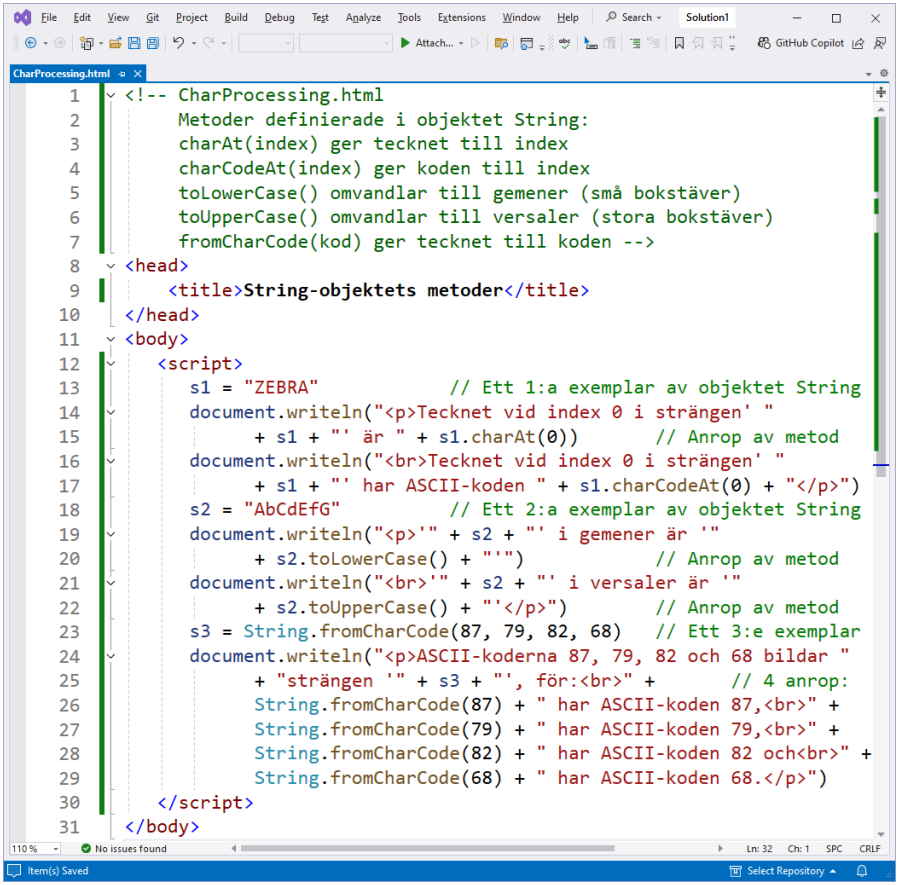
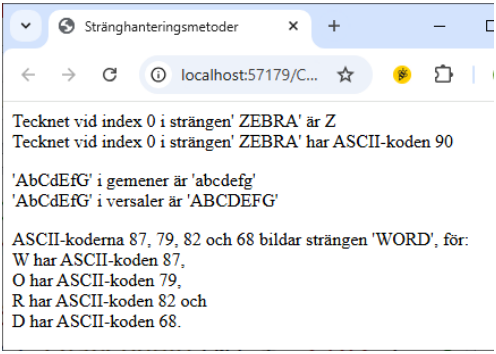
6.1 Stränghantering med String-objekt



Objektet `String`

`String` är ett *fördefinierat* objekt i JavaScript. I scriptet **`CharProcessing`** (sid 119) skapar vi några exemplar av objektet **`String`** och anropar metoder som är definierade i dem. På rad **11** skapas ett första exemplar:

```
s1 = "ZEBRA"
```



Detta verkar vara en vanlig deklaration och initiering av variabeln **`s1`**. Det stämmer också – med tillägget att variabeln är ett exemplar av *objektet* **`String`**, därför att citationstecknen **`" "`** gör

ZEBRA till en **String**. Och **String** är ett objekt som är definierat i JavaScript. Vi använder det bara i scriptet **CharProcessing** för att dra nytta av dess metoder. Det gör vi för första gången på rad **13**:

```
s1.charAt(0)
```

Dvs vi anropar metoden **charAt()** och skickar parametern **0**, för att få den första bokstaven (index **0**) till variabeln **s1** som i satsen innan initierats till strängen **ZEBRA** (rad **11**). Därför returnerar **charAt()** den första bokstaven **Z** som skrivs ut sedan, se körresultatet på sid 119.

