

JS

Simple JavaScript

Book Version 1.0

Emre Can ÖZTAŞ

Kapak Tasarımı: Samet KÖROĞLU

Simple JavaScript

Book Version: 1.0

Yazan: Emre Can ÖZTAŞ

Önsöz

Merhaba, ben Emre Can ÖZTAŞ. Bu kitabın yazarıyım. Gazi Üniversitesi – Bilgisayar Mühendisliği Bölümü'nde öğrenciyim. İlgi alanım; Java ve Java Web Teknolojileri. Fakat bu kitapta nerede çıktı diyebilirsiniz. Bu kitap; fakültede üzerinde çalıştığım bir proje sayesinde doğdu. İşte bahsettiğim projede sıklıkla JavaScript ve Ajax gibi teknolojilerin kullanılma ihtiyacı olduğu için mecburen, daha önce hiç bulaşmadığım JavaScript'e doğrudan girmek zorunda kaldım. Pişman mıyım? Bilakis JavaScript öğrenirken ve dahası bu kitabı yazarken çok büyük zevk aldım. Umarım sizde okurken aynı zevki alırsınız.

JavaScript ile aşağı yukarı 5 ay gibi kısa bir süredir tanışıklığımız var. Bu 5 ay içerisinde öğrendiklerimi bir kitapta toplamak istedim. Hatta bu kitabı yazarken bir çok şey de öğrendim. Mesela; kitap yazmak çok zor bir iş imiş. Bu kitabı yazarken bölüm bölüm yazdım. Daha sonra bölümleri bir araya getirmek ve tek bir kitap halinde oluşturmak oldukça sıkıntılı bir iş oldu lakin buradan bir ders çıkarttım kendime.

Bu kitabı yazmamın 3 nedeni var. Bunlardan birincisi: Türkçe kaynağa destek olmak için, ikincisi: kitap alacak parası olmayan fakat öğrenmeye istekli arkadaşlarımızın faydalanması için, üçüncü neden: Allah nasip eder ilerleyen yaşlarımı görürsem geriye dönüp baktığımda neler yaptığımı hatırlamak için.

Kitap, sadece JavaScript ve temel olarak HTML üzerine kurulu. O yüzden kitabı takip edebilmek için temel seviyede HTML bilmeniz gerekmektedir. HTML5 biliyorsanız daha iyi tabiki.

Bu kitap tamamen ücretsizdir. İçinde bulunan bölümlerin veya kitabın tamamının çıktısı alınabilir, kopyalanabilir, dağıtılabilir ve herkesle paylaşılabilir. Sizden istediğim yegane şey; bu kitabın tamamını veya bir bölümünü, herhangi bir yerde kullanılıcaksanız kaynak göstererek kullanmanızdır.

Kitap hakkındaki görüş ve önerilerinizi, hatalı olan kodları veya yazım yanlışlarını lütfen bana bildirin. Mail adresim: emrecaoztas@outlook.com
Bir diğer konuda herhangi bir yerde takıldığınız zaman bana soru ile gelmenizi istemem. Ben sorularınıza içtenlikle cevap vermek, yardım etmek isterim fakat sizin takıldığınız yerde belki başka bir arkadaşımızda takılmış olabilir. O yüzden; www.btsoru.com gibi herkese açık sosyal bir platformda, bu takıldığınız sorularınızı sorarsanız, aldığınız cevaplarla diğer arkadaşlarımıza da yardımcı olmuş olursunuz.

Ve son olarak umarım bu kitap, gelişiminizde temel bir taş olur. Bildiğim her şeyi anlatmaya ekstra olarak bilmediğim şeyleri de araştırarak kitaba koymaya özen gösterdim. Ayrıca bu kitap ne ilk ne de son kitap olacak. Bilgim dahilinde sahip olduğum diğer konularda da yazmaya ve bu yazdıklarımı kitap haline getirmeye çalışacağım.

Birilerine yardımcı olursam ne mutlu bana.

Teşekkürler

Bu kitabın yazımında bana olumlu / olumsuz destek olan çevremdeki herkese teşekkür ederim. Siz olmasaydınız; gelişimimi tamamlayamaz, bugünkü eriştiğim noktaya erişemezdim. İyi ki varsınız. Umarım hep var olursunuz.

