Arduino

Hur funkar det? - Arduino

Arduino är det fantastiska kretskortet som har tagit maker-världen med storm. Arduino gör det lätt för vem som helst att börja bygga elektroniksaker. Det kan röra sig om allt från väckarklockor och klimatsensorer till robotar och konstnärliga ljussättningar. För att kunna bygga Arduino-projekt krävs grundläggande kunskaper inom elektronik och programmering. I den här delen av hur funkar det går vi igenom just detta.

God läsning!

Vad är Arduino och Genuino?

Arduino kan styra det mesta

Det ständigt växande utbudet av mobilappar är ett tydligt tecken på att intresset för programmering ökar. Idag kan alla som lär sig programmera skapa häftiga och användbara appar till datorer, mobiler och surfplattor. Det kan handla om allt från enkla blinkande appar till avancerade bildredigeringsprogram (för att inte tala om alla spel).

Programmering behöver inte vara begränsat till appar som visas på en skärm. Idag finns kod som någon programmerare har skrivit i nästan alla saker som kopplas till elnätet eller drivs på batteri. Mikrovågsugnen har en timermjukvara som styr hur länge maten ska värmas. TV-fjärrkontrollen har en mjukvara med kommandon som skickas när någon trycker på dess knappar. Robotdammsugaren kör en avancerad mjukvara som styr hur dammsugaren rör sig, tar sig förbi hinder och hittar tillbaka till sin laddstation.

Arduino är lösningen som gör att allt från nybörjare till erfarna programmerare kan skriva kod som körs utanför mobilens, surfplattans eller datorns skärm. Med Arduino kan alla som har intresset bygga egna elektronikprojekt. Det kan röra sig om allt från ljusdetektorer och väckarklockor till mobiltelefoner och radiostyrda bilar. På kjll.cm/arduino-ideas finns en lång lista med för närvarande över 50 olika projektidéer (med tillhörande bygg- och programmeringsinstruktioner).

Ännu fler exempel på vad som går att åstadkomma med Arduino finns självfallet på Youtube. Se exempelvis svenska Simone Giertz som har blivit världsberömd för sina "shitty robots". 2015 blev hennes frukostmaskin (en Arduinobaserad robotarm som serverar klassisk frukostmat) en viral succé. Hon har sedan dess publicerat många fler roliga projekt, bland annat en popcornmatare.

Simone Giertz popcornmatare (fotograf: Alba Giertz)

Utvecklingskort

Som det nämndes tidigare i detta kapitel styrs allt från mikrovågsugnar till robotdammsugare av kod. Den koden körs på en mikrokontroller som sitter i produkten. Mikrokontrollern är en liten dator i ett chip som innehåller processor, minne samt in- och utgångar. Hur snabb processorn är, hur stort minnet är och hur många in- och utgångar den har varierar från modell till modell. En enkel uppgift (t.ex. räkna ned tiden på en mikrovågsugn) kan utföras med ett fåtal rader kod och en förhållandevis klen mikrokontroller. En avancerad uppgift (t.ex. styra en robotdammsugare) kräver betydligt fler kodrader och en kraftfullare mikrokontroller.

Oavsett modell på mikrokontroller och vilken uppgift som ska utföras måste mikrokontrollern programmeras på något sätt (d.v.s. koden måste komma in i mikrokontrollern). Det krävs normalt en fysisk mikrokontrollerprogrammerare för att få koden från datorn där den skrivs till mikrokontrollern. För att slippa behovet av en sådan använder Arduino-plattformen utvecklingskort.

Ett utvecklingskort är ett kretskort med en förmonterad mikrokontroller och nödvändig kringelektronik. Utvecklingskortet kan anslutas till en dator (vanligtvis via USB) och därigenom göra det enkelt att föra över kod från datorn till mikrokontrollern. Ett av de populäraste utvecklingskorten är Arduino Uno (rev. 3) som för övrigt är utvecklingskortet som vi kommer att utgå från i denna bok. På Arduino Uno sitter en mikrokontroller som heter ATmega328. Den programmeras för att styra allt som kopplas till utvecklingskortet.

Utvecklingskortet Arduino Uno kopplas till datorn med USB och har en mikrokontroller som heter ATmega328.

Relaterade produkter

Nedan ser du ett urval av smarta hem-produkter från vårt sortiment. Klicka in på en produkt för att läsa mer om den eller scrolla vidare för att fortsätta läsa artikeln.

Kom igång med Arduino

Arduino
Introduktion

- Vad är Arduino och Genuino?
- Kom igång med Arduino
- Arduino-hårdvara
- Arduino IDE
- Hello World!

Grundläggande elläraGrundläggande programmeringGrundläggande elektronikArduino-projekt

Vad behövs?

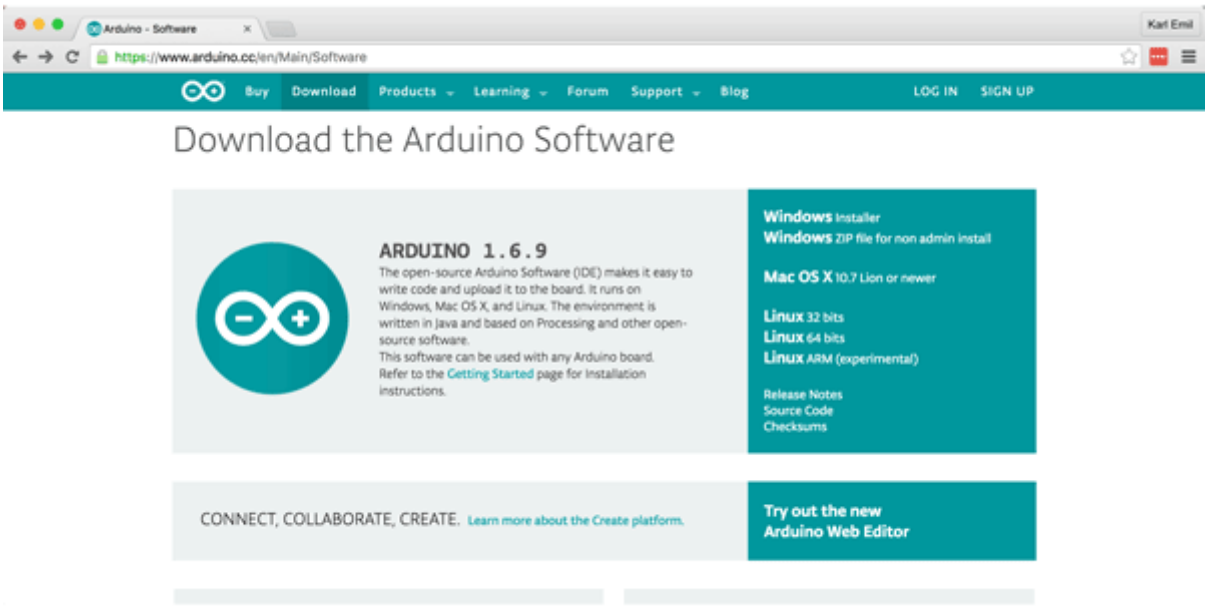
För att komma igång med Arduino behöver du följande saker.

- En dator med Windows 7 eller senare alternativt Mac OS 10.7 (Lion) eller senare. I exemplen kommer vi att använda Windows 10 eller Mac OS 10.12 (Sierra). Det går även att använda Linux, men vi har valt att begränsa bokens omfattning till Windows och Mac OS.
- Ett Arduino Uno-utvecklingskort eller ett Arduino Uno-kompatibelt utvecklingskort (d.v.s. ett utvecklingskort som fungerar på samma sätt som de officiella motsvarigheterna). Ett sådant kort följer med i bokens tillhörande komponentkit.
- En USB-B-kabel (följer med i bokens tillhörande komponentkit).

