

# **Börja programmera med KPL**

**Av Jon Schwartz**

**Översättning till svenska av Anders Lundberg**

**Senaste uppdatering 16 november, 2005**

## Börja programmera med KPL

### Innehållsförteckning

<b>Varför ska jag lära mig programmering med KPL? .....</b>	<b>3</b>
<b>Hur ska jag använda den här handledningen?.....</b>	<b>3</b>
<b>OK, visa mig ett program!.....</b>	<b>4</b>
<b>Men grafik då?!.....</b>	<b>6</b>
<b>Spelgrafik med bildobjekt.....</b>	<b>11</b>
<b>Använda variabler och slingor i KPL.....</b>	<b>14</b>
<b>Använda KPL på din dator.....</b>	<b>17</b>
<b>Nästa steg efter denna handbok.....</b>	<b>24</b>

*Översättarens anmärkningar:*

*Bildobjekt – eller Sprites – jag har inte hittat något bättre ord än just bildobjekt för dessa tvådimensionella grafiska objekt.*

*Efter några instruktioner har jag valt att inom hakparentes [] skriva en försvenskat namn på instruktionen för bättre förståelse. Observera att det inte går att använda dessa svenska instruktioner i programmet.*

*Metoder – subrutiner, procedurer eller vad man vill kalla det, eftersom skaparna av programmet har valt att kalla det för Method så kallar även jag det metod vid några tillfällen.*

## Introduktion: Vad är programmering?

Datorprogrammering är kort och gott: **att ge datorn instruktioner.**

Datorer är mycket bra på att följa instruktioner. Dom gör precis det vi säger till dom att göra. Men dom har **ingen fantasi!** När vi skriver ett program måste vi därför tala om för datorn exakt vad den ska göra och när den ska göra det.

## Varför ska jag lära mig programmering med KPL?

Olika datorprogramspråk ger dig, programmeraren olika sätt att berätta för datorn vad du vill att den ska göra. "Kid's Programming Language", eller KPL, har flera viktiga fördelar för dig som är nybörjare:

- KPL är utvecklat för att göra det **så enkelt som möjligt** för nybörjaren att lära sig programmering.
- KPL är utvecklat för att göra det **så roligt som möjligt** för dig att lära dig programmera
- KPL, är, olikt andra nybörjarspråk, utvecklat för att vara **så likt "riktiga" programmeringsspråk som möjligt**

Ett talesätt som passar KPL är: **"Du måste lära dig gå, innan du kan springa."** Att lära sig programmera med KPL är som att lära sig gå. När du lärt dig och behärskar KPL, kommer det vara mycket enklare för dig att lära dig springa, oavsett om du väljer att springa med Java, Python, Visual Basic eller C#.

## Hur ska jag använda den här handledningen?

**Observera:** Den här handledningen baseras på de förändringar och förbättringar som införts i KPL från och med versionen som kom ut den 10 oktober 2005. Om du laddade ned KPL före den 10 oktober 2005, eller om programexemplen i denna handledning inte fungerar som dom beskrivs, ladda ned den senaste versionen från hemsidan: <http://www.kidsprogramminglanguage.com/download.htm>.

Om du är nybörjare, och aldrig tidigare programmerat, så använder du denna handledning bäst genom att läsa kapitel för kapitel i tur och ordning. Var säker på att du förstår det du läst innan du börjar läsa nästa kapitel. Förmodligen är det allra bäst att läsa handledningen utan att du använder datorn i början, åtminstone till du kommer till kapitlet "Att använda KPL på din dator". Att studera och lära dig KPL på detta sätt hjälper dig att undvika att bli distraherad av detaljer som kommer behandlas i senare delar av handledningen.

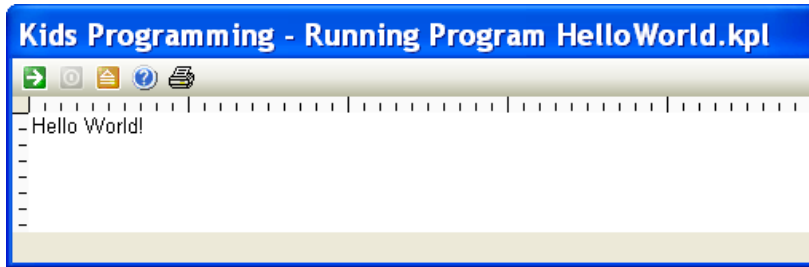
Att lära sig programmera innebär att du lär dig tänka på ett sätt som du kanske inte gör normalt. Datorer kräver att vi är betydligt mer logiska, precisa och strukturerade än vi behöver vara i normala fall! Det kan vara svårt i början, men du klarar det! När du börjar få grepp om det, så kommer du märka att det inte är så svårt.

Ett av de bästa sätten att lära sig det här nya sättet att tänka är att fråga någon som redan kan programmera. Kan du komma på någon som kan hjälpa dig med dina funderingar medan du läser den här handledningen?

## OK, visa mig ett program!

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

Så, det var det! När du startar det här KPL-programmet kommer resultatet att bli följande bild:



De fem raderna KPL-kod överst är en skärmbild från KPL, exakt så ser koden när du programmerar i KPL.

En sak som jag bör påpeka tidigt är att siffrorna till vänster om varje rad egentligen inte används av KPL – dom finns bara där för att det ska vara lättare för dig att hitta rätt när du läser och arbetar med KPL-koden. Äldre programspråk som GWBASIC använde radnummer för att bearbeta instruktionerna i den ordning programmeraren avsåg. KPL behöver inte det. KPL-program processas vanligen en rad i taget, genom att börja överst i programmet och successivt arbeta sig nedåt i raderna. Det finns undantag, men dessa behandlar vi inte i denna nybörjarhandledning.

En viktig regel när du programmerar i KPL är att varje rad med instruktioner måste vara en separat rad. Se i exemplet nedan, det är precis samma instruktioner som i programmet ovan, men eftersom allt är på en rad skulle det inte fungera:

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

Alla programspråk har vissa regler som programmeraren måste följa, så att datorn kan förstå instruktionerna i programmet. När två personer pratar med varandra finns det också regler som har samma syfte, det är bara det att vi är så vana vid dom att vi inte tänker på det. Till exempel säger vi inte "Hej då" när vi svarar i telefonen, och vi säger inte "Hallå" när vi lägger på! OK, det kanske var ett löjligt exempel men det finns en poäng, KPL behöver på samma sätt en start och ett slut på programmet. Alla KPL-program måste börja med en rad som t.ex. **Program HelloWorld** som visas på rad 1, nedan. För att KPL-program ska veta när det är slut måste det finnas en rad där det står **End Program** som på rad 5 nedan:

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

Du kan döpa ditt program till vad du vill, men det är bra om du döper det till något som beskriver vad programmet gör. I detta exempel valde jag namnet "**HelloWorld**". Du kan döpa ditt första program till något annat, t.ex. **Program MittFörstaProgram**.

**Method Main()** är nästa viktiga sak du måste känna till om KPL-programmering. Alla KPL-program börjar läsa den första instruktionen i metoden **Method Main()**. Om du tittar på exemplet nedan ser du att den instruktionen är **Print("Hello World!")**.

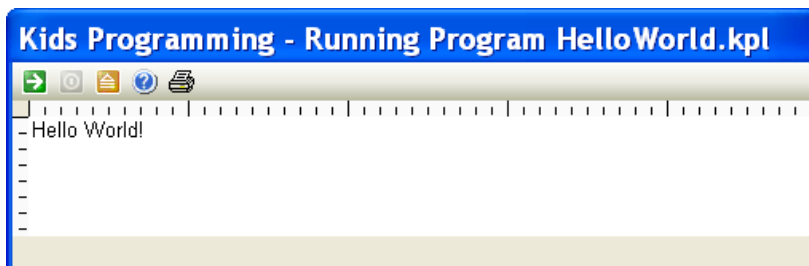
**Method Main()** låter kanske som ett lite stelt sätt att bestämma var programmet ska börja, men det är faktiskt så de flesta moderna programspråk fungerar. Som du sedan kanske märker finns det en avslutning på metoden också, **End Method** matchar och definierar avslutningen på **Method Main()**.

Vi går inte närmare in på metoder i den här handledningen. Men när du nu tittar i exemplet nedan tycker du förhoppningsvis att det verkar riktigt om vi säger att det bara finns en instruktion i metoden **Method Main()** och det är raden **Print("Hello World!")**.

Allt detta var ett invecklat sätt att beskriva att detta speciella program egentligen bara innehåller en instruktion, instruktionen som säger till datorn att skriva "Hello World", **Print("Hello World!")**. En summering: Rad 1 och 5 säger till datorn att det finns en början och ett slut på detta KPL-program. Rad 2 och 4 säger till datorn att det finns en början och ett slut på **Method Main()**.

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

Nu tittar vi en gång till på fönstret som dyker upp när du kör programmet. Kolla noga på det som står i fönstret, datorn har bara utfört en sak, precis det vi sa till den att göra. Datorn har skrivit texten "Hello World"!



