Vad är JavaScript?

JavaScript förkortas ofta som JS, men när vi talar om språket brukar vi använda hela namnet. Nyckeln till vad det är för typ av språk finns i namnet: JavaScript är ett skriptspråk. Till skillnad från HTML och CSS *är* JavaScript att klassa som ett programmeringsspråk, eftersom det kan utföra riktiga kommandon. Trots det kan vi inte riktigt säga att skriptspråk som JavaScript har samma status som andra programmeringsspråk.

Vad är ett skriptspråk?

Skriptspråk är som sagt en typ av programmeringsspråk. Skriptspråk specialiserar sig på att utföra uppgifter inom redan befintliga miljöer, som till exempel webbläsare. Källkoden till skriptspråk kallas vanligen skript, därav namnet. Just JavaScript används också på serversidan genom programsystemet Node.js

En skillnad mellan skriptspråk och andra programmeringsspråk, som C#, har att göra med deras prestanda. Skriptspråk är enklare att använda och programmera i, och riktigt bra på det de gör, men används generellt sett inte i mer avancerade program. Mindre och större funktioner på hemsidor är vad JavaScript gör bäst.

Nedan finns ett kortare exempel på hur JavaScript kan användas för att interagera och utföra ett kommando på en webbsida.

```
<html>
<body>

<p id="knapp">Tryck på knappen!</p>

<button type="button" onclick='document.getElementById("knapp")
.innerHTML = "Nu står det inte samma sak!"'>Klicka här!</button>

</body>
</html>
```

Hur används JavaScript med HTML?

Nu är det förhoppningsvis tydligt att JavaScript kan ta din hemsida till nya höjder, genom att ge dig funktioner som annars inte skulle finnas där. Men hur samspelar egentligen JavaScript med resterande kod?

För att få koden att samspela med HTML-koden används skript-taggen ”script”. Inom skrip-taggen skrivs koden du vill utföra. För att JavaScript-koden ska aktiveras måste den veta var och när koden ska köras. Det kan göras genom att ge exempelvis en knapp, en länk, eller en bild ett ID via HTML. Texten ovan har jag gett ID:t ”Knapp”. För att texten ska ändras har knappen kopplats ihop med samma ID, och en annan text.

JavaScript-koden behöver inte stå i anslutning till HTML-koden. Funktionen (kommandot) kan finnas skriven på en annan plats. På så sätt kan samma funktion användas på fler ställen.

Som sagt så är skriptspråken inte lika snabba som andra programmeringsspråk. Därför behöver man vara noggrann med att bara ha med de funktioner som *verkligen* lyfter hemsidan!

Medan HTML och CSS används för att strukturera upp din hemsida och göra den riktigt snygg, så används JavaScript för att göra din hemsida häftig. Och vem vill inte ha en häftig hemsida?

Några begrepp, en snabb överblick

Allt som du skriver tolkas. Siffror som **tal**, +-*/ som **operatorer** som manipulerar talen. Vissa ord och tecken som speciella **markörer**, **semikolon** som att nu är en instruktion slut. All annan text tolkas oftast som variabelnamn, om det inte står innanför **citattecken** vilket gör dem till helt vanlig text, **strängar**. **Variabler** har olika **typer**. De kan vara strängar (strings), listor (arrays), heltal (integers), decimaltal (floats), objekt (objects) och funktioner (functions), faktiskt mer.

Ett **kodblock** är en bit kod, ofta omfamnad av måsvingas, curly brackets, {}, och skiljer sig inte från annan kod på något sätt annat en just det att de skiljer sig. Som en paragraf i en text.

Loopar är kodblock som repeteras, som utförs ett visst antal gånger. Tillexempel: ??k framåt *tills det blir rött ljus*, eller, Häll i en deciliter vatten i kastrullen *fyra gånger*. I dessa fall så skulle den kursiverade texten bestämma hur många gånger kodblocket (den andra texten) ska utföras.