Kapak tasarımını yapan Samet KÖROĞLU'na da özel bir teşekkür etmek isterim.

1	JavaScript	10
	1.1 JavaScript Nedir?	10
	1.2 Neden JavaScript?	11
2	JavaScript için Araçlar	12
3	İlk JavaScript Programı	15
	3.1 alert()	16
4	Variables ve Ekrana Yazdırma	18
	4.1 Değişken Tanımlama	18
	4.2 Ekrana Yazdırma	19
	4.3 Sabit Değişken Tanımlama	21
5	Operators ve Comment Lines	23
	5.1 Operators	23
	5.1.1 Simple Assignment Operators	23
	5.1.2 Arithmetic Operators	23
	5.1.3 Unary Operators	26
	5.1.4 Equality And Relational Operators	29
	5.1.5 Conditional Operators	32
	5.1.6 Bitwise And Bitshift Operators	35
	5.2 Comment Lines	37
6	prompt ve confirm	39
	6.1 prompt	39
	6.2 confirm	42
7	Data Types and Type Casting	45
	7.1 Data Types	45
	7.2 Type Casting	46
	7.2.1 parseInt	47
	7.2.2 parseFloat	48
	7.2.3 toString	49
8	Control Statement	50
	8.1 if – else	50
	8.2 swicth...case	53
9	Loops ve continue & break Deyimleri	57
	9.1 for	57
	9.2 while	60
	9.3 do...while	62
	9.4 continue & break Deyimleri	63

9.4.1 break	63
9.4.2 continue	65
10 Arrays ve for...in Yapısı	67
10.1 Arrays	67
10.1.1 reverse()	70
10.1.2 sort()	70
10.1.3 shift()	71
10.1.4 slice()	72
10.1.5 hasOwnProperty()	73
10.1.6 push()	73
10.1.7 pop()	74
10.2 for...in Yapısı	75
11 Functions	76
11.1 return	82
12 Events	87
12.1 onload	87
12.2 onclick	88
12.3 ondblclick	90
12.4 onmouseover	91
12.5 onmouseout	92
12.6 Other Events	92
13 HTML DOM	95
13.1 Elementlere Erişmek	95
13.1.1 getElementById()	95
13.1.2 getElementsByTagName()	102
13.2 Elementlerin Değerlerini Değiştirmek	104
13.3 Elementleri CSS ile Şekillendirmek	114
13.4 Extras	117
13.4.1 domain	117
13.4.2 lastModified	118
13.4.3 location	118
13.4.4 title	118
13.4.5 writeln()	119
14 Browser	120
14.1 navigator	120
14.1.1 appcodeName	120
14.1.2 appVersion	121
14.1.3 language	122
14.1.4 platform[]	123

14.1.5	userAgent	123
14.2	window	124
14.2.1	open()	124
14.2.2	close()	127
14.2.3	print()	128
14.2.4	location	129
14.3	screen	131
14.3.1	width	131
14.3.2	height	131
14.3.3	colorDepth	132
14.3.4	availWidth	132
14.3.5	availHeight	133
15	String	135
15.1	charAt()	135
15.2	charCodeAt()	136
15.3	concat()	136
15.4	indexOf()	137
15.5	lastIndexOf()	139
15.6	slice()	140
15.7	split()	141
15.8	ubstr()	142
15.9	toLowerCase()	143
15.10	toUpperCase()	144
16	Math	146
16.1	e	146
16.2	LN2	147
16.3	LN10	147
16.4	LOG2E	148
16.5	LOG10E	149
16.6	PI	149
16.7	abs()	150
16.8	acos()	151
16.9	asin()	152
16.10	atan()	153
16.11	ceil()	154
16.12	cos()	155
16.13	floor()	156
16.14	log()	157
16.15	max()	158

