the Learning Programs, you will probably have no problem learning those programs next.

Below are some hyperlinks to the online communities for KPL – these are also available from the Community page on the www.kidsprogramminglanguage.com site:

[Cool KPL programs](#)

[Kids' Discussion of KPL](#)

[Parents' discussion of KPL](#)

[Teachers' discussion of KPL](#)

[KPL Questions and Answers](#)

[International Language Versions of KPL](#)

If you have questions or would like more assistance with KPL, the KPL Questions and Answers forum is the first place to look. The online community is already fairly active, and is becoming more active all the time!