Ladda ned Arduino IDE

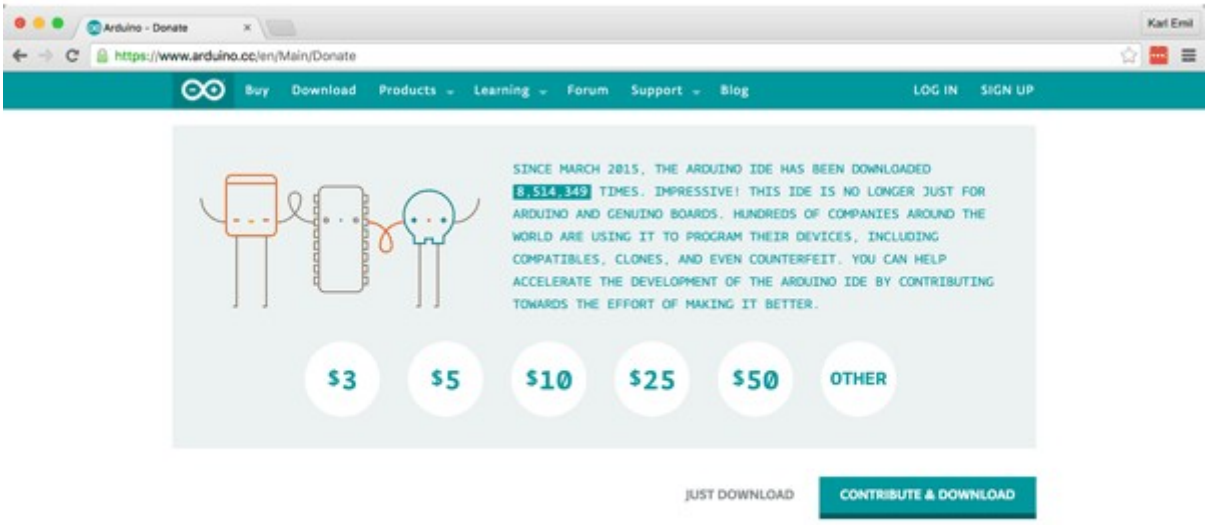
Börja med att öppna webbläsaren och gå till kjll.cm/arduino-103. Välj att ladda ned Windows-versionen eller Mac OS-versionen beroende på vilket operativsystem din dator kör.

Det finns två versioner för Windows: *Windows installer* och *Windows Zip*. Ladda ned versionen som heter Windows installer, så att allt installeras automatiskt. Den enda fördelen med Zip-versionen är att den inte behöver installeras och därför inte kräver administratörsrättigheter på datorn.



Ladda ned Arduino IDE.

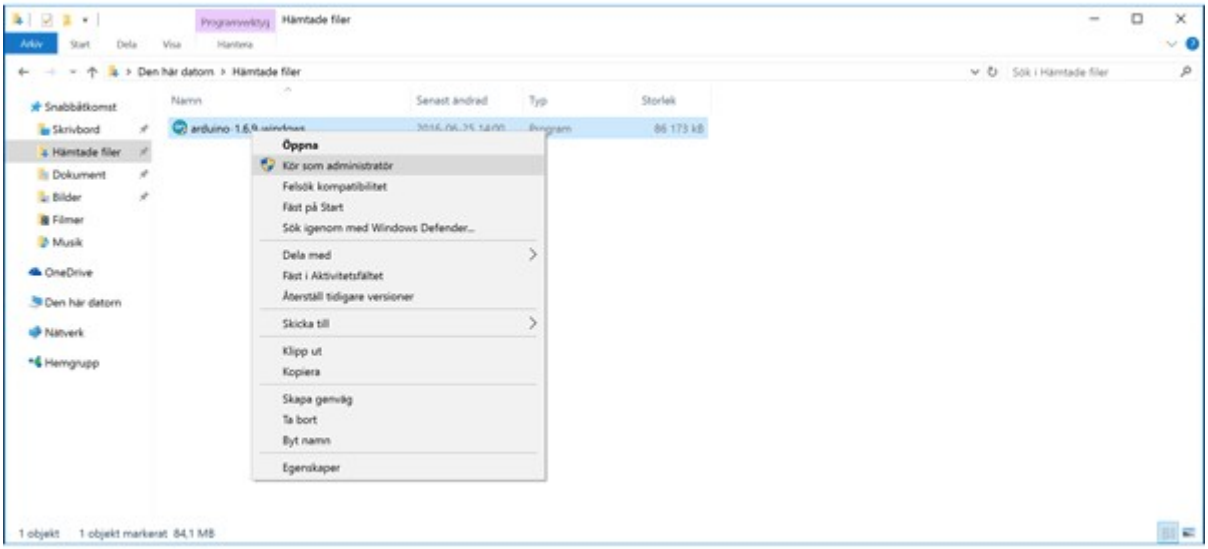
Innan nedladdningen påbörjas får du möjlighet att donera pengar till utvecklingen av utvecklingsmiljön. Självfallet finns möjligheten att ladda ned utan att donera pengar.



Det är frivilligt att donera pengar till utvecklingen av utvecklingsmiljön.

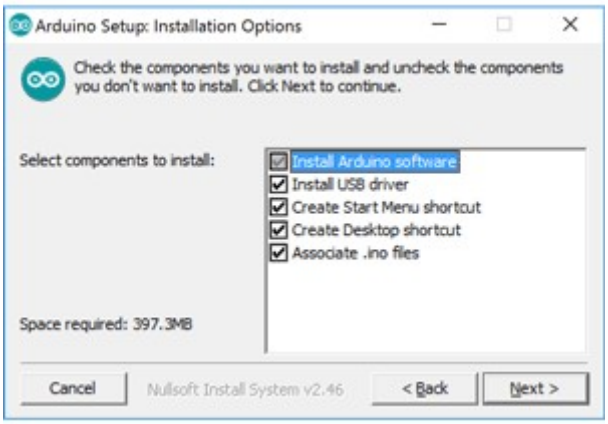
Installera Arduino IDE i Windows

Filen som laddas ned till Windows-datorer är ett exekverbart program (gäller Windows installer-alternativet). Högerklicka på programmet och välj att köra det som administratör.



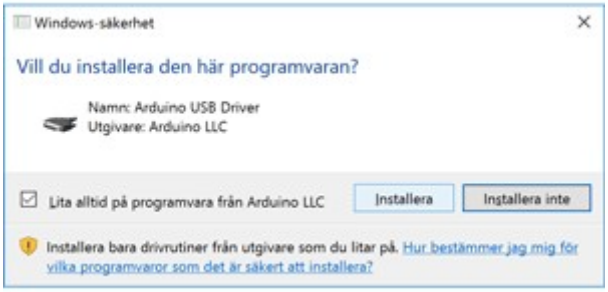
Kör installationsprogrammet som administratör.

Välj att installera *Arduino software* (själva mjukvaran) och USB *driver* (drivrutin för utvecklingskort). Kryssa i de övriga rutorna om du även vill ha en genväg på startmenyn, en genväg på skrivbordet och att .ino-filer öppnas i Arduino IDE som standard.



Installera både Arduino software och USB driver.

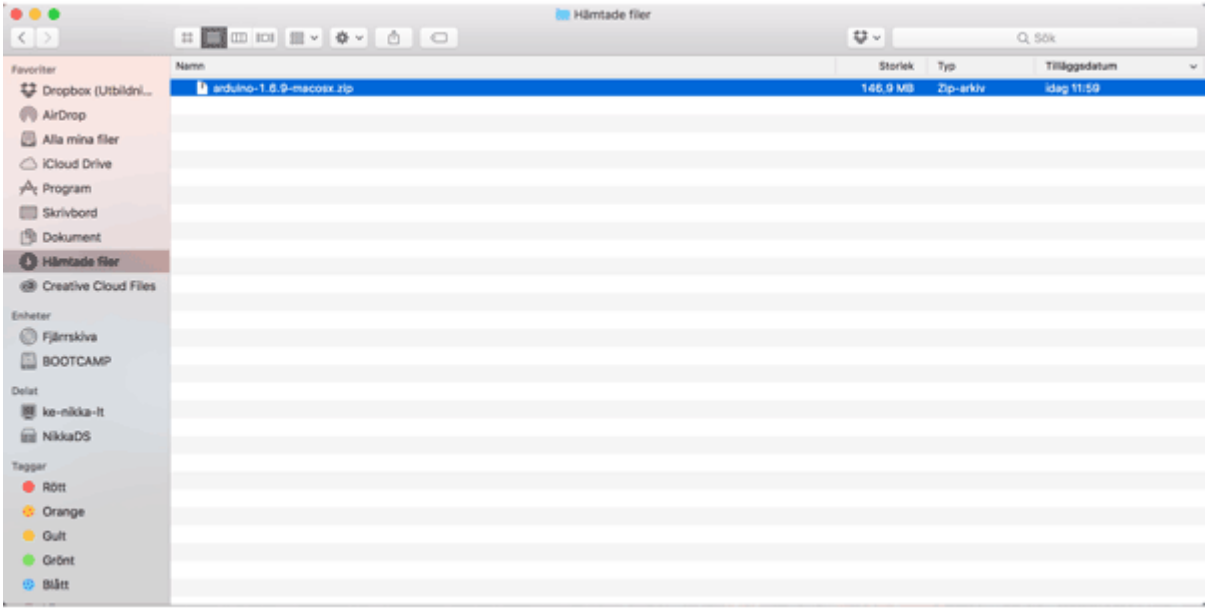
Godkänn att installera drivrutinen för utvecklingskort.



Godkänn installationen av drivrutin.