If-satser är en fråga med tillhörande åtgärder. Om svaret är ja så utförs ett kodblock, annars ett annat, eller inget alls om man så önskar. Som hjälp för att konstruera själva frågan så finns det **logiska operatorer** och **jämförelse operatorer**, som lika med (==), mindre än (<), större än (>), och (&&), eller (||) och inte (!).

En **funktion** är ett kodblock som utförs endast när den blir kallad till att göra så. Man **anropar** den (får den att utföra det den är gjord för) genom att sätta två parenteser efter dess namn, som i *alert()*. Om du endast skulle skriva *alert* så skulle det endast vara en **referens** till funktionen, men inte anropa den.

Listor består av artiklar (**items**) som i sin tur är en variabel och som har ett **index**. Ett item's index är alltid ett heltal och unikt i just den listan. Genom att skriva två **hakparenteser**, [], med ett index i, efter listans namn, så får man item:et i listan med det indexet, tillexempel *minLista[1]*, för att få det andra item:et i listan *minLista* (indexet börjar på noll).

Objekt är som listor, fast istället för artikel kallas de för properties (egenskaper), och deras värde (som fortfarande kan vara vilken sorts variabel som helst) kallas **value**, och istället för index så har de nycklar (som är unika i objektet), **keys**, som är strängar. Så när man vill få tag på ett specifikt värde med en nykel, säg color i objektet myPaper, så skriver man *myPaper["color"]* eller, vilket endast går om nyckeln är ett sammanhängande ord och inte börjar på en siffra, *myPaper.color*.

Kommentarer i koden är till för att göra den mer lättläst och för att hjälpa både sig själv och andra att förstå vad som verkligen händer. Allt som kommer efter två framåtslashar, //, på den raden de står, blir kommentarer och därefter ignoreras av tolken. Blockkommentarer är oftast längre kommentarer som kan sträcka sig över flera rader, vilket är användbart om man vill inaktivera en viss kodsnutt utan att ta bort den från resten av koden. Detta gör man med /* i början, och */ i slutet.

Variabler

Innan man använder en variabel bör man deklarera den, och säga att den finns. I många andra språk måste man i deklarationen bestämma sig för vad variabeln ska ha för typ, men i JavaScript, och många andra skriptspråk för den delen, så tar tolken hand om det. Så, när man deklarerar en variabel skriver man endast var, let eller const, följt av namnet på den, (och sedan ett semikolon när man är klar).

När man sedan vill ge variabeln ett värde (definiera den) skriver man dess namn följt av ett likamedstecken, =, och värdet man vill att den ska få. För att använda variabeln skriver man bara dess namn.

Funktioner

De kan vara **anonyma**, utan namn, eller använda ett namn. När man skapar/definierar en anonym funktion skriver man *function* fölt av två parenteser och sedan två måsvingar, curly brackets, {}. Innanför måsvingarna skriver man själva koden som ska utföras när man anropar funktionen. Innanför de vanliga parenteserna kan man definiera **parametrar** som funktionen tar emot. Mellan function och de vanliga parenteserna så kan man skriva ett namn, om man inte vill att funktionen ska vara anonym. Man kan också låta en variabel vara en referens till en funktion. Då skriver man helt enkelt namnet på variabeln, likamedstecknet, och den anonyma funktionsdefinitionen. Funktioner kan också svara när de blir kallade. De kan returnera ett värde. Då skriver man endast *return* följt av vad man vill svara med (och ett semikolon) någonstans i funktionen, vanligtvis som sista instruktion eftersom alla instruktioner efter *return* inte exekveras (körs), tolken återgår ju då tillbaka till där funktionen anropades ifrån, och stoppar in värdet som returnerades.