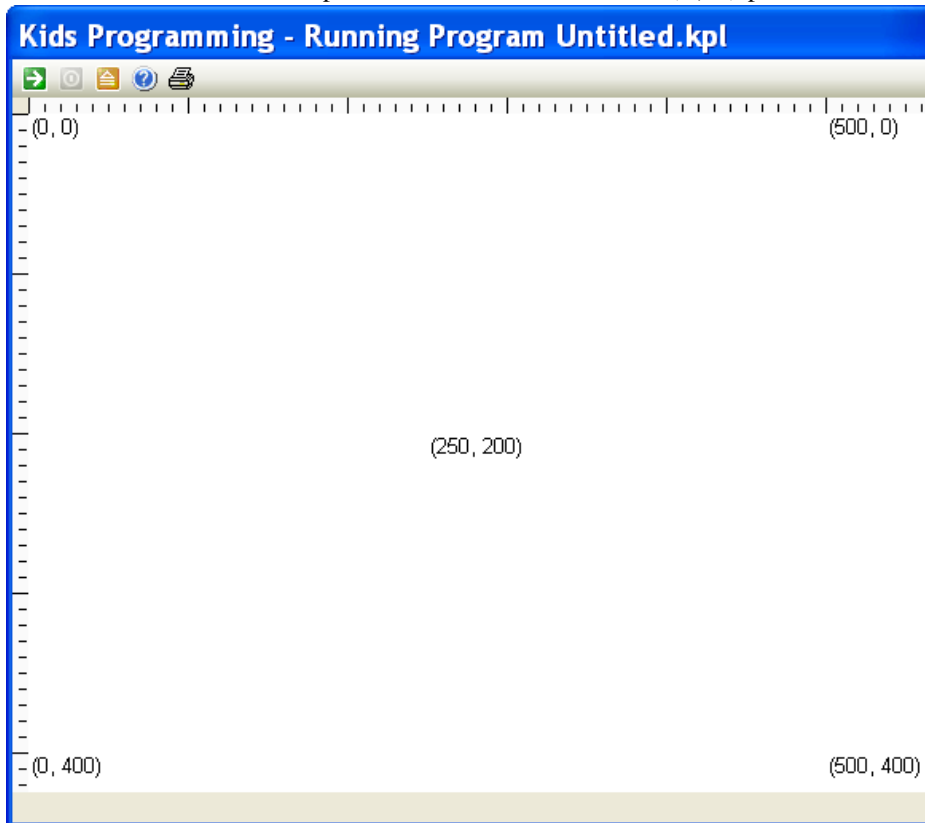
## Men grafik då?!

**Hello World!** Är ett klassiskt första program att skriva – fast det är inte speciellt spännande! Så låt oss därför gå vidare till datorgrafik.

En förklaring av och exempel på datorgrafik måste börja med en förklaring av hur datorn placerar grafik på skärmen. Datorer använder ett **koordinatsystem** som är annorlunda än det koordinatsystem du kanske lärt dig i matten i skolan. Men datorns sätt att använda koordinater är lätt att förstå och arbeta med, dessutom är det lätt att arbeta med var på skärmen olika saker befinner sig.

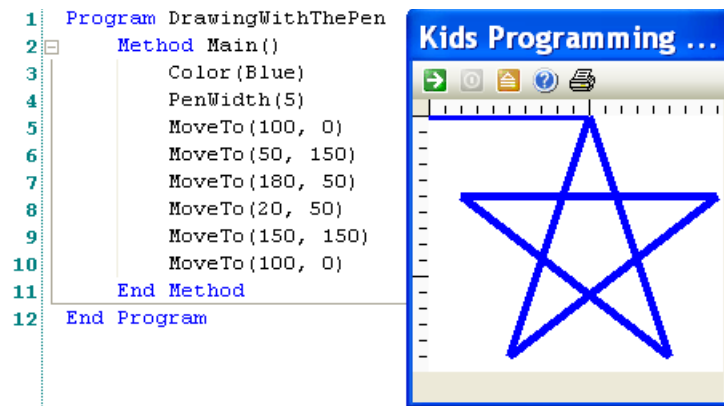
Datorer använder ett **(X, Y)**-koordinatsystem för olika punkter på skärmen. På den vänstra kanten på skärmen är **X=0**, på den översta delen av skärmen är **Y=0**. Detta innebär att origo, där  $X=0$  och  $Y=0$ , är punkten som ligger längst till vänster allra högst upp, alltså det övre vänstra hörnet. Förflyttar du dig till höger över skärmen ökar X-värdet, och rör du dig nedåt på skärmen ökar Y-värdet.

Om du inte är van vid att tänka i koordinatsystem, så är det en komplicerad förklaring, men det kanske blir enklare att förstå om du tittar på bilden. På bilden ser du flera **(X, Y)**-punkter utskrivna i ett KPL-program:



Ta dig lite tid på att fundera på och försöka förstå siffrorna. Det första värdet i varje par är **X-värdet**, som du kan se blir det större ju längre till höger på skärmen det står. Det andra värdet är **Y-värdet**, Y blir större ju längre ner på skärmen det står.

Låt oss skriva vårt första KPL-grafikprogram, under tiden lär du dig hur du använder koordinater i KPL för att skapa fräck grafik på skärmen. Vi börjar med att visa hela programmet och resultatet som blir när du kör programmet. Sedan går vi igenom del för del i detalj för att förklara exakt hur det fungerar:



Den här gången är det 8 KPL-instruktioner i **Method Main()**, istället för som tidigare bara en instruktion. Det är betydligt mycket mer än i det första exemplet, fast förhoppningsvis tycker du ändå att det är ganska coolt att KPL kan rita en blå stjärna på skärmen med bara 8 instruktioner!

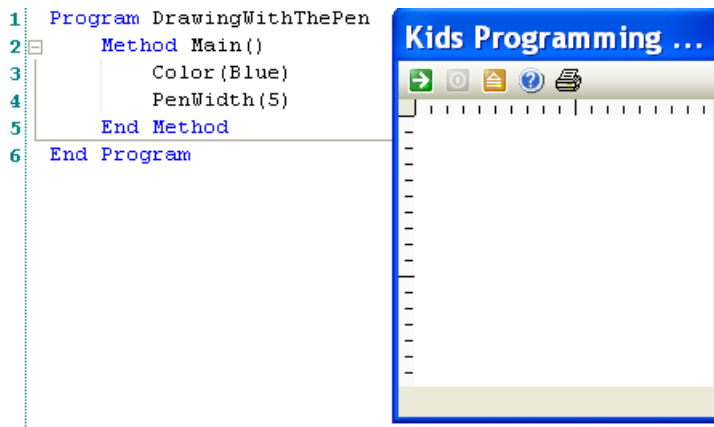
Som vanligt så börjar och slutar programmet på samma sätt som alla KPL-program måste göra. Den enda skillnaden från första exemplet är att detta program heter **DrawingWithThePen**:

```

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

```

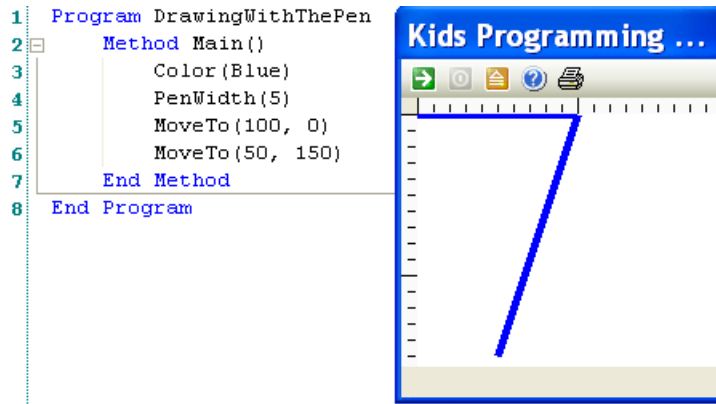
Nu ska vi lägga till några instruktioner i **Method Main()**, och se vad dom gör:



Dom två första instruktionerna har lagts till, men som du kan se syns det inget i fönstret ännu.

**Color(Blue)** [Färg(Blå)] säger till KPL att vi vill att datorn ska använda färgen Blue när den ritar på skärmen. **PenWidth(5)** [Pennbredd(5)] säger till KPL att när datorn ritar med pennan ska bredden på linjen vara 5 punkter. Du kan givetvis använda andra värden på bredden. **PenWidth(2)** ritar en smalare linje, **PenWidth(10)** drar en tjockare linje.

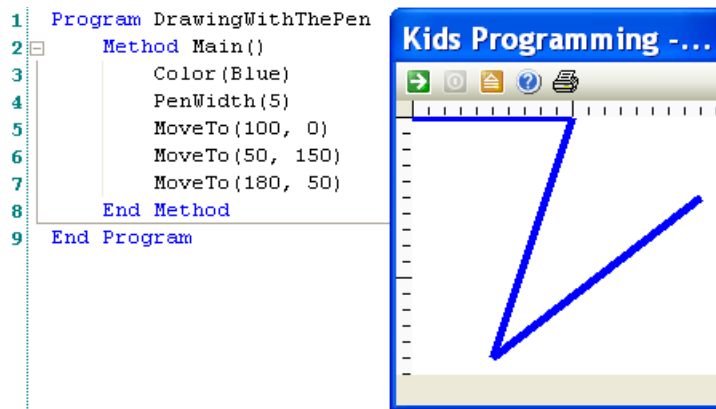
Så här långt har vi berättat för KPL hur vi vill att datorn ska rita, men vi har inte berättat för programmet var datorn ska rita. Så låt oss rita det första strecket i stjärnan:



Den första instruktionen vi lagt till är **MoveTo (100, 0)** [FlyttaTill(100, 0)]. Detta säger till KPL att flytta pennan till punkt **(100, 0)** på skärmen, vilket är början på vår stjärna. Eftersom KPL alltid börjar med pennan på punkt **(0, 0)** – det övre vänstra hörnet – innebär denna förflyttning till punkt (100, 0) att en linje ritas på vägen, det är den översta vågräta linjen.