16.16 min()	159
16.17 pow()	160
16.18 random()	161
16.19 round()	162
16.20 sin()	163
16.21 sqrt()	164
16.22 atan()	165
17 Date	167
17.1 Local: Tarih ve Saat	167
17.1.1 getDate()	167
17.1.2 getDay()	168
17.1.3 getMonth()	169
17.1.4 getFullYear()	171
17.1.5 getMilliseconds	171
17.1.6 getSeconds()	172
17.1.7 getMinutes()	173
17.1.8 getHours()	173
17.1.9 getTime()	174
17.1.10 getTimezoneOffset()	175
17.1.11 setDate()	176
17.1.12 setMonth()	177
17.1.13 setFullYear()	178
17.1.14 setMilliseconds()	179
17.1.15 setSeconds()	180
17.1.16 setMinutes()	181
17.1.17 setHours()	182
17.2 Global: Tarih ve Saat	183
17.2.1 getUTCDate()	183
17.2.2 getUTCDay()	183
17.2.3 getUTCMonth()	184
17.2.4 getUTCFullYear()	185
17.2.5 getUTCMilliseconds()	185
17.2.6 getUTCSeconds()	186
17.2.7 getUTCMinutes()	186
17.2.8 getUTCHours()	187
17.2.9 setUTCDate()	187
17.2.10 setUTCMonth()	188
17.2.11 setUTCFullYear()	189
17.2.12 setUTCMilliseconds()	189

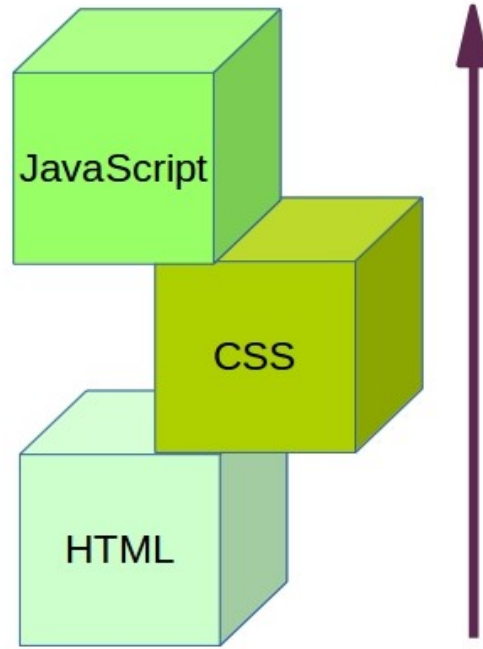
17.2.13	setUTCSeconds()	190
17.2.14	setUTCMinutes()	190
17.2.15	setUTCHours()	191
17.3	Extras	192
17.3.1	toDateString()	192
17.3.2	toTimeString()	193
17.3.3	toLocaleDateString()	193
17.3.4	toLocaleTimeString()	194
17.3.5	toLocaleFormat()	195
18	Errors and Exception	197
18.1	try...catch	198
18.1.1	finally	198
18.1.2	throw	199
19	External JavaScript Files	201
20	JSON	205
20.1	JSON Nedir?	205
20.2	JSON Kullanımı	205
21	JavaScript Code Compress	209
	 KAYNAKLAR	 214

1.1 JavaScript Nedir?

JavaScript, bir programlama dili değildir. Oracle Java ile de hiç bir alakası yoktur. Başta bu iki konu üzerinde anlaşalım. JavaScript, Netscape firması çalışanlarından, Brendan Eich tarafından 1996 yılında, tamamen web ortamı için geliştirilmiş script bir dildir. JavaScript'in ticari ismi ECMAScript'tir. ECMA-262'de JavaScript'in ticari standartlarıdır.

JavaScript, HTML (Hyper Text Markup Language) sayfaları ile birlikte kullanılmaktadır. Kullanımı zorunlu olmayıp fakat kullanıldığında çok daha profesyonel ve özelleşmiş interaktif web sayfaları elde etmek mümkündür.

HTML; web sayfalarının iskeletini oluştururken, CSS bu web sayfalarına sunum özelliği kadar. Sunumdan kasıt; HTML sayfalarının görselliği ve her sayfanın diğer kardeş sayfalarla görsellik bakımından tutarlılığıdır. JavaScript ise web sayfalarına davranış özelliği katar.



Davranıştan hareketle; JavaScript, web sayfaları için programlama özelliğini sağlar. HTML veya CSS kullanarak oluşturulan bir web sayfası; kullanıcıya herhangi bir soru karşısında yanıt veya bir tepki veremez. Örneğin; kullanıcıların üye olarak girebildiği bir sitede, JavaScript bu üye kayıt formuna ait alanların boşluğunu / doluluğunu veya belli kriterlere göre değerlendirmeler yapıp formun uygunluğunu kontrol edebilir. Bunun dışında sayfa geçişleri v.s gibi çok çeşitli özellikleri web sayfalarına kazandırabilir. O yüzden şuan JavaScript web dünyasının bel kemiğidir desek yanlış olmaz.