If-satser

Detta är en utav de viktigaste komponenterna inom programmering. Att programmet kan göra olika saker beroende på omständigheterna. Man ställer ett villkor (condition). Om det är sant så utförs ett kodblock eller en instruktion. Annars fortsätter programmet vidare, om man inte har lagt dit en else if-sats (annars om), eller en else-sats (annars). Villkoret brukar undersöka

en eller flera variablers värden mot angivna kriterier. Man kan också lista flera villkor och sätta logiska operatorer i mellan. Ett villkor kan antingen vara sant eller falskt, en logisk operator tar två villkor och, beroende på om båda eller minst en är sann svarar med sant eller falskt.

Listor

Inte att underskatta. Det som vi gör med programmering är ju i grund och botten att hantera data. Sortera data, analysera det, lägga till och ta bort. Vi har nu våra variabler, men undertiden som koden körs så är det obekvämt att låta koden skriva om sig själv för att deklarera fler variabler, även om det går med javascript. Man brukar vilja ha en så generell kod som möjligt, som är flexibel och lätt att ändra. Då är listor fantastiska.

Listor, eller arrays på engelska, kan vi definiera på två sätt. Antingen genom att göra en ny instans av array klassen, *new Array()*, eller bara använda förkortningen, *[]*. Innanför parenteserna kan man då rada upp värdena, separerade med komma, som man vill att listan från början ska innehålla. Alla listor har en egenskap, *length*, som är en integer som anger hur många artiklar listan innehåller. Som beskrivet ovan så har alla artiklar ett index som även det är en integer, och som börjar på noll

Loopar

Det finns ett antal olika typer av loopar. Gemensamt har dem att de repeterar ett kodblock. För att de inte ska göra samma sak om och om igen i all oändlighet behöver man en kontrollmekanism som kan säga stopp. Denna del brukar kallas för ett villkor. Det vanligaste sättet man brukar repetera saker på är "Så länge som ... är sant, gör ...", eller på engelska, "while ... is true, do ...", eller i JavaScript (**while-loopen**)

Eftersom det är kul att räkna upp saker, och för att mattematiken (som på sätt och vis ligger till grund för programmering) gillar att säga "för alla x, då x ingår i talmängden heltal, mindre än 10, gäller att ..." så har man även skapat **for-loopen**. Där definierar vi först en variabel, oftast en räknare. Inom mattematiken brukar vi döpa dem till i eller j, så samma här. Sedan följer ett villkor. Om det är sant så körs kodblocket en gång, sedan körs den sista instruktionen man bör ge for-loopen, vilket oftast används för att öka på räknaren. Därefter kollas villkoret en gång till. Repetera.

För att hoppa ur en loop på ett annat ställe än vid villkoret, så kan man anropa *break;* vart som helst i kodblocket, och då kommer tolken hoppa direkt till första instruktionen efter loopen.

Objekt

Listor är bra när allting inuti är lika på något sätt och de har en viss inbördes ordning. Men om man inte bryr sig om ordningen, och de är mer olika och specifika än lika, men fortfarande tillhör samma sak, då passar objekt bättre. Objekt har egenskaper (properties), och dessa egenskaper har värden (values). För att hitta till en viss egenskap använder man sig av en nyckel (key), som är unik för egenskapen i objektet. Bara en egenskap kan ha just den nyckeln, men flera egenskaper kan ha samma värde. Nyckeln är en sträng. På samma sätt som

när vi definierade en lista så går det även att definiera ett block på två olika sätt. Antingen genom att skapa en ny instans av objekt klassen, och därefter lägga till värden, eller via förkortningen med måsvingar (denna gång markerar de inte ett kodblock)

Faktum är att vi kan loopa igenom alla egenskaper i ett objekt, även fast de inte har något index. Detta gör vi med en **for..in loop**. Då tar loopen två argument, en variabel som kommer att bli varje nyckel som finns i objektet, och själva objektet. Men innan man går vidare och använder nyckeln måste man kontrollera att det är objektets egna nyckel och inte något som webbläsaren har lagt dit för skojs skull (rättare sagt, för att få saker att fungera)