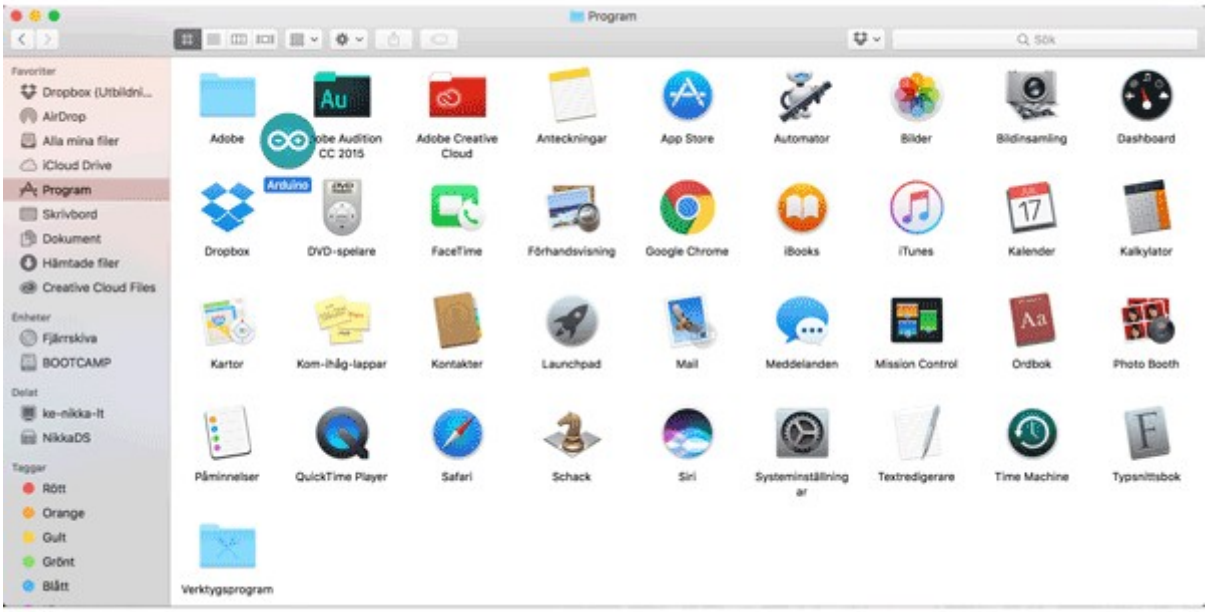
Installera Arduino IDE i Mac OS

Filen som laddas ned till Macar är ett Zip-arkiv. Dubbelklicka på det nedladdade Zip-arkivet för att packa upp det.



Packa upp Zip-arkivet.

Zip-arkivet innehåller applikationen Arduino.app (filändelsen visas inte). Flytta den till mappen *Program* genom att dra den dit i Finder.



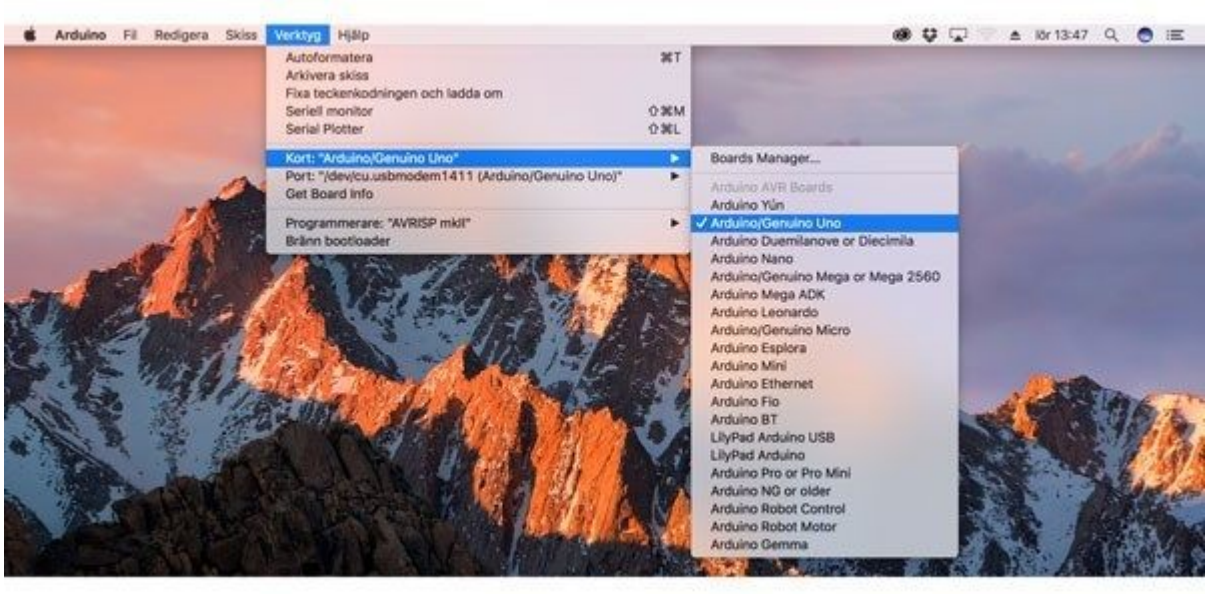
Flytta applikationen Arduino.app till Program.

Dubbelklicka på Arduino för att starta applikationen. Om Gatekeeper i Mac OS hindrar dig från att starta applikationen kan du lösa det genom att högerklicka på applikationen och välja *Öppna*.

Konfigurera Arduino IDE i Windows eller Mac OS

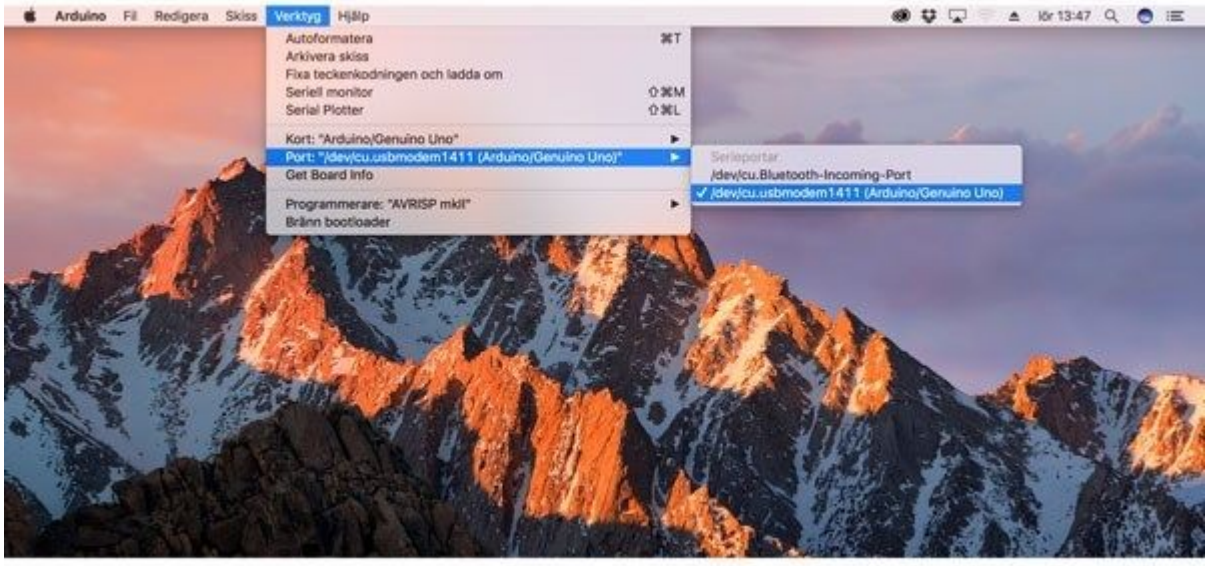
Avsluta installationen med att konfigurera utvecklingsmiljön. Anslut ditt Arduino Uno-utvecklingskort till en ledig USB-port på datorn och starta applikationen Arduino (Arduino IDE).

Klicka på *Verktyg* och välj utvecklingskortet *Arduino/Genuino Uno.*



Välj utvecklingskortet Arduino/Genuino Uno.

Välj porten för ditt utvecklingskort. I Windows heter porten något i stil med COM3 (Arduino/Genino Uno). I Mac OS heter porten något i stil med /dev/cu.usbmodem1411 (Arduino/Genuino Uno).



Välj porten för ditt utvecklingskort.

Om du kan välja ditt utvecklingskort fungerar allt som det ska. Ifall något skulle bråka kan troligtvis den officiella felsökningssidan vara till stor hjälp. Där beskrivs lösningarna på de vanligaste problemen som kan uppkomma (engelska). Besök kjll.cm/arduino-104.