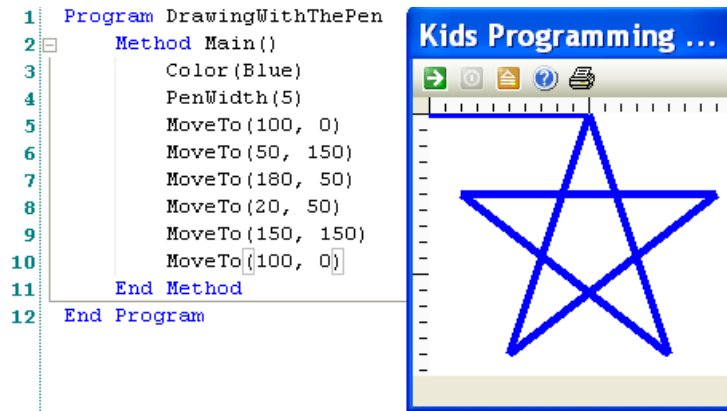
Den andra instruktionen vi lagt till är **MoveTo (50, 150)**. Denna instruktion ritas den första linjen i vår stjärna. Fundera på **(X, Y)**-värdena för dessa två punkter, ett ögonblick. Eftersom denna instruktion säger till KPL att flytta pennan från X=100 (där den var innan) till X=50, kommer linjen att gå mot vänster. Och eftersom vi flyttar från Y=0 till Y=150 så kommer den samtidigt att gå nedåt.

Nu kör vi på och ger ytterligare en instruktion till KPL, vilket ritas nästa linje på stjärnan:



Instruktionen vi la till nu är **MoveTo (180, 50)**. KPL fortsätter att flytta pennan, från den senaste punkten **(50, 150)** till den nya punkten **(180, 50)**. Har du någonsin sett en Etch-A-Sketch-ritbräda, dom där röda leksakstavlorna som du kan rita med genom att vrida på två rattar? KPL:s penna är ungefär som en datoriserad Etch-A-Sketch. Låt oss lägga till dom sista raderna med instruktioner för att avsluta stjärnan:



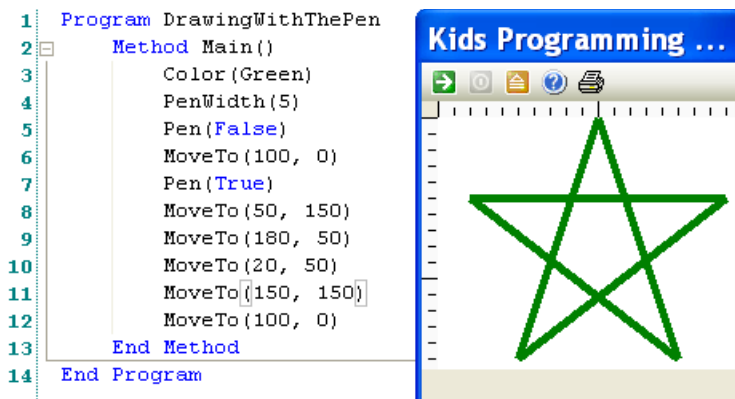


Som du kan se så gör tre instruktioner till att KPL ritar ut dom sista linjerna i stjärnan. Totalt har vi flyttat pennan 6 gånger och ritat 6 linjer. Resultatet har blivit en stjärna, ritade enligt de instruktioner vi gett vårt KPL-program.

KPL-programmet som behövs för att utföra detta är litet, bara 8 KPL-instruktioner. Det svåraste i detta exempel har varit att vänja sig vid det sätt som koordinatsystemet (X, Y) fungerar i KPL. Om du vid det här laget fortfarande är osäker på vad (X, Y)-värden är, bör du kanske gå tillbaka till början av detta kapitel och läsa igenom det en gång till. Om du då fortfarande är osäker på (X, Y)-värden kanske du ska skriva ditt eget KPL-program enligt exemplet ovan, och experimentera lite med siffrorna och värdena för att dra egna linjer och se vad som händer. Kanske kan du öva dig på att rita en triangel eller fyrkant? Om du nu bestämmer dig för att göra det, läs då först kapitlet "KPL på din dator", så att du vet hur det fungerar innan du går vidare. När du sedan övat och känner dig säkrare, kom då tillbaka till den här punkten i handledningen och fortsätt att lära dig om grafik.

Det är viktigt att ägna tid åt att förstå hur KPL hanterar (X, Y)-koordinater, eftersom dessa är grunden i alla grafikprogrammering. Andra programspråk hanterar grafik på samma sätt, så när du lär dig detta med KPL lär du dig samtidigt den mest grundläggande tekniken för att programmera grafik i vilket annat språk som helst.

Låt oss slutligen lägga till en sista detalj i vårt exempel, vilket också visar en liten extra finess som du kan göra med KPL:s penna. Det är en grej du inte kan göra med en Etch-A-Sketch! En annan sak du inte heller kan göra med en Etch-A-Sketch är att byta färg på det du ritar, så vi gör det också, bara för att vi kan med KPL:

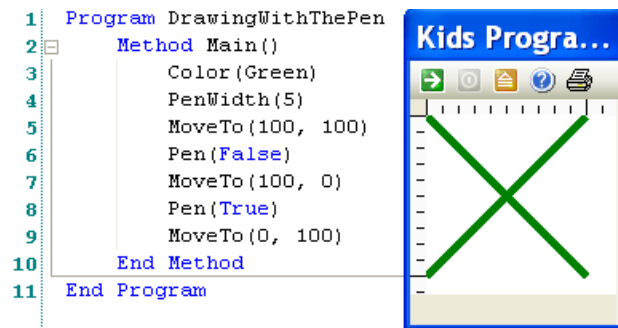


Som du såg, så bytte vi just färg från **Color (Blue)** till **Color (Green)**. Enkelt! Dessutom har vi gett KPL en ny instruktion **Pen (False)** [Penna(Falskt)], precis innan vi flyttar den med **MoveTo (100,**

0). **Pen (False)** betyder helt enkelt att KPL inte ska låta datorn rita med pennan medan den flyttar. När pennan flyttar sig från (0, 0) till (100, 0) denna gång, ritas inte längre den vågräta linjen från hörnet till början av stjärnan, och visst ser det bättre ut!

Kom ihåg att datorer saknar fantasi och bara gör som du instruerar den. Eftersom vi sa att pennan inte skulle rita **Pen (False)**, slutar KPL att rita när pennan flyttas. Så vad behöver vi då göra för att få pennan att börja rita igen? Jo, vi måste ge KPL instruktionen: **Pen (True)** [Penna(Sant)], så att KPL börjar rita igen när pennan flyttas.

En summering: **Pen (False)** säger till KPL att stänga av pennan när den flyttas, och **Pen (True)** säger till KPL att sätta på pennan när den flyttas. En Etch-A-Sketch kan inte göra det, in den är pennan alltid på. Att slå på och av pennan under tiden du flyttar den låter dig rita allt möjligt, både komplicerade figurer och även flera olika figurer i samma teckning. Här är ett väldigt enkelt exempel på en figur som är enklast att rita genom att först rita ett streck, sedan slå av pennan, flytta den, slå på den och rita nästa streck:

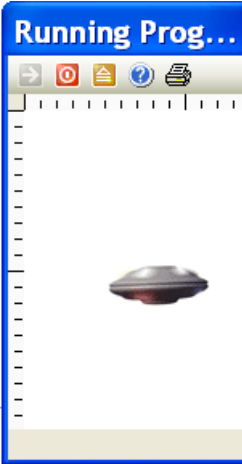


## Spelgrafik med bildobjekt

Vitsen med exemplet ovan är framför allt att förklara hur (X, Y)-koordinater fungerar i KPL. Men vi kan ju knappast kalla det för spelgrafik ännu? Spelgrafik **ritas** normalt inte på skärmen. Spelgrafik laddas från färdiga bildfiler, som t.ex. ett flygande tefat, en asteroid eller en alv med pilbåge. För att göra detta på ett enkelt sätt använder KPL något som på engelska kallas **sprites**! Jag kommer att kalla det för bildobjekt i texten nedan.