Samma sak händer med de övriga anropen av metoderna **charCodeAt()** på rad **15**, **toLowerCase()** på rad **18**, **toUpperCase()** på rad **20** och **fromCharCode()** på raderna **24-27**. Vad de gör framgår av koden samt kommentarerna, se scriptet **CharProcessing** samt körresultatet på sid 119.

Objektbaserad programmering

JavaScript är ett *objektbaserad programmeringsspråk.* Vad betyder det? Man kan säga att det är en Light-variant av de objekt*orienterade* programmeringsspråken (OOP), så som vi känner till det från C++ eller från andra objektorieneterade språk. Anledningen är att JavaScript är ett interpreterande (inte kompilerande) språk som används på webben. Exekveringsmiljön är webbläsaren som måste ge respons i realtid.

En given definition på programmering är problemlösning med hjälp av datorn. Om man då beskriver problemets lösning i form av en *algoritm* kan man kort säga:

Program = algoritm + data

Denna definition ställdes upp av Niklaus Wirth på 60-talet och återspeglar den procedurala synen på programmering. Fokuset ligger på *algoritmen* dvs att inte bara hitta utan även *beskriva* tillvägagångssättet (proceduren) för att lösa ett problem. Sedan återstår bara att koda denna beskrivning. En annan definition som kom upp på 80-talet och återspeglar den objektbaserade synen på programmering är:

Program = Modell av verkligheten

Om man i Wirths formel *Program = algoritm + data* lägger vikten på data istället för på algoritmen och inte längre betraktar data som ett slags bihang till algoritmen utan som *objekt*, kommer man till *objektbaserad programmering*.

Vad är ett objekt i JavaScript?

Objektbaserad programmering syftar åt att efterlikna verkligheten. Man vill avbilda den reala världen – åtminstone den del som tillåter datorisering – och konstruera en modell av den i sina datorprogram för att kunna simulera verkligheten genom att tes-

ta modellen. För att undvika filosofiska diskussioner kan vi anta att den reala världen består kort sagt av *objekt*. Världen kring oss är full med sådana objekt: Människor, byggnader, bilar, tåg, flygplan, träd, möbler, böcker, butiker, skolor, bibliotek, kontor, anställda, kunder, varor, fakturor, order, bokningar, kurser osv. Objekten kan vara verkliga eller virtuella. Ett datorprogram försöker att beskriva dessa objekt.

Ett *objekt*, t.ex. en bil, har vissa egenskaper. Man kan t.o.m. säga att bilen är summan av alla sina egenskaper. Ett annat ord för egenskap är *attribut*. Summan av alla attribut utgör objektet. Bilen har som attribut: fabrikat, modell, färg, årsmodell, antal körda mil, antal hästkrafter, maximala hastigheten, antal och storlek på cylindrar i motorn osv. Alla dessa data utgör objektet bil och ger svar på frågan ”Vad är det för bil?”. Alla bilar har sådana attribut. Därför abstraherar man – dvs bortser från bilarnas olikheter – och samlar bilarnas gemensamma egenskaper (attribut) i något som man kallar för *objektet Bil*. När man programmerar, deklarerar man objektet *Bil* och skriver upp alla dessa bilattribut som objektets *egenskaper*. Vi sammanfattar:

> I JavaScript är objekt behållare för logiskt sammanhängande variabler och funktioner. Variablerna bildar objektets *attribut* (*egenskaper*) och funktionerna objektets *metoder*. Ett JavaScript-objekt organiserar data, kapslar in dem och ger ett gränssnitt till sina medlemmar (både attribut och metoder) via sitt namn.