Arduino-hårdvara

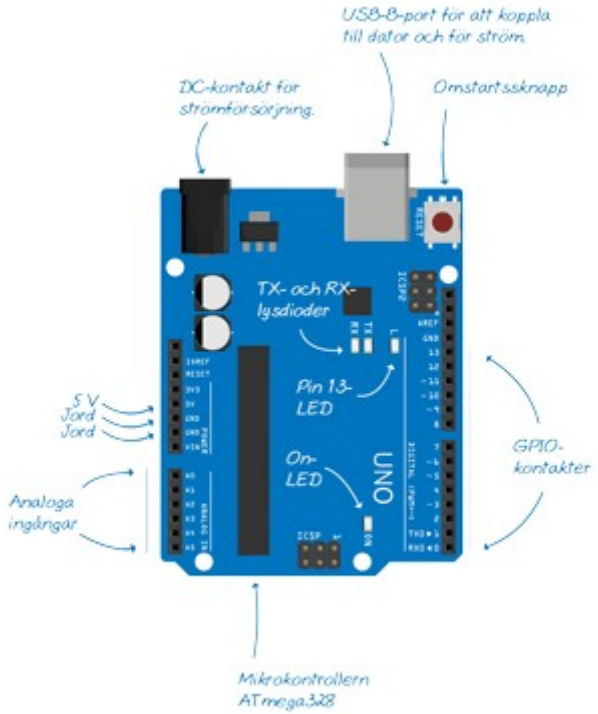
Utvecklingskortet Arduino Uno

Uno är det mest använda Arduino-utvecklingskortet. Det tillverkas av många olika företag som alla följer samma ritning. Det spelar därför mindre roll vem som har tillverkat kortet.

Utvecklingskortet finns i olika revisioner. I skrivande stund är det revision tre som används och som vi utgår från i denna bok. Boken kräver inte att du har den senaste revisionen av utvecklingskortet, så det går även att följa exemplen med en äldre revision. Undantaget är de sista projekten som inte kan köras på äldre utvecklingskort då de har mindre minne än revision tre.

Arduino Uno har en USB-B-kontakt för att kopplas till datorn. USB-B-kontakten kan också användas för strömförsörjning av utvecklingskortet och många av komponenterna som kopplas på det. Om Arduino-utvecklingskortet ska användas fristående (utan att vara kopplat till en dator) kan det strömförsörjas med en nätadapter eller ett batteri som kopplas till den runda DC-kontakten (spänningen ska vara mellan 7 V och 12 V).

Tips! Det går även att strömförsörja Arduino Uno med en så kallad powerbank (ett uppladdningsbart batteripaket med USB-kontakt som normalt används för att ladda mobiler på resan).



Översiktsbild över Arduino Uno[2].

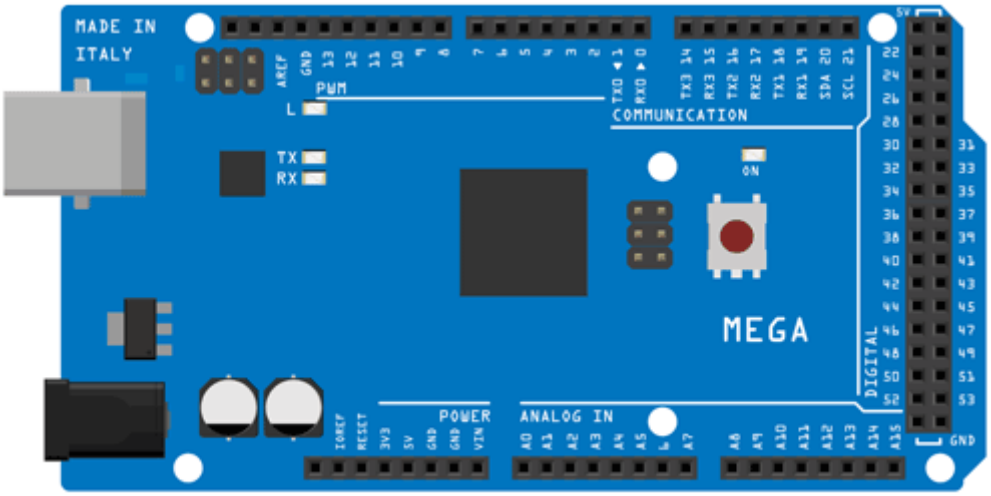
Längs med sidorna av utvecklingskortet sitter långa kontaktlister. Den ena kontaktlisten är numrerad 0 till 13. Dessa kontakter kallas digitala GPIO-kontakter (General Purpose Input Output). De används exempelvis för att styra lysdioder och registrera när någon trycker på en knapp. På motsatt sida sitter sex analoga ingångar. De används för att koppla in bland annat analoga sensorer (t.ex. temperatursensorer). Strax intill de analoga ingångarna sitter ytterligare en liten kontaktlist med bland annat ett 5 V-stift och två jordstift. Dessa stift används för att strömförsörja de flesta saker som kopplas till utvecklingskortet.

Utvecklingskortet är även utrustat med en uppsättning lysdioder som är praktiska för bland annat felsökning. Den gröna On-lysdioden visar att utvecklingskortet är igång. TX- och RX-lysdioderna används för att visa när data skickas respektive tas emot av utvecklingskortet. De ska blinka högfrekvent när data överförs till och från utvecklingskortet. Sist men inte minst finns lysdioden som kallas Pin 13-LED. Den kommer till nytta i kapitel 4 Arduino IDE.

Utvecklingskortet är även utrustat med en omstartsknapp. Den används om det skulle bli problem och utvecklingskortet behöver startas om.

Övriga utvecklingskort

Arduino Uno är långt ifrån det enda utvecklingskortet som finns. Ett annat exempel är Arduino Mega som är en storasystermodell till Arduino Uno. Arduino Mega har betydligt fler portar, mer minne och snabbare processorkärna. Den är populär till stora och komplexa byggen.



Arduino Mega[3]

Adafruit är en av de riktigt stora tillverkarna av Arduino-tillbehör och Arduino-utvecklingskort (de är till och med en officiell tillverkare). De har tagit fram ett litet utvecklingskort som heter Trinket som har blivit väldigt populärt för sin kompakta storlek och låga pris. Nackdelen med Trinket är att utvecklingskortet har färre portar, mindre minne och långsammare processorkärna än Arduino Uno.

Kopplingsplatta

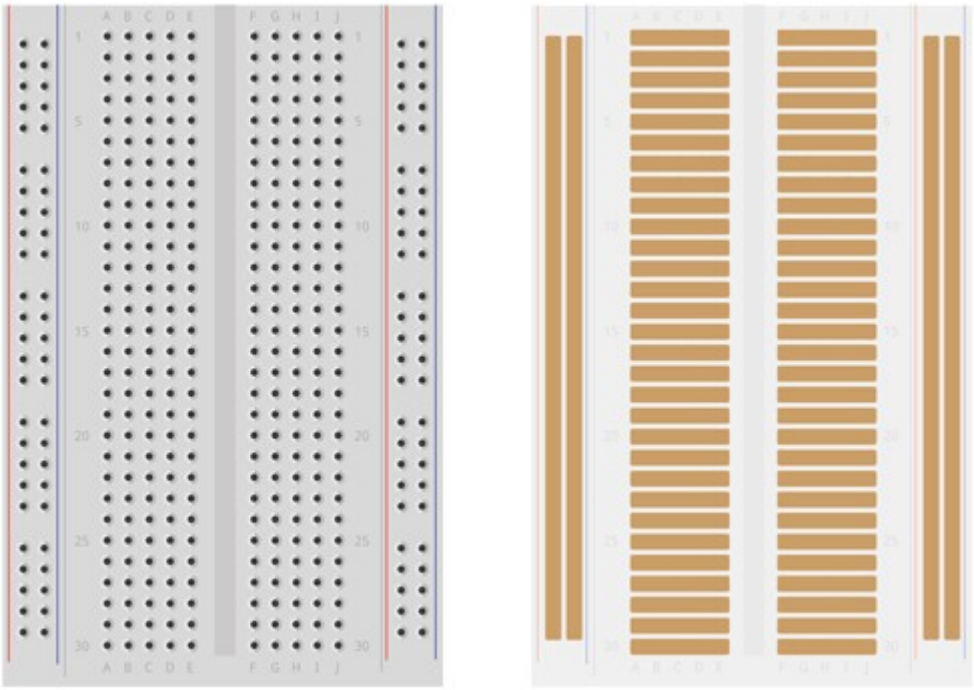
Det är inte praktiskt att koppla komponenter direkt på utvecklingskortet. I övningar och prototyparbete kopplas komponenterna i stället på en kopplingsplatta, vilken i sin tur kopplas till

utvecklingskortet. Kopplingsplattor kräver ingen lödning och gör det därför snabbt och enkelt att ansluta komponenter.