Precis som vi gjorde med det förra exemplet så börjar vi med att visa det färdiga programmet i sin helhet, sedan förklarar vi detaljerat hur varje del fungerar så att du lär dig skriva egna program. Här nedan ser du hela KPL-programmet, som innehåller exakt åtta instruktioner i **Method Main()**, lika många som det förra exemplet. De åtta instruktionerna ger KPL i uppdrag att visa ett flygande tefat i överkanten av skärmen och sedan flytta det sakta nedåt i fönstret, tills det hamnar där du ser det i exemplet:

```
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
```



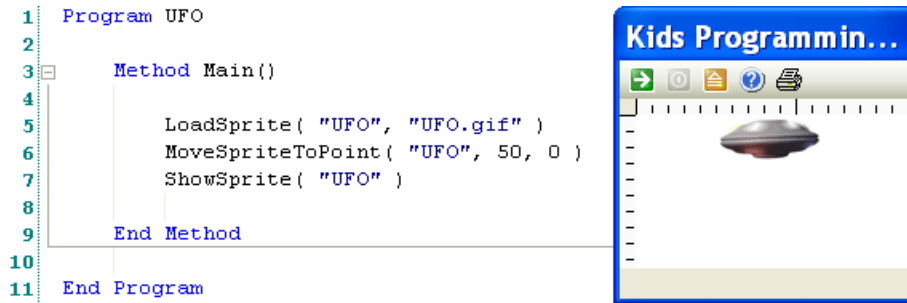
Tittar du noga på koden så ser du att det börjar och slutar på samma sätt som våra andra exempel med den enda skillnaden att detta program är döpt till **UFO**:

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

Här kan du se några andra exempel på bilder som finns med i KPL-installationen. Du ser också vad filnamnen heter. Du kan använda vilken bildfil som helst i KPL, inklusive sådana du gör själv. Bilderna nedan är bara några exempel på de 65 bilder som följer med i KPL:



OK, nu har du sett några exempel på vilken typ av bilder du kan använda i KPL, ska vi fortsätta med att visa vilka KPL-instruktioner du använder för att visa bilderna på skärmen:



Den första instruktionen vi lägger till är **LoadSprite( "UFO", "UFO.gif" )** [LaddaBildobjekt("UFO", "UFO.GIF")]. Som jag sagt tidigare, i KPL finns det lite regler för hur man skriver in instruktioner. Reglerna för **LoadSprite** är inte svåra, men dom måste vara exakta! **LoadSprite** kräver två "värden" för att fungera. Det första värdet är det namn du vill ha på bildobjektet – i vårt fall, **"UFO"**. Det andra värdet är namnet på bildfilen som KPL ska använda för det här bildobjektet, i vårt fall **"UFO.gif"**. Bägge värdena måste vara omgivna av citationstecken (") precis som exemplet visar. Dessutom måste det vara ett komma mellan värdena. Namnet på bildobjektet (det första värdet) behöver inte vara samma som den första delen av filnamnet, men det kan vara praktiskt, du använder namnet när du gör saker med bilden, t.ex. flyttar den.

Här är några exempel på felaktiga sätt att instruera KPL att ladda ett bildobjekt. Jag upprepar: **alla fyra exemplen nedan är felaktiga**, eftersom dom inte följer KPL:s regler för LoadSprite. Du kan nog lista ut vad som är fel och hur det ska vara för att KPL ska kunna förstå instruktionerna?

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

**ummering:** LoadSprite( "UFO", "UFO.gif" ) instruerar KPL att skapa ett nytt bildobjekt med namnet "UFO", av bilden i filen "UFO.gif".

Nästa steg är att ge instruktioner till KPL om var på skärmen bildobjektet ska ritas. Instruktionen **MoveSpriteToPoint( "UFO", 50, 0 )** [FlyttaBildobjektTillPunkten("UFO", 50, 0)] gör det. **MoveSpriteToPoint** kräver tre värden, först namnet på bildobjektet, dvs. **"UFO"** eftersom det var namnet vi bestämde i **LoadSprite**. Nästa två värden är (X, Y)-värdena för en punkt på skärmen, som du säkert kommer ihåg, det första värdet (X) är **50** och det andra värdet (Y) sätter vi till **0**.

En viktig detalj som du bör lägga märke till är att det inte är några citationstecken runt siffrorna. I vanliga fall ska det alltid vara citationstecken runt värden som är ord (text), men inte när det är siffror som är numeriska (dvs. tal eller nummer på punkterna i koordinatsystemet i det här fallet).

**ummering:** MoveSpriteToPoint( "UFO", 50, 0 ) instruerar KPL att flytta bildobjektet "UFO" till koordinaten (50, 0) på skärmen.

Det enda som återstår är att visa bildobjektet. Det är den enklaste instruktionen hittills: **ShowSprite( "UFO" )** [VisaBildobjekt("UFO")]. Det var det! Tefatet visas nu på skärmen precis där vi sa till KPL att rita ut det.

Stön, vilken detaljerad och lång förklaring, bara för att rita en bild på skärmen. Nää, jag ska nog ta bort alla förklaringar och bara visa hur enkelt det är i KPL. Här nedan ser du instruktionerna i programmet jag har beskrivit och hur det ser ut när vi kör det:

```
1 Program UFO
2
3 Method Main()
4
5     LoadSprite( "UFO", "UFO.gif" )
6     MoveSpriteToPoint( "UFO", 50, 0 )
7     ShowSprite( "UFO" )
8
9 End Method
10
11 End Program
```



## Använda variabler och slingor i KPL

Nu ska vi få tefatet att landa, dvs. flytta det nedåt lugnt och fint. För att landningen ska bli lyckad presenterar jag härmed vår första ”variabel”, en viktig del i KPL och andra datorprogram.

Uttrycket variabel är en bra beskrivning på vad det är. Det är ett värde som kan ändra sig under tiden programmet kör. Att kunna ändra på ett värde under programkörningen är väldigt bra, vilket du kommer att se i exemplet. En variabel bör döpas med ett namn som är en bra beskrivning på vad den ska användas till. I vårt exempel kommer variabeln att användas till att ändra Y-värdet för att flytta tefatet nedåt på skärmen, därför kallar vi variabeln **ufoY**.

Här är den rad med kod som presenterar vår variabel:

```
Define ufoY As Int
```

**Define** är ett såkallat nyckelord i KPL som låter KPL veta att du definierar (bestämmer) en ny variabel som KPL kan använda. **ufoY** är namnet på variabeln. **As Int** berättar för KPL vad variabeln är för typ, i det här fallet är **ufoY** ett heltal. **Int** är en förkortning för Integer som betyder heltal. Heltal är siffror utan decimaler, t.ex. **-1** eller **0** eller **43**.

Vi vet att vårt tefat börjar sin flygning på positionen **(50, 0)** på skärmen, eftersom vi sa till KPL att flytta bildobjektet dit. Men hur gör vi för att flytta det nedåt? Genom att öka det andra värdet (**Y-värdet**) i instruktionen **MoveSpriteToPoint( "UFO", 50, 0 )** flyttar vi bildobjektet, först till **(50, 1)**, sen till **(50, 2)**, sen till **(50, 3)**, sen till **(50, 4)**, osv.... Det är upp till oss hur långt vi vill flytta det innan vi tycker att det har landat, men vi kan komma överens om att vi stannar vid punkten **(50, 150)**.

Ser du mönstret? Det första värdet (**X-värdet**) är alltid 50, men det andra värdet (Y-värdet) ökar med 1 hela tiden, från 0 och till 150 (som vi sa vi skulle stanna vid). Det är viktigt att du hänger med nu och att förstår och kan se mönstret innan vi fortsätter med att visa hur man gör detta med KPL-kod. Om du tycker att det verkar lite konstigt, läs en gång till innan du fortsätter.

I raderna nedan visar jag hur du med KPL-kod instruerar KPL att öka värdet från 1 till 150, ett steg i taget och sedan flytta bildobjektet till den nya punkten:

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

Detta är din första ”slinga”. Slingor är ytterligare en av dessa nya grejer du måste lära dig om du aldrig har programmerat tidigare, men när du förstår vad det handlar om så kommer du att tycka det här med slingor är ganska enkelt.

Den här slingan (**For...To...Next**, ungefär Från ... Till ... Nästa) startar med att variabeln är lika med 1, **ufoY = 1**. Vi säger också att vi ska räkna till 150, **To 150**. Slingan slutar alltså gå runt när variabeln är lika med 150, **ufoY = 150**. Slutet på slingan anges med nyckelordet **Next**. När KPL kommer till nyckelordet **Next**, vet KPL att det är dags att gå tillbaka till början av slingan och öka värdet på variabeln med 1, alltså öka värdet på **ufoY** till nästa värde, från 1 till 2, från 2 till 3, från 3 till 4 osv. hela vägen till den sista ökningen från 149 till 150.

**I grund och botten säger vi åt KPL att räkna från 1 till 150, och vi vill att KPL lagrar siffran i variabeln ufoY för att vi ska hålla ordning på var vi är.**

Det var själva början och slutet på slingan, men vi vill att något ska hända under tiden KPL räknar, eller hur? Vi vill att tefatet ska landa! Det är det instruktionen **MoveSpriteToPoint( "UFO", 50, ufoY )** instruerar KPL att göra.