Metoder

Bilden vore ofullständig om vi nöjde oss med objektets intressanta, men statiska egenskaper (attribut). Vi vill också veta vad man kan *göra* med dem. Ett objekt med alla sina attribut kan i regel även utföra vissa aktioner eller operationer. I den objektbaserade programmeringens terminologi kallas dessa aktioner för *metoder*. Typiska metoder för t.ex. en bil är att köra fram, att backa, att accelerera, att bromsa, att parkera, att byta olja osv. Den fullständiga definitionen på en bil vore alltså att ange *både* dess attribut *och* metoder. Bilfabrikanten måste förse bilen med alla dessa färdigheter för att kunna sälja den som en bil. Därför går man i bilfabriken efter en plan när man tillverkar bilen. Denna plan för konstruktion av bilen är *objektet Bil*. Konstruktörerna, mest ingenjörer, måste skapa denna plan, innan bilen kan byggas. När vi skriver ett program måste vi först definiera *objektet Bil* för att sedan kunna anropa objektets metoder. I definitionen måste vi ta upp alla attribut och metoder som är relevanta eller av någon anledning önskvärda för en bil.

En *metod* är en funktionalitet som definieras i ett objekt. Den talar om vad ett objekt kan *göra*. Det finns två steg i hantering av metoder: Först definierar man dem dvs skapar man deras kod i ett objekt. Sedan *anropar* dvs aktiverar man dem i exemplar av detta objekt. Ofta är det första steget redan genomfört av andra, så vi behöver bara anropa en redan *fördefinierad* metod. I objektet *Bil* t.ex. är metoderna att köra fram, att backa, att accelerera, att bromsa osv. definierade i huvuden på

bilkonstruktörerna och i deras konstruktionsritningar och dokumentationer. Sedan har man tillverkat massor med exemplar av objektet Bil i fabriken och byggt in dessa metoder i alla bilar. Vi behöver bara anropa dem i den bil vi kör. Den bil vi kör är ett specifikt exemplar av objektet Bil. Låt oss kalla det för **`minVolvo`**. Exemplaret **`minVolvo`** har ett antal attribut som t.ex. fabrikat, modell, färg, årsmodell osv., men också ett antal metoder, bl.a. metoden **`kör()`**. Parenteserna i metodens namn brukar man skriva för att karakterisera **`kör()`** som en *metod*. Man skriver ett anrop av metoden **`kör()`** så här:

```
minVolvo.kör()
```

Observera att *före* punkten står ett exemplar av objektet. Det är ju den specifika bil som jag använder just nu som ska köras. Först *efter* punkten står själva anropet av metoden **`kör()`**. Det här sättet att skriva kallas för *punktnotation*. Metoder måste alltid anropas med punktnotation, vilket har sin grund i att de endast är deklarerade i objektet. Till skillnad från fristående *funktioner* kan metoder varken definieras eller anropas utanför objekt. I JavaScript finns både metoder och funktioner.

En annan variant av metoden **`kör()`** kan anropas på följande sätt:

```
minVolvo.kör(40)
```

Det kan t.ex. betyda: Kör bilen med hastigheten **`40`** km/h. Värdet **`40`** kallas då en *parameter* som skickas till metoden när den anropas. I så fall måste även metoden **`kör()`** vara definierad så att den har beredskapen att ta emot denna parameter. Så det kan inte vara samma metod som anropades *utan* parameter. Det måste vara en annan variant av den, exakt talat en annan metod med samma namn. Konceptet kallas *överlagring av metoder* och innebär två eller flera metoder med samma namn, men olika parametrar.

I slutet vill vi dra parallellen till det vi lärt oss tidigare, nämligen *funktioner*. Vad är egentligen skillnaden mellan metoder och funktioner? Metoder är inget annat än funktioner vars definition placerats i ett objekt. Därför kan vi anropa metoder endast i exemplar av sådana objekt som innehåller metodens definition. Vi ska nu titta på några exempel på anrop av metoder som är fördefinierade i JavaScript-objektet **`String`** och som vi använt i vårt script **`CharProcessing`**.