JavaScript, yorumlanması için Web Browser (Tarayıcı) ihtiyaç duyar. Bu web browser;

Internet Explorer (hala kullanan varsa), Chrome, Firefox Mozilla, Yandex v.s olabilir. Buradan hareketle JavaScript kullanıcının tarayıcısında yorumlanır ve çalıştırılır diyebiliriz. Daha da açıklayıcı olması açısından; hazırlamış olduğumuz JavaScript betikleri web sayfasına bağlanan kullanıcıların tarayıcılarında çalışır. Öte yandan her JavaScript kodu her tarayıcıda çalışacak diye bir durum yoktur. Her tarayıcı JavaScript'i tam anlamıyla desteklemez. O yüzden bu durumu düşünerek daha önceden önlem alınmalıdır. İlerleyen bölümlerde detaylı olarak bu konu üzerine duracağız.

JavaScript, hem HTML sayfalarında gömülü olarak kullanılabileceği gibi hem de ayrı bir dosya şeklinde oluşturulup kullanılabilir. Seçim tamamen bizlere bırakılmış durumda. Fakat önerilen; HTML sayfalarında gömülü olarak kullanılmasıdır.

JavaScript, web sayfaları için programlama desteği sağlar dedik. Burada belirtmek istediğim ayrı bir konu var. JavaScript, web sayfalarında bir çok alan için programlama desteği getirebilir fakat JavaScript herhangi bir veri tabanına bağlanıp CRUD (Create, Run, Update, Delete) gibi işlemleri gerçekleştiremez. Veri tabanına bağlanma işi; web programlama için ayrı olarak geliştirilmiş çeşitli programlama dillerinin işidir.

1.2 Neden JavaScript?

Web Browser'lar (Tarayıcılar), sadece HTML + CSS + JavaScript bilirler. Bunların dışında herhangi bir programlama dili veya araç bilmezler. Bu süper üçlü web programlamanın temelini oluştururlar. O yüzden web programlamaya adım atacak birisinin bu süper üçlü hakkında bilgi sahibi olması gerekir. front-end kısmında veya back-end kısmında çalışın farketmez. Bu durum herkes için geçerlidir. Özellikle JavaScript konusunda diyeceğim şey; ister masaüstü programcılıkla uğraşın ister oyun programlama veya mobil uygulama geliştirin. Emin olun JavaScript bir gün karşınıza çıkacaktır. O yüzden bence öğrenmek bir seçenek değil ihtiyaçtır.

2008 yılında, HTML 5'in çıkışıyla birlikte; JavaScript'in de yükselişi başlamış oldu. Daha önceden kullanılmıyordu mu? Evet, kullanılıyordu fakat HTML 5'in getirdiği yeniliklerle JavaScript'in önemi bir kez daha artmıştır. www.tiobe.com adresinde, dünyada en çok kullanılan programlama dillerinin popülerlik oranlarına bakabilirsiniz. Şuan JavaScript bu sıralamada ikinci sıradadır (2015).

Android programlama şuan çok revaşa ve etrafımda gördüğüm herkes birden mantar gibi Android Developer ilan etti kendini. Bildiğiniz gibi Android'in yıldızını parlatan herhalde Samsung ile kurduğu ortaklıktır. Samsung telefonlarının satışlarını da Android yükseltmiş ve kendisine karşı çıkan Nokia yerle bir olmuş, enkazı da Windows devralmıştır. Samsung ve IBM şuan bir işletim sistemi üzerinde çalışıyorlar, bilmiyorum hiç duydunuz mu adını. Tizen. Evet, Fırsatınız olduğunda Tizen'i incelemeyerseniz incelemenizi tavsiye ederim. Tizen için programlama: HTML 5 + CSS ve JavaScript ile yapılıyor. Bunun dışında her platforma destek veren PhoneGap uygulamasıda yine aynı şekilde HTML 5 + CSS + JavaScript ile yapılıyor. Yani JavaScript'in önemi giderek artıyor.