Kopplingsplattor finns i många olika utföranden. I komponentkittet som hör till denna bok finns en av de vanligaste kopplingsplattorna. Den har 30 rader med tio hål per rad som komponenter kan kopplas till. Längs med båda sidorna av kopplingsplattan finns hål som normalt används för strömförsörjning.

Lägg märke till att samtliga hål är namngivna utifrån ett koordinatsystem (raderna har nummer och kolumnerna har bokstäver).

Lägg även märke till hur hålen är uppdelade på kopplingsplattan. Segmenteringen indikerar hur de olika hålen är sammanlänkade med varandra eller frånskilda från varandra.



Sättet som hålen på en vanlig kopplingsplatta är sammanlänkade.[4]

Relaterade produkter

Nedan ser du ett urval av smarta hem-produkter från vårt sortiment. Klicka in på en produkt för att läsa mer om den eller scrolla vidare för att fortsätta läsa artikeln.

Modul

Med en kopplingsplatta går det att bygga precis vad som helst. Med moduler går det snabbare att bygga det. Moduler är kretskort som har komponenterna färdigkopplade på sig, så att de kan anslutas direkt till utvecklingskortet med ett fåtal kopplingar. Det sparar mycket tid.

Tryckknappar används frekvent i Arduino-sammanhang. De består av en fjädrande brytare och en resistor. För att slippa koppla ihop tryckknappens komponenter varje gång en tryckknapp behövs kan en färdig modul användas i stället.

Moduler används också för att ansluta komplexa saker till Arduino-utvecklingskort. Det är praktiskt för saker som hade varit svåra att koppla ihop själv. Ett exempel på detta är ett relämodulkort som hade tagit mycket lång tid att sätta ihop på en kopplingsplatta. Kopplingsplattan hade dessutom behövt vara väldigt stor.

Relämodul ansluten till Arduino Uno.

Relaterade produkter

Nedan ser du ett urval av smarta hem-produkter från vårt sortiment. Klicka in på en produkt för att läsa mer om den eller scrolla vidare för att fortsätta läsa artikeln.

Arduino IDE

Arduino
Introduktion

- Vad är Arduino och Genuino?
- Kom igång med Arduino
- Arduino-hårdvara
- Arduino IDE
- Hello World!

Grundläggande elläraGrundläggande programmeringGrundläggande elektronikArduino-projekt

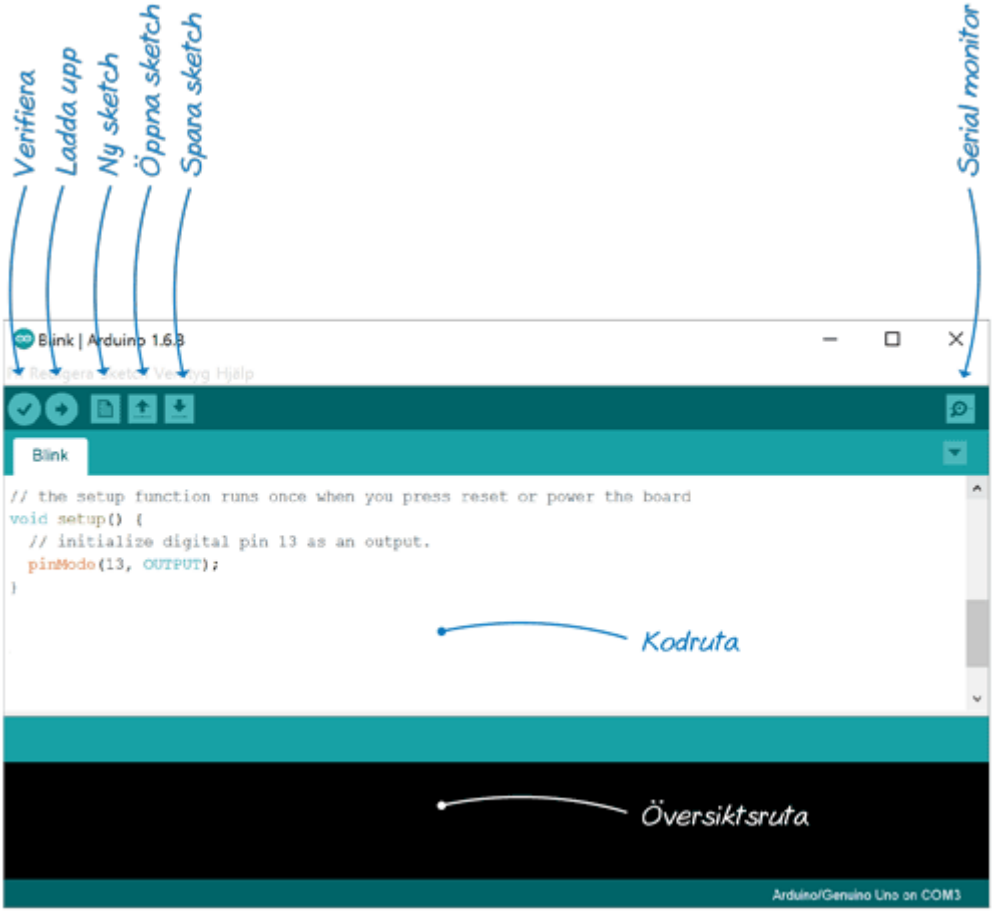
Överblick över utvecklingsmiljön

Utvecklingsmiljön består av en stor vit textruta där själva koden skrivs. Koden som skrivs där kallas sketch och lite längre fram i detta kapitel visar vi hur en sketch ser ut och hur den skrivs. Ovanför kodrutan finns knappar för sex av de vanligaste åtgärderna:

- verifiera en sketch
- kompilera och ladda upp en sketch till Arduino
- skapa en ny sketch
- öppna en befintlig sketch
- spara en sketch
- visa Serial monitor.

Vad de nämnda åtgärderna innebär och hur de används förklaras längre fram i detta kapitel.

Under kodrutan finns en svart ruta. Den använder utvecklingsmiljön för att berätta vad den håller på med. Den används också för att delge eventuella felmeddelanden.



Översikt över Arduino IDE.

Utvecklingsmiljön fyller många funktioner. Den låter programmeraren göra bland annat dessa fyra saker på ett smidigt sätt:

- skriva kod (sketcher)
- verifiera kod
- kompilera kod
- ladda upp kod.

Skriva sketcher

Sketchen beskriver vad Arduinon ska göra. Om vi exempelvis vill att en lysdiod ska blinka kan vi skriva en sketch för det. Om utvecklingsmiljön hade förstått svenska, hade vi kunnat skriva en kod som såg ut på följande vis.

```
Sätt GPIO-stift 13 som en utgång.

Gör följande saker om och om igen:
* slå på lysdiod på stift 13
* vänta 1000 millisekunder
* slå av lysdiod på stift 13
* vänta 1000 millisekunder.
```

Eftersom Arduino-utvecklingsmiljön inte förstår svenska måste vi skriva vår kod på ett språk som både vi och utvecklingsmiljön förstår: Arduino Programming Language (APL). Som tur är det går det lätt att översätta svenskan till APL! Då ser sketchen ut på följande vis.

```
void setup() {
 pinMode(13, OUTPUT);
}

void loop() {
 digitalWrite(13, HIGH);
 delay(1000);

 digitalWrite(13, LOW);
 delay(1000);
}
```

BlinkLed.ino

Tips! Sketchen ovanför är den klassiska nybörjarsketchen inom Arduino-programmering. Faktum är att den redan finns på din dator! Klicka på *Fil, Exempel, 01. Basics* och välj *Blink*. På exempelmenyn hittar du många andra bra exempel som kan vara till stor hjälp när du skriver dina första sketcher.

Sketchen består av två funktioner (vad en funktion är förklaras i Funktioner) som kallas *setup* respektive *loop*. Dessa två funktioner måste finnas med i alla sketcher, men avancerade sketcher har ofta fler funktioner.

Allting som står i setup är det som Arduinon bara gör en gång (när den startar). Vi vill börja med att göra något med GPIO-stift 13. Det GPIO-stiftet har en lysdiod kopplat till sig (fastmonterat på utvecklingskortet), så vi behöver inte koppla någon hårdvara till utvecklingskortet för att testa denna sketch.

Eftersom GPIO-stift 13 både kan användas som en ingång och som en utgång måste vi välja vilken funktion som GPIO-stiftet ska fylla. Jämför detta med när du konverserar med en vän. Få människor kan prata och lyssna samtidigt, så du måste bestämma dig för vilket av det du vill göra. Arduinon kan "prata" och "lyssna" på olika stift, men den kan inte prata och lyssna samtidigt på samma stift. Vi vill att GPIO-stift 13 ska användas som en utgång (för att "prata").