Eftersom instruktionen är inuti **FOR**-slingan, kommer KPL att utföra instruktionen varje gång den räknar från 1 till 150. Det är det som är själva vitsen och det som är så bra med slingor. Att bara räkna från 1 till 150 kanske inte är speciellt imponerande, men att utföra en uppgift varje gång vi räknar ett steg, det är det verkligt fiffiga med en slinga. Det är det datorer också är bra på, datorn tröttnar inte på att göra om samma sak 150 gånger! Låt oss studera vad som egentligen händer:

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

Vid det här laget vet du nog hur **MoveSpriteToPoint** fungerar: KPL flyttar bildobjektet **"UFO"** till den (X, Y)-punkt som vi anger. OK, det visste du, men vad är skillnaden den här gången? Det första vi gör i programmet är att flytta "UFO" till punkten (50, 0). Men vad händer när vi sedan gör det i slingan istället och med variabeln **ufoY** istället för en siffra? Kommer du ihåg att först gången KPL går igenom slingan så är värdet på **ufoY = 1**, andra gången är värdet på **ufoY = 2**, tredje gången är **ufoY = 3**, sedan är **ufoY = 4**, osv.... hela vägen up till **ufoY = 150**. Så den första gången KPL får instruktionen att flytta bildobjektet **MoveSpriteToPoint( "UFO", 50, ufoY )** är variabeln **ufoY=1** alltså som instruktionen nedan:

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

Andra gången KPL får instruktionen är **ufoY = 2**, så KPL får följande instruktion:

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

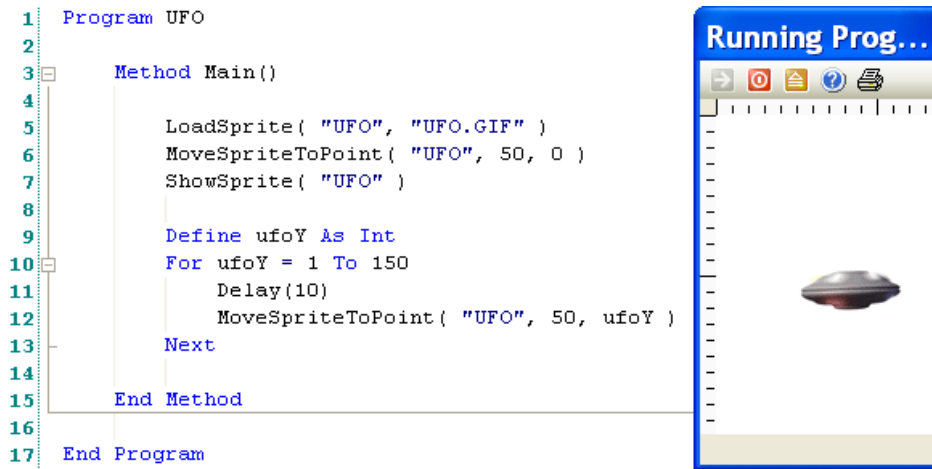
Sedan fortsätter det på samma sätt ända tills KPL har gått runt i slingan och räknat till 150:

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

Varje gång KPL går igenom ett varv i slingan så flyttar sig tefatet en punkt nedåt på skärmen precis som vi ville att det skulle göra!

Det var mycket på en gång, första variabeln, första slingan och dessutom en instruktion till KPL varje gång den gick ett varv runt slingan. Allt detta är mycket viktiga programmeringskunskaper! Om det inte är helt glasklart för dig, gå tillbaka till början av kapitlet och läs det en gång till.

Nu när du har fortsatt kan vi lägga till en detalj för att göra färdigt programmet:



Ser du vilken ny rad vi lagt till? Den enda nya raden är instruktionen **Delay (10)** [Fördröj(10)], på rad 11 inuti **For**-sligan. Var det verkligen nödvändigt? Jo, det är det, **för datorer är väldigt väldigt väldigt snabba!** Du hinner inte blinka under tiden datorn räknar från 1 till 150 och dessutom ritar tefatet på skärmen. Jag lovar! Datorer är så snabba! Om vi vill hinna se vårt tefat röra sig på skärmen behöver vi lägga in en fördröjning för att göra datorn lite långsammare. **Delay (10)** instruerar helt enkelt KPL att ta en liten rast (en väldigt kort rast) innan den flyttar tefatet.

Du kanske känner till att om du ska räkna hur många sekunder det går om du inte har en klocka så kan du räkna 1001, 1002, 1003 osv. för att få en liten fördröjning när du räknar? Det är precis på samma sätt med **Delay (10)** vi fördröjer KPL:s arbete lite. När du i nästa kapitel jobbar med programmet själv kan du prova att ändra värdet **10** till några andra värden för att prova vad som händer när du kör programmet. Prova **Delay (2)** kör programmet, byt till **Delay (100)** och kör igen. Det är lite skillnad va?

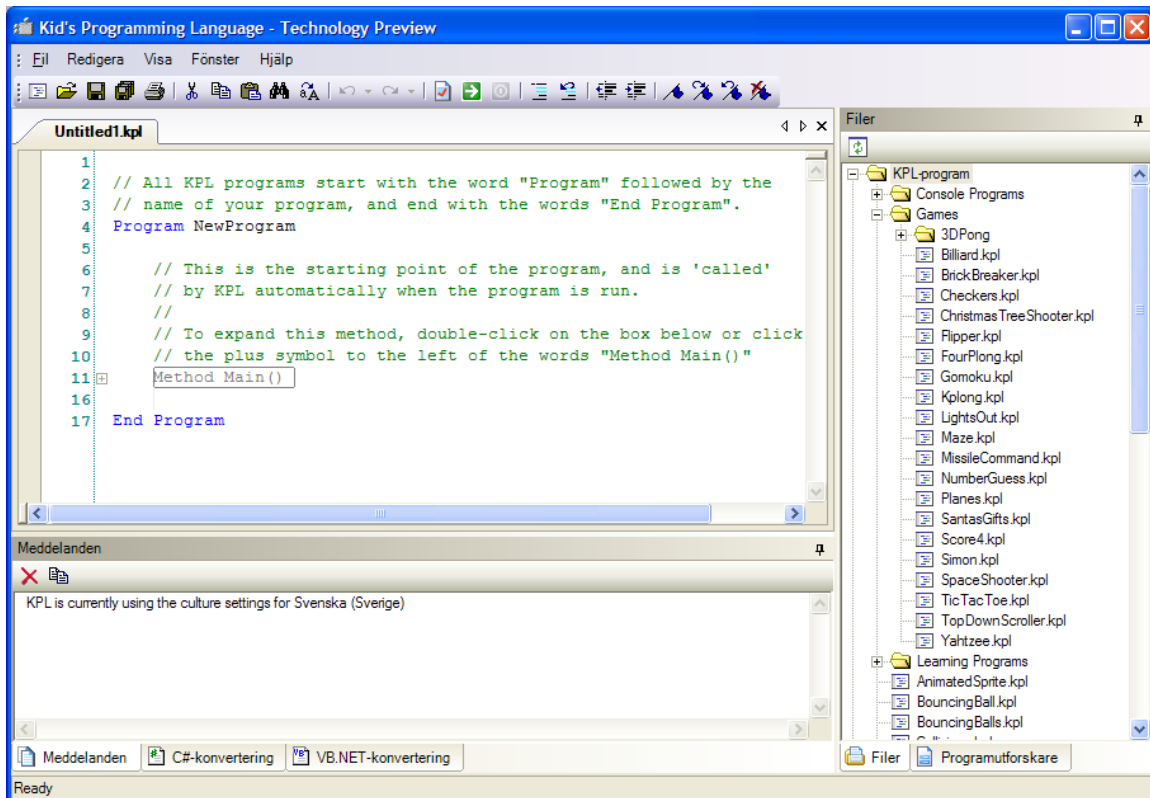
Nu har vi ägnat flera sidor åt att förklara detta program i detalj, vilket kanske får det att låta mer komplicerat än det är. Kom ihåg att det bara var åtta instruktioner, när du börjar få grepp om programmering i KPL kommer det gå snabbt att skriva åtta instruktioner. Förhoppningsvis håller du redan med om att det verkar ganska lätt att med bara några få KPL-instruktioner kunna rita lite cool grafik!



## Använda KPL på din dator

**Observera:** Den här handledningen baseras på de förändringar och förbättringar som införts i KPL från och med versionen som kom ut den 10 oktober 2005. Om du laddade ned KPL före den 10 oktober 2005, eller om programexemplen i denna handledning inte fungerar som dom beskrivs, ladda ned den senaste versionen från hemsidan: <http://www.kidsprogramminglanguage.com/download.htm>.

Detta kapitel förutsätter att KPL redan finns installerat på datorn och fokuserar på att du ska komma igång med användningen av KPL. Leta upp och starta KPL, du hittar KPL under Start/Program/Kids Programming Language. När du startat KPL ser det ut ungefär som bilden nedan:

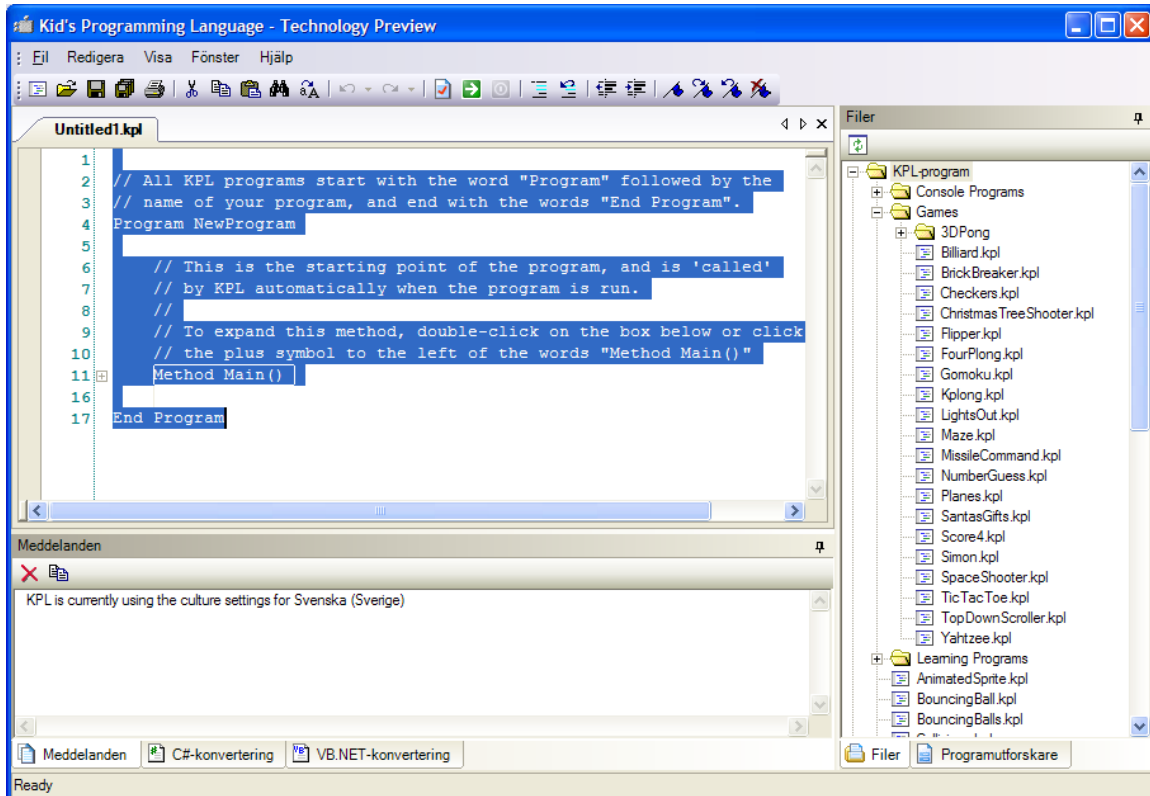


KPL startar med en ny KPL-program fil som heter "Untitled1.kpl" öppnad i kodfönstret. De gröna linjerna du ser i kodfönstret är kommentarer som ger information till den som läser programkoden, men som egentligen inte innehåller några instruktioner till datorn. Kommentarer föregås alltid av två stycken snedstreck: //

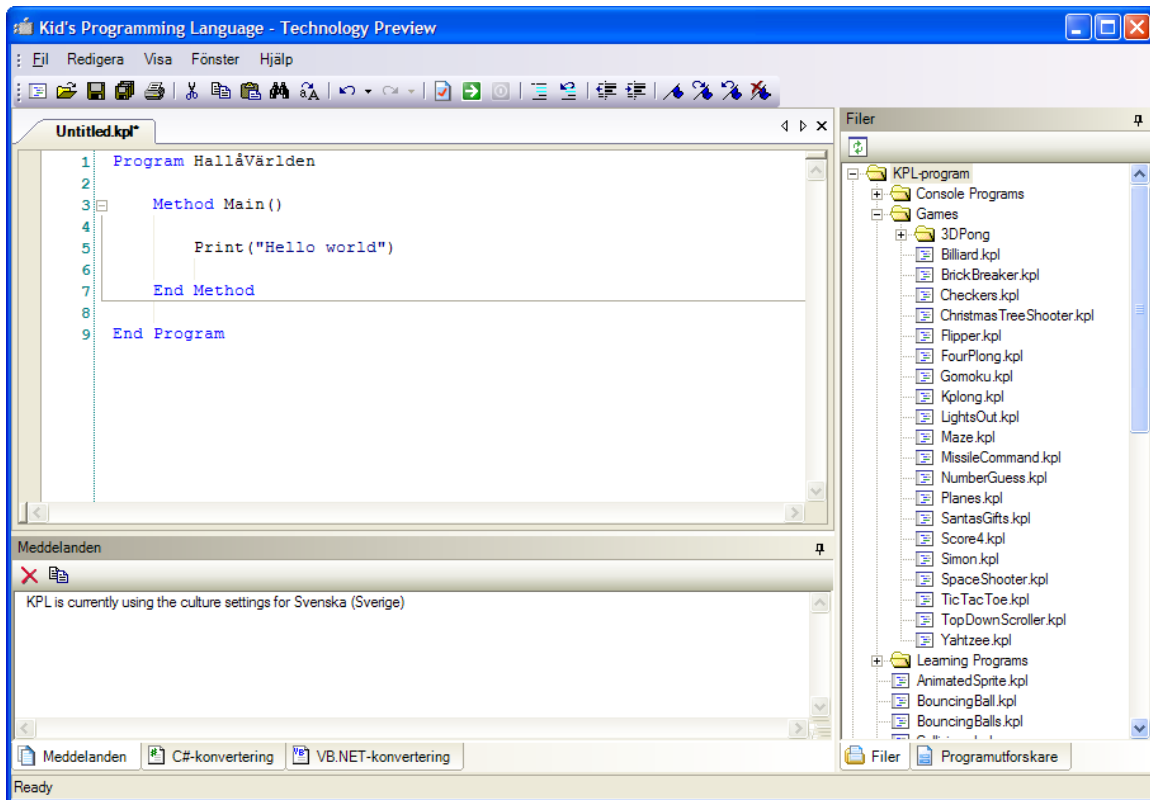
När du tittar i andra KPL-exempelprogram kommer du att se många kommentarer, med tiden kommer du säkert att börja använda kommentarer i dina egna KPL-program. Kommentarer hjälper andra att förstå vad som händer i programmet – en gång i framtiden kanske kommentarerna i koden kan hjälpa dig att förstå något du själv har skrivit men sedan glömt bort!

I KPL:s kodfönster finns många funktioner som du säkert känner igen från t.ex. ordbehandlings- eller e-postprogram. Ägna en stund åt att menyalternativen och verktygsraden. Om du för musen sakta över knapparna på verktygsraden kommer du att se små tipsrutor som beskriver vad knapparna är till för.


Vi kommer att fokusera på att göra om de tre exempelprogram som du redan har läst om tidigare i denna handledning. Så låt oss börja med att rensa koden som du ser i filen "Untitled1.kpl" i kodfönstret. För att göra detta, klicka på **Redigera** i menyn och välj **Markera allt**. Du kommer att se att all kod som finns i fönstret nu är markerad, som nedan. När all kod är markerad kan du antingen trycka på **Delete** på tangentbordet eller välja **Redigera** och **Klipp ut** på menyn och hela markeringen (dvs. all kod i det här fallet) kommer att försvinna.

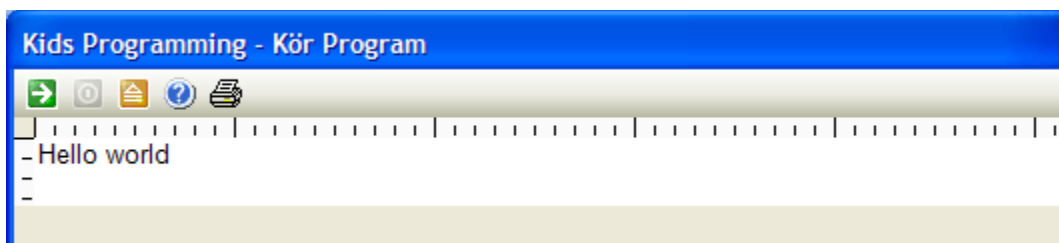


När väl den gamla koden är raderad kan du gå vidare och skriva in ditt första KPL-program som i exemplet nedan:



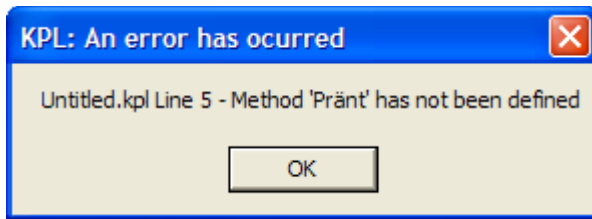
Kom ihåg att datorer inte har någon fantasi utan kräver väldigt exakta instruktioner – så försök att skriva av texten ovan så exakt som möjligt. Om du skriver annorlunda kanske det inte fungerar. För att skjuta in texten ett eller två steg från raden ovanför kan du använda **TAB**-tangenter. För att skapa en tom rad trycker du på **ENTER**-tangenter. Att använda indrag och tomma rader är inte nödvändigt men det gör koden enklare att läsa.

När du har skrivit in koden med instruktionerna, klicka på knappen med den gröna pilen  eller tryck F5 på tangentbordet för att starta ditt KPL-program. Om KPL-koden är inskriven rätt kommer ett fönster att visas, som nedan men lite större:



**Så var det med den saken! Nu är du en programmerare! Snyggt jobbat! Ett fyrfaldigt leve för ....** okej, okej, jag kanske ska lugna ner mig lite. Det är kanske inte det mest komplicerade programmet i världen men det är ett program, du har skrivit det och fått det att fungera!