Bunun dışında JavaScript için çeşitli Framework'ler geliştirilmeye başlandı. Bunlara örnek verecek olursak; EmberJS, NodeJS, JQuery, BackboneJS, MonkeyJS v.s daha bir çok var. Geliştirilen her JavaScript Framework'un özellikleri farklı. Hepsinin belli bir alana hitap ettiği kesin. Bunlardan en çok kullanılan JQuery. Tabiki hepsinin temeli de JavaScript.

Ve son olarak; JavaScript, kullanmak için herhangi bir ücret ödemek zorunda değilsiniz. Tamamen ücretsizdir. İsteyen herkes istediği şekilde ve istediği web sayfasında gönlünce JavaScript'i kullanabilir.

BÖLÜM 2:

JavaScript için Araçlar

JavaScript programlamak için herhangi bir araca, kurulumu veya IDE (Integrated Development Environment)'ye ihtiyaç yoktur. Bilgisayarınızda herhangi bir text editor programı ve Web Browser (Tarayıcı) var ise kolları sıvayıp hemen JavaScript geliştirmeye başlayabilirsiniz. Ama yine de iyi bir araç kullanmak mantıklı bir seçim olacaktır. Ben bütün işlerimde Eclipse'yi kullanırım fakat bu kitabın konusu Java değil o yüzden diğer gözde bir programım olan Geany'yi önermek isterim. Neden Geany sorusuna da cevap vermek isterim.

Geany hem GNU / Linux hemde Microsoft için sürümü bulunan ücretsiz bir programdır. Aynı zamanda birden fazla programlama diline hitap eden ve kod renklendirme özelliğine sahip bir text editor programıdır. Kod tamamlama özelliği maalesef bulunmamaktadır. Zaten bulunmaması da şuan bizim için bir avantajdır. Programlamaya, ya da yeni bir dile başlayanlara her zaman ilk tavsiyem; kendiniz yazın, deneyin, herhangi bir araç kullanmayın olmuştur. Çünkü böylelikle işin mutfağında daha çok bulunacaksınız ve arkaplanda ne oluyor ne bitiyor öğrenmiş olacaksınız. Öte yandan şuan piyasada en azından benim gördüğüm kadarıyla kaliteli bir JavaScript kod editor programı bulunmamaktadır. Web Storm dışında. O da zaten ücretli. Ücretsiz olarak belki Aptana veya Sublime Text bir nebze olabilir lakin onlardan da yüksek performans beklememek lazım. O yüzden Geany ile devam diyorum.

Microsoft kullanıcıları Geany'yi indirmek için aşağıdaki adrese girebilirler.

<http://www.geany.org/>. Bunun dışında son bir öneri olarak Microsoft kullanıcıları, Geany'yi kullanmak istemezlerse Notepad++ programını da kullanabilirler.