Allting som står i loop är det som Arduinon ska göra om och om igen. Vi vill att lysdioden på GPIO-stift 13 ska vara tänd i en sekund (1000 millisekunder) och därefter vara släckt lika länge. Så ska den fortsätta att växla läge i all oändlighet. Vi skriver därför att Arduinon först ska sätta hög spänning på GPIO-stift 13 (det får lysdioden att lysa) och därefter låg spänning på GPIO-stift 13 (det får lysdioden att slockna). För att lysdioden ska blinka i rätt takt åtskiljer vi de två instruktionerna med en fördröjning (ett delay).

För att dokumentera vad vi gjort och vad vi vill åstadkomma lägger vi in kommentarer. Kommentarerna kan vi skriva på vilket språk som helst. Vi väljer att skriva dem på engelska eftersom det har blivit standardspråket för kommentarer i programmeringskod. Om vi skulle ladda upp vår sketch på ett forum för att få hjälp med felsökning är det praktiskt om alla som läser sketchen kan förstå kommentarerna.

Vi inleder alla kommentarer med två snedstreck (//). Detta indikerar för utvecklingsmiljön att allt som kommer efter på samma rad är kommentarer och inte något som hör till själva programmet.

Välskriven kod är alltid väldokumenterad, så att även andra programmerare kan förstå vad koden gör!

```
/*
  Example sketch BlinkLedBinary.ino version 7.01.
  From Kjell & Company's book "Hur funkar Arduino?" version 7.01.
  From Kjell & Company's book "Hvordan virker Arduino?" version 7.01.

  This code is in the public domain.
*/

void setup() {

  // Use GPIO Pin 13 as output.
  pinMode(13, OUTPUT);
}

void loop() {

  // Turn on LED and then wait 1000 ms.
  digitalWrite(13, true);
  delay(1000);

  // Turn off LED and then wait 1000 ms.
  digitalWrite(13, 0);
  delay(1000);
}
```

Se hela filen

https://github.com/kjellcompany/Arduino_701/blob/master/BlinkLedBinary/BlinkLedBinary.ino

Prova att skriva sketchen på din dator. Lägg märke till att det finns många skiljetecken med i koden. Dessa skiljetecken måste skrivas för att koden ska tolkas rätt.

När du har skrivit sketchen kan du spara den på din dator genom att klicka på Spara-knappen eller välja *Spara* på Fil-menyn. Välj ett filnamn som enbart består av bokstäverna A till Z.

Tips! Du kan välja var dina sketcher sparas som standard i inställningsrutan som du når via Fil-menyn.

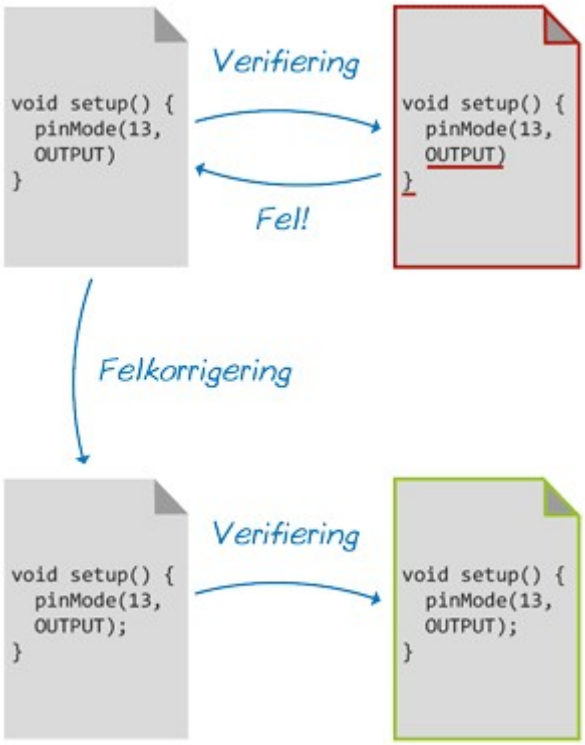
Verifiera kod

Vi människor är ganska bra på att förstå varandra även om vi inte uttrycker oss hundra procent korrekt. Våra datorer (inkl. Arduino) är däremot värdelösa på att förstå vad vi menar om vi inte uttrycker oss korrekt.

Om du får frågan "vill du ha ett äpple eller ett päron" svarar du troligtvis antingen "ett äpple" eller "ett päron". Om en dator får motsvarande fråga svarar den "ja" eller "nej"! För att datorn ska förstå frågan måste den omformuleras till: vill du ha "ett äpple", "ett päron", "varken ett äpple eller ett päron" eller "både ett äpple och ett päron"?

Vi kan slarva med stavning och grammatik när vi skriver på svenska, men när vi skriver APL måste vi uttrycka oss helt korrekt. Stavar vi fel på ett ord förstår inte datorn vad vi menar. Slarvar vi med grammatiken (som i programmeringsvärlden kallas syntaxen) förstår datorn inte heller vad vi menar. Syntaxreglerna i APL är lyckligtvis lättare att lära sig än grammatiken i svenskan.

Likt rättstavningsprogrammet i Microsoft Word kan verifieringsfunktionen i Arduino IDE hjälpa oss kontrollera sketchen som vi har skrivit. Prova att klicka på Verifiera-knappen och se om sketchen du skrev är korrekt.

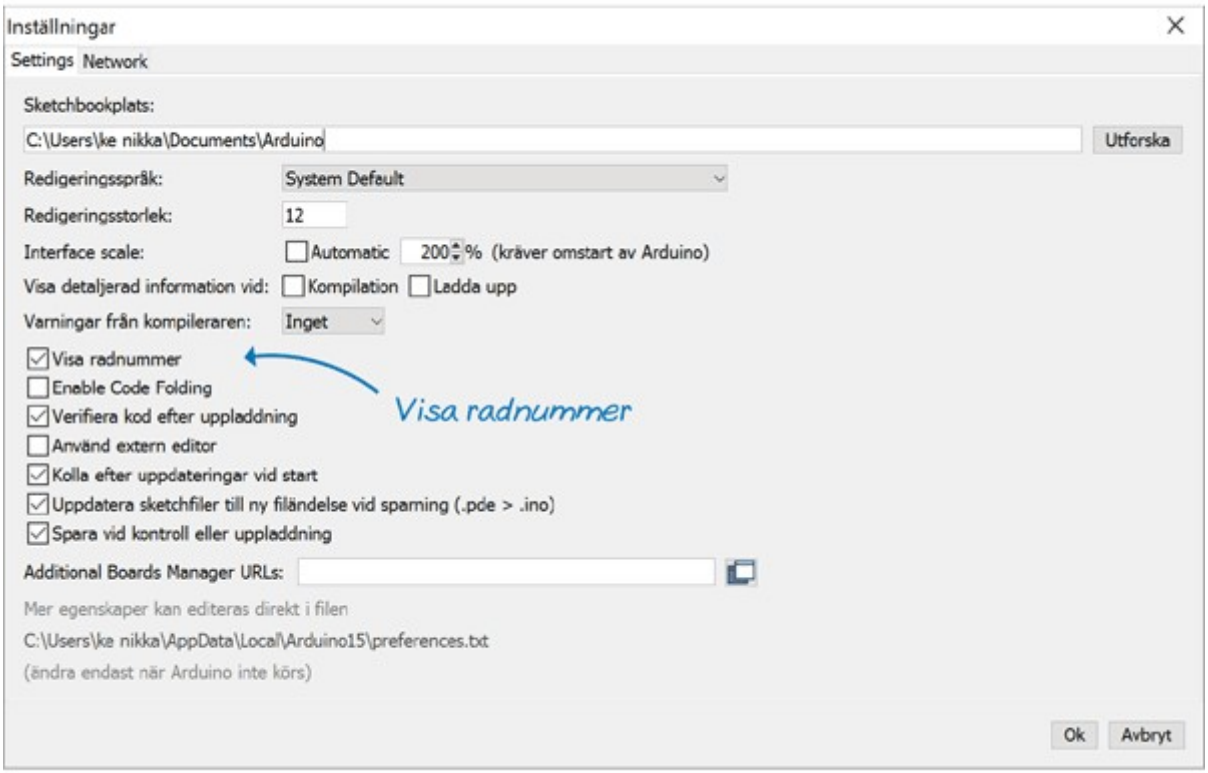


Verifieringsfunktionen i Arduino IDE fungerar som en rättstavningsfunktion.

Verifieringsfunktionen i Arduino IDE kan till och med hjälpa dig att korrigera en felskriven sketch. Om du exempelvis tar bort det sista semikolonet från din sketch och försöker att verifiera den på nytt kommer verifieringsfunktionen ge följande anmärkning.