Om KPL inte förstår instruktionerna i koden kommer ett felmeddelande att visas, t.ex. som detta nedan:

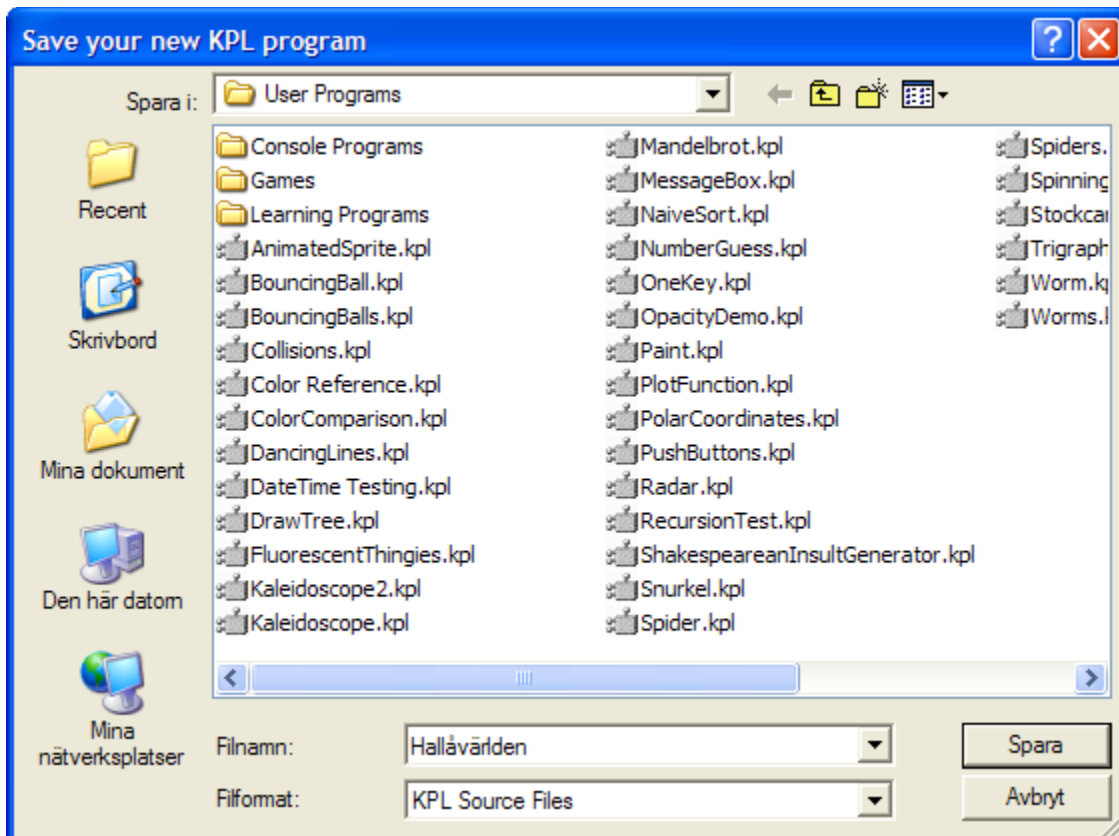


(istället för Print("Hello world") har jag skrivit Pränt("Hello world"))

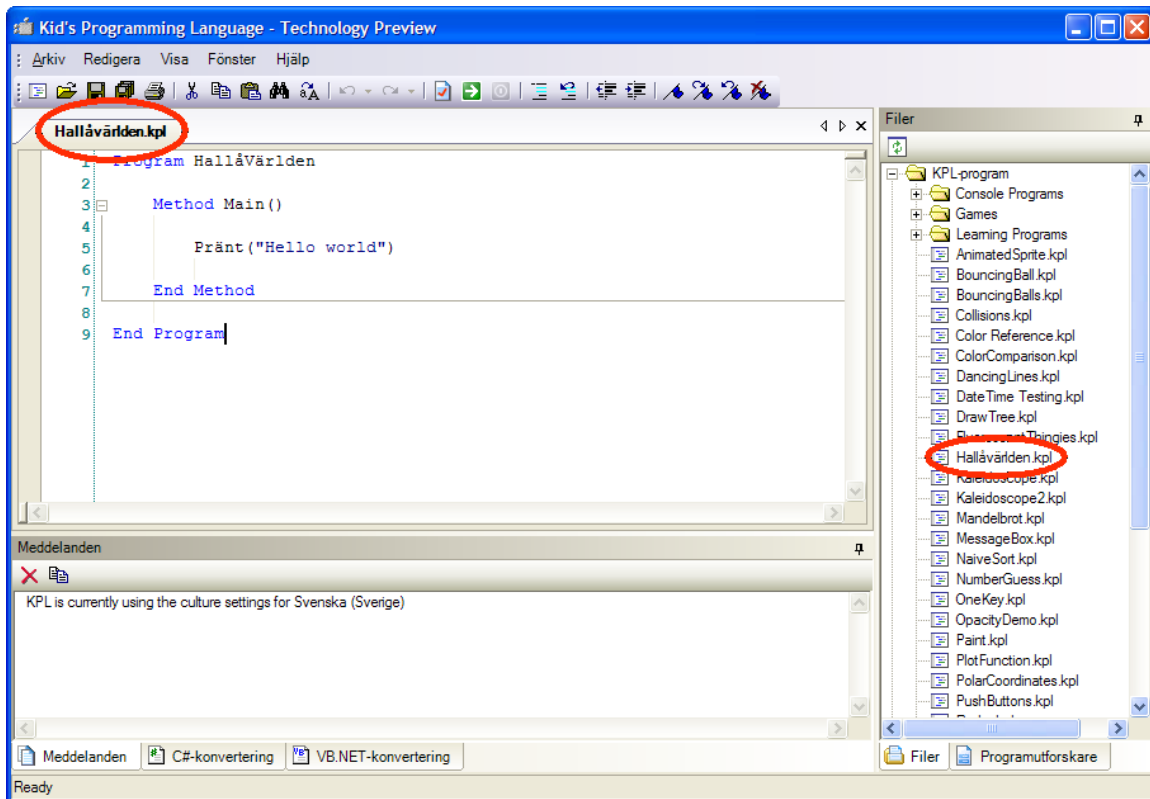
Om det skulle hända, klicka på **OK**, och titta noga på det du har skrivit och jämför med texten i exemplet ovan. Det är ofta bra att börja leta på den rad som felmeddelandet säger är fel, i meddelandet ovan är det rad 5 (Line 5). När du hittar det som blivit fel rättar du det och provar igen, då går det säkert bättre!

**Alla programmerare skriver ibland fel i sin kod**, förr eller senare kommer det hända dig med, men ge inte upp för det, ingen är perfekt! När det händer, var lugn, noggrann och gå igenom koden du skrivit ordentligt. Koncentrera dig på ny kod du skrivit om programmet fungerat tidigare, kolla också raden som anges i felmeddelandet.

När du fått exemplet "HelloWorld" att fungera, klicka på Arkiv i menyn och sedan alternativet **Spara**, och döp programmet till "Hallåvärlden" som exemplet visar:

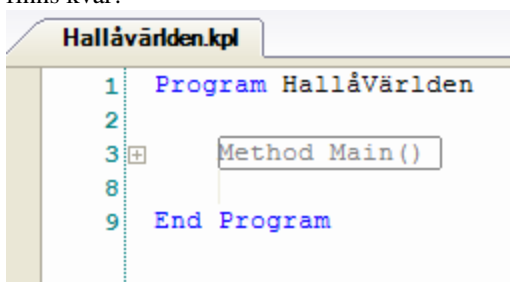


När du sparat programmet kommer du att se bevis för det på två ställen:



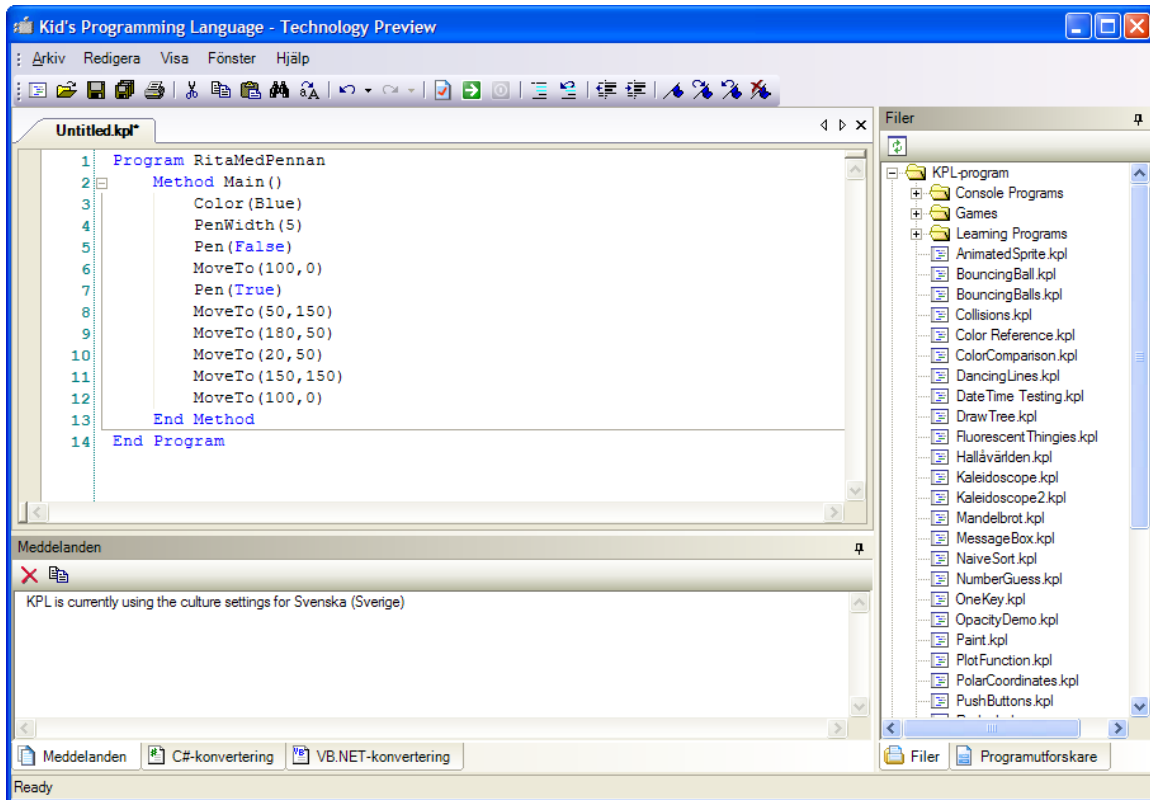
Ta för vana att spara ditt program innan du ska prova de senaste förbättringarna du gjort, ibland så kan man ha sån otur att programmet stannar och då är det rätt skönt att ha kvar de senaste ändringarna.