Verifieringsfunktionen anmärker på att det borde finnas ett semikolon före denna slutparentes.

Tips! När communityn hjälper till med felsökning hänvisas ofta till radnummer. Genom att slå på radnummersvisning blir det lättare att felsöka. Klicka på Fil-menyn, välj *Inställningar* och *Visa radnummer*.



Slå på radnummersvisning för lättare felsökning.

Kompilera kod

Efter att ha läst denna bok kommer du att förstå APL. Det kommer dock aldrig din Arduino att göra! Den förstår enbart maskinkod, det vill säga kombinationer av ettor och nollor som instruerar Arduinon att göra något. Sketchen som du har skrivit måste därför kompileras.

Att kompilera något innebär att ta källkoden som programmeraren har skrivit (sketchen) och översätta den till maskinkod som datorn kan köra. Kompilatorn finns inbyggd i utvecklingsmiljön, så det räcker med att trycka på en knapp för att få sketchen kompilerad. Faktum är att du redan har kompilerat kod en gång! När du verifierade sketchen tidigare i detta kapitel, skickade du den till kompilatorn för att se om den kunde göra om din källkod till maskinkod.

Kompilatorn är en fantastisk uppfinning. Utan den hade vi behövt skriva kod som ser ut på följande vis (maskinkoden i exemplet är den du får när du kompilerar din sketch).

```
:100000000C945C000C946E000C946E000C946E00CA
:100010000C946E000C946E000C946E000C946E00A8
:100020000C946E000C946E000C946E000C946E0098
:100030000C946E000C946E000C946E000C946E0088
:100040000C9488000C946E000C946E000C946E005E
:100050000C946E000C946E000C946E000C946E0068
:100060000C946E000C946E0000000080002010069
:10007000000304070000000000000000102040863
:10008000102040800102040810200102040810200 2
```

```
:10009000040404040404040202020202020203032E
:1000A000030303030000000250028002B000000CC
:1000B0000000240027002A0011241FBECFEFD8E043
:1000C000DEBFCDBF21E0A0E0B1E001C01D92A930AC
:1000D000B207E1F70E9403020C9413020C94000093
:1000E00061E08DE00C94930161E08DE00E94CC0111
:1000F00068EE73E080E090E00E94F50060E08DE043
:100100000E94CC0168EE73E080E090E00C94F50072
:100110001F920F920FB60F9211242F933F938F933C
:100120009F93AF93BF9380910101909102 01A091A1
:10013000030 1B09104013091000123E0230F2D371A
:1001400020F40196A11DB11D05C026E8230F0296DB
:10015000A11DB11D2093000180930101909302 0124
:10016000A0930301B0930401809105019091060 1D1
:10017000A0910701B09108010196A11DB11D8093C6
:1001800005019093060 1A0930701B0930801BF9168
:10019000AF919F918F913F912F910F900FBE0F9034
:1001A0001F9018953FB7F894809105019091060132
:1001B000A0910701B091080126B5A89B05C02F3F6B
:1001C00019F00196A11DB11D3FBF6627782F892F19
:1001D0009A2F620F711D811D911D42E0660F771FDE
:1001E000881F991F4A95D1F708958F929F92AF92D9
:1001F000BF92CF92DF92EF92FF926B017C010E943F
:10020000D2004B015C01C114D104E104F104F1F00E
:100210000E9412020E94D200681979098A099B097A
:10022000683E73408105910570F321E0C21AD10840
:10023000E108F10888EE880E83E0981EA11CB11C2D
:10024000C114D104E104F10429F7DDCFFF90EF9050
:10025000DF90CF90BF90AF909F908F90089578944B
:1002600084B5826084BD84B5816084BD85B58260BB
:1002700085BD85B5816085BDEEE6F0E08081816059
:100280008083E1E8F0E0108280818260808380815 9
:1002900081608083E0E8F0E0808181608083E1EB31
:1002A000F0E0808184608083E0EBF0E08081816019
:1002B0008083EAE7F0E0808184608083808182 60CF
:1002C000808380818160808380818068808310 92B8
:1002D000C1000895833081F028F4813099F0823094
:1002E000A1F008958730A9F08830B9F08430D1F4B6
:1002F000809180008F7D03C0809180008F778093F4
:10030000800008958 4B58F7702C084B58F7D84BD49
:10031000895809 1B0008F7703C08091B0008F7DE9
:10032000893B0000895CF93DF9390E0FC01E458F0
:10033000FF4F2491FC01E057FF4F8491882349F13E
:1003400090E0880F991FFC01E255FF4FA591B491F1
:100350008C559F4FFC01C591D4919FB7611108C086
:10036000F8948C91209582238C93888182230AC0F3
:10037000623051F4F8948C91322F309583238C9312
:10038000888 1822B888304C0F8948C91822B8C9373
:100390009FBFDF91CF9108950F931F93CF93DF936A
:1003A0001F92CDB7DEB7282F30E0F901E859FF4F93
:1003B0008491F901E458FF4F1491F901E057FF4F80
:1003C00004910023C9F0882321F069830E946A0107
:1003D0006981E02FF0E0EE0FFF1FEC55FF4FA59174
:1003E000B4919FB7F8948C91611103C0109581234B
:1003F00001C0812B8C939FBF0F90DF91CF911F91F4
:100400000F91089508950E942F010E9402020E94F8
:10041000 7000C0E0D0E00E9474002097E1F30E94D9
:0A0420000000F9CF0895F894FFCF13
:00000001FF
```

Maskinkoden här i boken är inte skriven med ettor och nollor (binärt). Det hade tagit alltför många sidor i anspråk. Vi har i stället skrivit koden med siffrorna *0* till *9* och bokstäverna *A* till *F*. Detta

kallas hexadecimalt och används av programmerare som behöver ändra direkt i maskinkoden (vilket du lyckligtvis inte behöver göra tack vare kompilatorn). Den hexadecimala koden kan omvandlas till binär kod genom att helt enkelt ersätta sifforna och bokstäverna med följande kombinationer av fyra ettor och nollor. Läs mer om binära tal i datordelen av Hur funkar det?-serien om du är intresserad.

Hexadecimalt	Binärt
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Ladda upp kod

Anledningen till att du kan ta till dig informationen i denna bok är att du kan läsa. Om du inte hade kunnat läsa, hade texten på dessa sidor inte fyllt någon funktion. Minns du hur du lärde dig att läsa? Det var troligtvis inte med hjälp av en tjock textbok som hette "Konsten att läsa". En ren textbok kan inte lära någon läsa eftersom den förutsätter att personen som läser den redan kan läsa. Du lärde dig troligtvis att läsa genom att du fick intryck från bilder, lärare och föräldrar som successivt gjorde att du förstod läskonceptet.

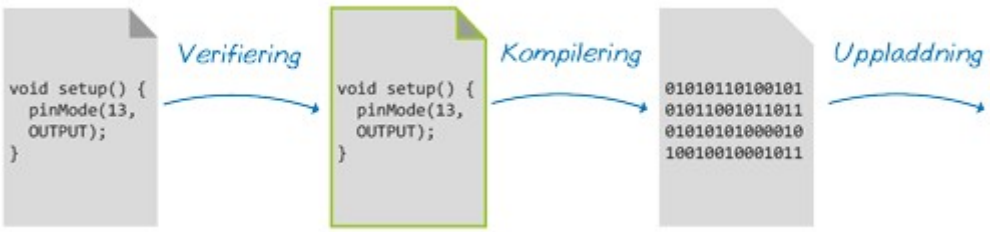
En mikrokontroller kan inte programmeras via USB. Den förstår inte vad det innebär att vara ansluten till en dator via USB. Precis som att vi inte kan läsa oss till hur vi ska läsa, kan inte heller en mikrokontroller lära sig använda USB-gränssnittet via USB-gränssnittet. Det ställer till problem på datorer som enbart är utrustade med USB-portar.

Detta "moment 22" löses i Arduino-världen med hjälp av en så kallad *bootloader*. Det är en mjukvara som "bränns" till mikrokontrollern genom en speciell hårdvara. Vi behöver inte ha tillgång till någon sådan hårdvara eftersom bootloadern redan är inbränd i mikrokontrollern på Arduino Uno.

Tack vare den förinstallerade bootloadern kan vi använda programmeraren som sitter på utvecklingskortet för att programmera vår Arduino. Det är den som gör att vi kan ansluta vårt utvecklingskort till en dator och trycka på en knapp för att få maskinkoden överförd.

Processen sammanfattad

Att programmera en Arduino sker i fyra steg. Det första vi gör är att skriva vår sketch (vår källkod[5]). Det gör vi i programmeringsspråket APL. Det andra vi gör är att verifiera koden så att vi inte har gjort några misstag. Om verifieringen går igenom kan vi kompilera vår källkod och få ut maskinkod. Den maskinkoden kan vi avslutningsvis, tack vare bootloadern, ladda upp till Arduinon och få Arduinon att göra något.



Sammanfattning över processen

Avsluta nu kapitlet med att göra just detta. Klicka på Ladda upp-symbolen så kommer utvecklingsmiljön att verifiera, kompilera och ladda upp din sketch. När maskinkoden laddas upp till Arduinon kommer TX- och RX-lysdioderna på utvecklingskortet blinka intensivt. Därefter kommer lysdioden som är kopplad till GPIO-stift 13 att blinka långsamt.

Tips! Du kan ändra hur snabbt lysdioden blinkar genom att ändra värdet som står inom parentes i *delay()*.

När maskinkoden är överförd till Arduinon behöver den inte längre vara kopplad till datorn. Allt den behöver ligger i mikrokontrollern. Det innebär att du kan koppla utvecklingskortet till exempelvis en powerbank med USB-port och använda din blinkande Arduino var du vill. Du kan även driva den på ett 9 V-batteri.

Referenser

5. Även känt som "Kjellkod".

Hello World!

Setup-funktionen

Vi börjar med att skapa en tom sketch och att spara den. Vi vill att namnet följer namngivningsschemat som används för sketcher inom APL. Vi namnger den därför *HelloWorld* (skrivet som det står stilistiskt med inledande versal på orden och utan mellanrum).

Utvecklingsmiljön har redan skapat de två funktionerna som alltid ska finnas (setup och loop). Vi kan därför inleda med att skriva koden som vi vill ha i setup-funktionen. Allting som står mellan klammerparenteserna är en del av setup (d.v.s. koden som körs när Arduinon startar). Varför det står void framför setup och varför det inte står något inom parentesen förklaras i *kapitel 13 Funktioner*.

I setup-funktionen skriver vi en kodsnutt så att resultatet ser ut på följande vis.

```
1  /*
2    Example sketch HelloWorld.ino version 7.01.
3    From Kjell & Company's book "Hur funkar Arduino?" version 7.01.
4    From Kjell & Company's book "Hvordan virker Arduino?" version 7.01.
5
6    This code is in the public domain.
7  */
8
9
10
11 void setup() {
12
13   // Open serial connection at 9600 Bd.
14   Serial.begin(9600);
15 }
16
17 void loop() {
18
19   // Print "Hello world!" in Serial monitor every second.
20   Serial.println("Hello world!");
21   delay(1000);
22 }
```

https://github.com/kjellcompany/Arduino_701/blob/master/HelloWorld/HelloWorld.ino

Eftersom vi vill att Arduinon ska skicka data till vår dator måste vi öppna en seriell anslutning (en port som gör att Arduinon och datorn kan kommunicera med varandra). Det gör vi genom att skriva *Serial.begin()*. Inom parentesen skriver vi hastigheten som vi vill överföra data i (d.v.s. hur snabbt Arduinon ska "prata"). Datahastigheten mäts i Baud (förkortas Bd) och vi vill överföra data i 9600 Bd. I denna kommunikation är 1 Bd (1 Baud) samma sak som 1 b/s (bit per sekund). Hastigheten vi

har valt är alltså 9600 bitar per sekund. Vi hade kunnat välja andra hastigheter också så länge mottagaren kan ta emot data (lyssna) i de hastigheterna.

Anledningen till att vi sätter denna instruktion (kodrad) i setup-funktionen (i stället för i loop-funktionen) är att vi enbart kan öppna anslutningen en gång (det ska inte göras varenda gång vi vill skicka något).

Vi avslutar instruktionen med ett semikolon. Detta gör vi efter alla instruktioner för att visa för kompilatorn var instruktionerna tar slut. När vi skriver på svenska gör vi våra läsare samma tjänst när vi använder punkter för att avsluta meningar.

Tips! Lägg märke till att vi inleder instruktionen med dubbelmellanrum. Det gör vi för att visa var i koden vi befinner oss. Eftersom vi är inne i en funktion börjar vi skriva lite längre in. Att justera vänstermarginalen hierarkiskt gör koden lättare att läsa.

Många programmerare föredrar att använda tabb i stället för dubbelmellanrum, vilket går precis lika bra.

Loop-funktionen

Utvecklingsmiljön har också skapat loop-funktionen åt oss, så vi kan börja fylla den med instruktioner. Precis som i setup-funktionen hoppar vi fram två steg (gör två mellanrum) för att tydligt visa var i koden vi befinner oss.

Vi vill att Arduinon ska skicka texten *Hello world!* till datorn. Det gör vi genom att anropa *Serial.println*. Println är en förkortning av "Print line" och innebär kort och gott att datorn skriver det som står inom parentesen och avslutar med radbrytning.

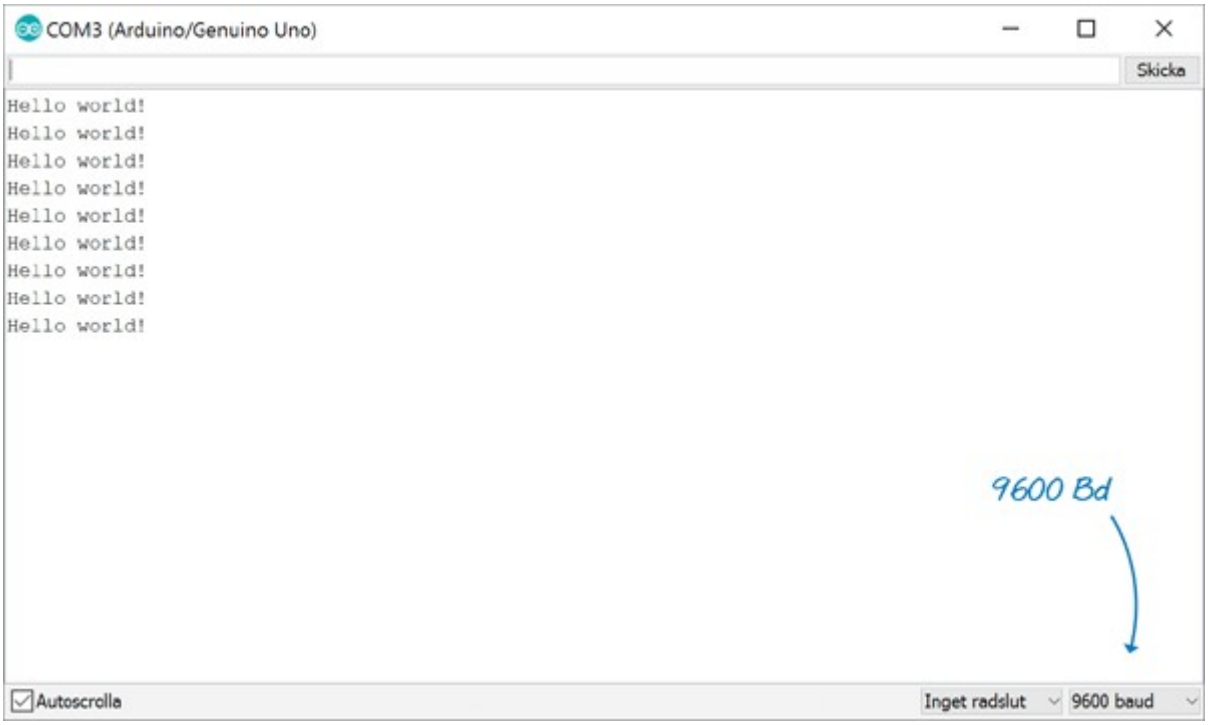
För att inte bli överösta med hälsningar lägger vi till en fördröjning. Vi använder *delay*, precis som i *Skriva sketcher*, och anger fördröjningen till en sekund (1000 ms).

Kör programmet

När vi har skrivit klart sketchen kan vi trycka på *Ladda upp* för att få sketchen verifierad, kompilerad och uppladdad. Om sketchen är korrekt skriven kommer Arduinon vara redo att skriva hälsningar på vår datorskärm efter några sekunder. Än så länge saknar den dock någonstans att skriva hälsningarna. Vi måste därför öppna Serial montior (knappen längst upp till höger).

Serial monitor

Längst ned till höger i Serial monitor kan vi ställa in datahastigheten. Den ska stå på 9600 Bd för att stämma överens med det vi skrev i vår sketch.



Den klassiska hälsningen i Serial monitor.

Tips! Du kan ladda ned bokens alla sketcher från github.com/kjellcompany. Vi rekommenderar dock att du skriver koden själv. Du lär dig otroligt mycket av att själv skriva koden, göra misstag och korrigera misstagen.