Nu när du har sparat programmet till hårddisken kan du när som helst öppna det igen genom att dubbelklicka på filnamnet i fönstret "Filer" till höger. När du öppnar ett sparat KPL-program, kommer du säkert att märka att det inte riktigt ser ut som tidigare, utan som bilden nedan, men få inte panik, all din kod finns kvar!



Klicka på det lilla +-tecknet på rad 3 och all kod som finns i Method Main kommer att "vecklas ut" så att du kan se den igen. Istället för ett plustecken är det nu minustecken på rad 3, klicka på det för att "fälla ihop" koden igen. Detta kanske är lite onödigt för ett så enkelt program, men det är väldigt praktiskt när du gör större program och vill koncentrera dig på en sak i taget.

När du har sparat HallåVärlden, klicka på **Arkiv** i menyn och välj **Nytt kodfönster**, då får du fram ett nytt "Untitled.kpl"-program, precis som när du började. Markera allt och radera som du gjorde förut och skriv sedan in koden för vårt nästa exempel:



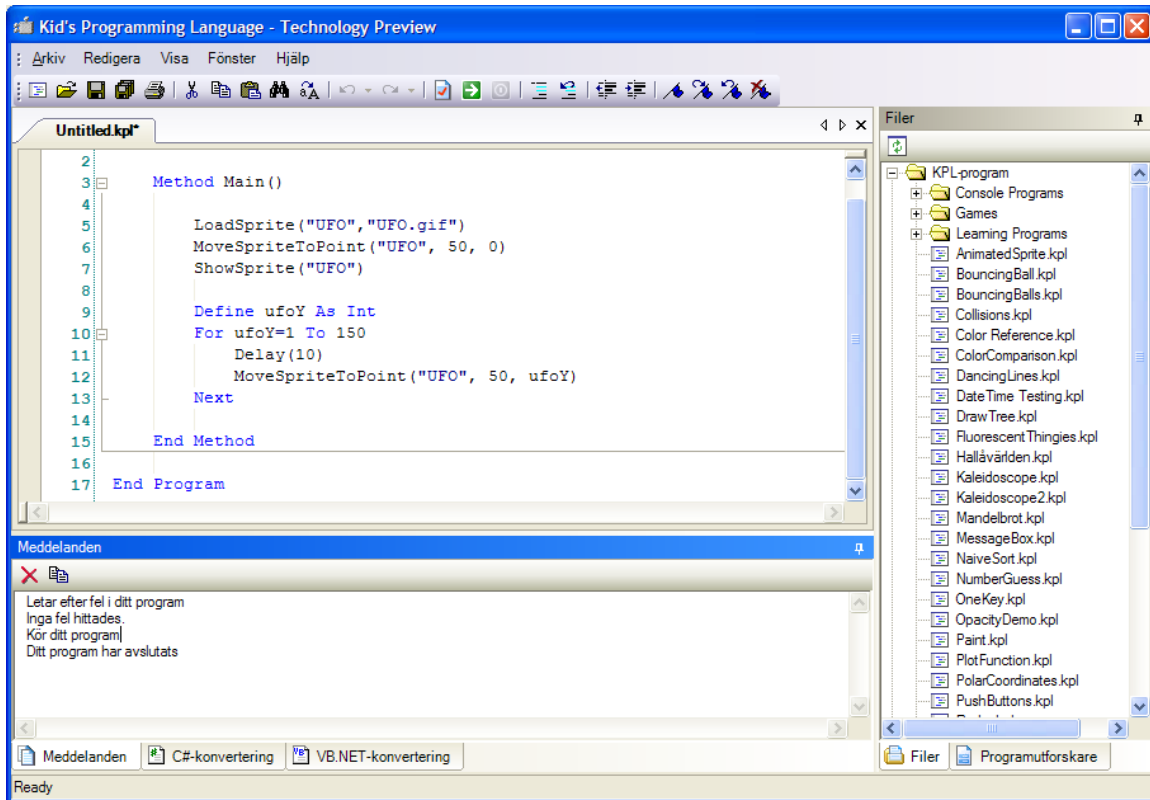
Om du skriver in KPL-koden exakt som den står ovan och kör den, kommer du att få se den blå stjärnan och du har avancerat ett steg till en datorgrafikprogrammerare! **Bra jobbat!**

Kom bara ihåg att om du råkar ut för något fel, var lugn och tålmodig, leta noggrant i koden och jämför med exemplet för att hitta skillnader. När du gjort de ändringar som behövs för att det ska bli precis lika, prova att köra igen.

**Och du, glöm nu inte bort att spara ditt program!**

Det här programmet kan du experimentera lite med på skoj. Vilka andra färger fungerar, testa både siffror och engelska ord för färgerna. Kan man rita tjockare eller smalare linjer? Vilka siffror behöver du skriva in för att det ska bli en triangel eller fyrkant istället för stjärnan? Eller en svår uppgift, hur gör du stjärnan större eller mindre? Testa!

När du sparat RitaMedPennan, klicka på **Arkiv** i menyn och välj **Nytt kodfönster**, då får du fram ännu ett nytt "Untitled.kpl"-program, precis som förut. Radera koden som du gjorde tidigare och skriv in koden för vårt UFO-exempel:



Om du skriver in KPL-koden exakt som den står ovan och kör den, kommer du att få se det flygande tefatet!  
**Snyggt jobbat!**

Kom bara ihåg att om du råkar ut för något fel, var lugn och tålmodig, leta noggrant i koden och jämför med exemplet för att hitta skillnader. När du gjort de ändringar som behövs för att det ska bli precis lika, prova att köra igen.

**Glöm inte bort att spara ditt program!**

Prova några roliga förändringar av programmet om du vill ha lite extra övning:

- Kan du få tefatet att flytta sig **längre**?
- Kan du få tefatet att flyga **från vänster till höger** istället för att flytta sig nedåt?
- Kan du få tefatet att flyga **från vänster till höger samtidigt som det flyttar sig nedåt**??

## Nästa steg efter denna handbok

När du läst igenom handboken och kommit hit så kan du ändå fortsätta att lära dig. Det första stället du kan leta efter mer information är i mappen ”**Learning Programs**”. Där finns ytterligare 6 KPL-program som du kan lära dig av. Dom är numrerade och det bästa är om du öppnar dom i tur och ordning. Dom två första programmen är väldigt lika program som redan har gjorts i den här handboken! Det är möjligt att det kommer fler exempelprogram (eller redan har kommit fler) i så fall är dom en del av installationen eller så kan du ladda ned dom på vår sida för nedladdning på [www.kidsprogramminglanguage.com](http://www.kidsprogramminglanguage.com).

Du kanske tar ännu ett steg och laddar ned **KPL User Guide for Teachers**, vilken också finns tillgänglig på samma nedladdningssida. Det är inte en handbok för nybörjare men kan ändå vara en bra fortsättning för dig när du går vidare och arbetar med dina egna KPL-program.

När du känner att du behärskar exempelprogrammen (se ovan) så finns det fler program, med i KPL-installationen, som du kan öppna och studera. **Kplong.kpl** och **NumberGuess.kpl** är två speciellt bra program att titta vidare på. Du hittar dom i mappen ”**Games**”. Det är betydligt större program – och fullt fungerande spel – än exempelprogrammen, men har du kommit så långt så kommer du förmodligen inte ha några problem med att förstå och lära dig från dom programmen.

Nedan visas några länkar till webbforumet för KPL-användare – du kan hitta samma länkar på hemsidan [www.kidsprogramminglanguage.com](http://www.kidsprogramminglanguage.com) :

[Cool KPL programs](#) – Coola KPL-program

[Kids' Discussion of KPL](#) - KPL-diskussion för barn

[Parents' discussion of KPL](#) – KPL-diskussion för föräldrar

[Teachers' discussion of KPL](#) – KPL-diskussion för lärare

[KPL Questions and Answers](#) - Frågor och svar om KPL

[International Language Versions of KPL](#) – Internationella versioner av KPL

Om du har frågor eller vill ha mer hjälp med KPL så är forumet ”KPL Questions and Answers” det första ställe du bör leta på. Forumet är redan välbesökt och det blir mer och mer aktivt hela tiden!