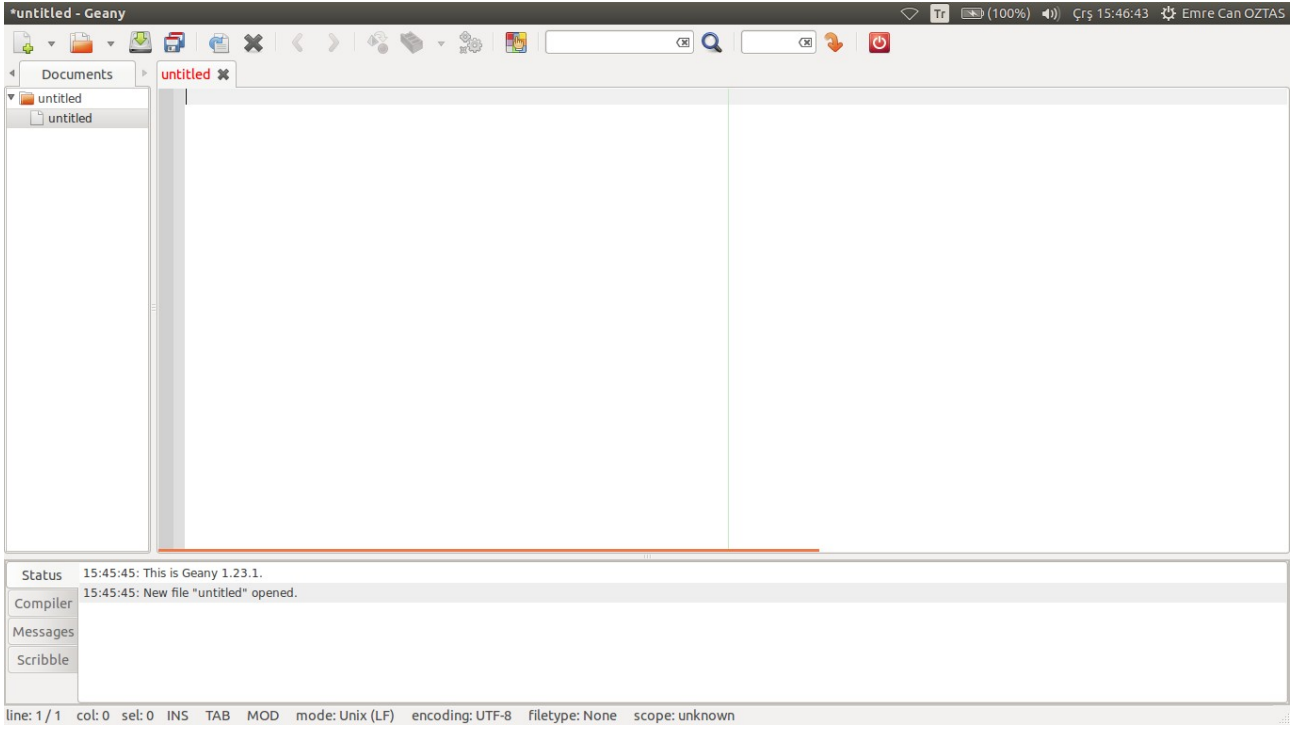
GNU / Linux kullanıcıları ise Geany'i komut ekranından kurulumunu yapabilirler.

```
root@acer: /home/oztas
root@acer:/home/oztas# sudo apt-get install geany_
```



Yukardaki kurulum Debian tabanlı GNU / Linux dağıtımları içindir. Bu kitap baştan sona kadar Ubuntu kullanılarak hazırlanmıştır.

Geany kurulumundan sonra çift tıklayarak programımızı açalım.



Yukarıdaki ekran alıntısında gördüğünüz gibi Geany ortamı açıldı. Burada bir kaç ayar yapalım ve diğer bölümde JavaScript kodlamaya başlayalım.

Aşağıdaki ayarları daha rahat bir kod geliştirme ortamı için uygulayalım.

Öncelikli olarak aşağıdaki Message penceresini kapatalım:

Menü >> View > Show Message Window yolunu takip edip tik işareti kaldıralım.

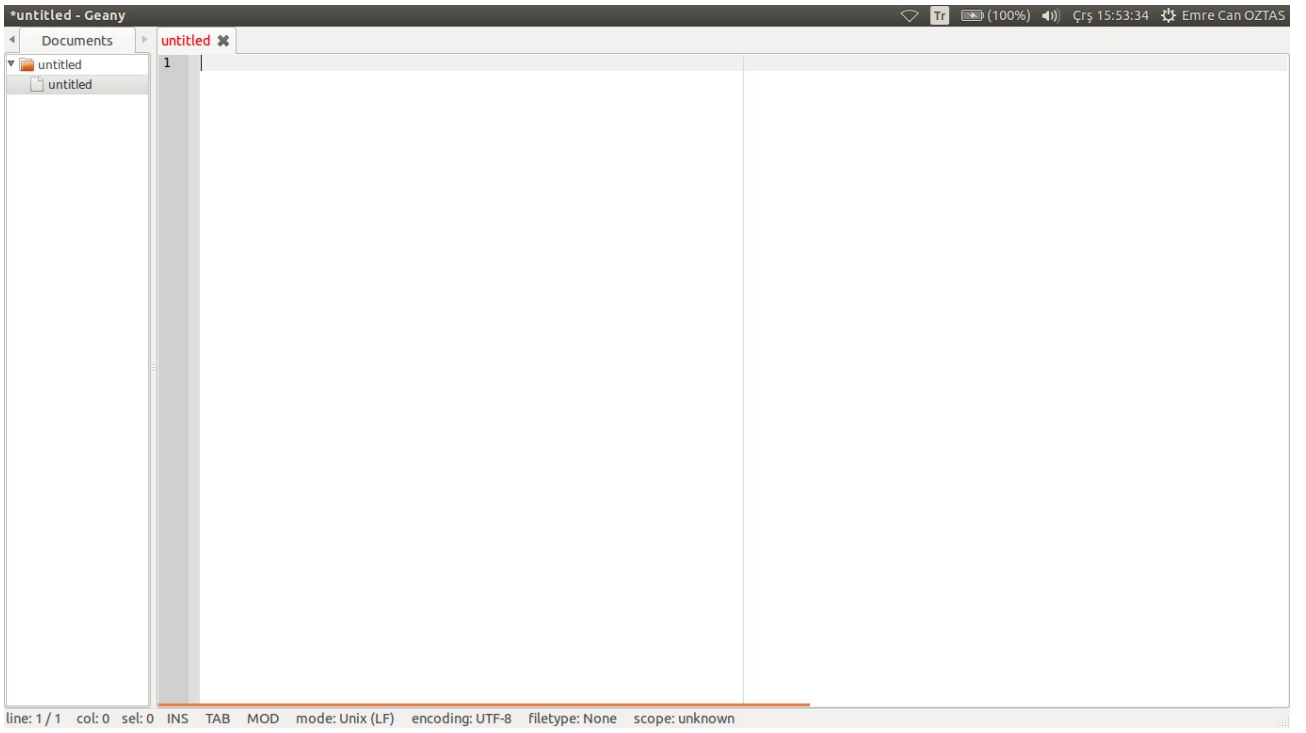
Araçlar çubuğunu kaldıralım:

Menü >> View > Show Toolbar yolunu takip edip tik işareti kaldıralım.

Satırları numaralandıralım:

Menü >> View > Editor > Show Line Number yolunu takip edip tik işareti koyalım.

Ayarlarımızı yaptık. Son olarak Geany aşağıdaki gibi görünecektir.



Geany ortamında Full Screen (Tam Ekran) çalışmak isterseniz F11 tuşunu kullanabilirsiniz.

Daha rahat bir geliştirme için Geany'i hazırlamış olduk. Artık JavaScript geliştirmeye başlayabiliriz.

BÖLÜM 3:

İlk JavaScript Programı

Bu bölümde ilk JavaScript programımızı yazacağız.

Bir önceki bölümde kurulumunu yaptığımız Geany programını açalım. Şimdilik JavaScript kodlarımızı HTML etiketleri arasına yazacağız. Daha sonra harici JavaScript dosyaları oluşturup, kullanacağımız web sayfasında çağıracağız.

Madem şimdilik HTML ile JavaScript kodlaması yapacağız; Geany ortamına HTML sayfasıyla çalışacağımızı bildirelim. Bunun için:

Menü >> Document > Set Filetype > Markup Languages > HTML source file yolunu takip edelim.



Geany ortamında pek çok farklı programlama dili ile çalışabilirsiniz. Yukarıdaki gibi herhangi bir programlama dili ile çalışmadan önce bunu Geany'e bildirmelisiniz. Aksi halde kod renklendirme özelliği çalışmayacaktır.

HTML etiketlerimizi yazalım.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
</head>
<body>
</body>
</html>
```

HTML etiketlerinin arasında, JavaScript kodlarımızı yazacağımızı söylemiştik. Burada özgürsünüz; **<HEAD></HEAD>** etiketleri arasında veya **<BODY></BODY>** etiketleri arasında JavaScript kodlarımızı yazabiliriz. Genelde fonksiyon içeren JavaScript kodlarının **<HEAD></HEAD>** arasına yazılması tavsiye edilir. Diğer durumlarda **<BODY></BODY>** arasında yazılması daha doğru olur.

JavaScript kodları, HTML etiketleri arasında kullanılacaksa aşağıdaki etiketler arasına yazılır. İlerleyen bölümlerde ayrı JavaScript dosyaları oluşturduğumuz zaman aşağıdaki etiketlere ihtiyacımız kalmayacaktır. Çünkü ayrı JavaScript dosyalarında bu etiketler kullanılmaz.

```
<script type="text/JavaScript">
</script>
```

Yukardaki `script` etiketleri arasında JavaScript kodlarımızı yazacağız. İlk programımız “Hello World” yazdırsın, gelenek bozulmasın.

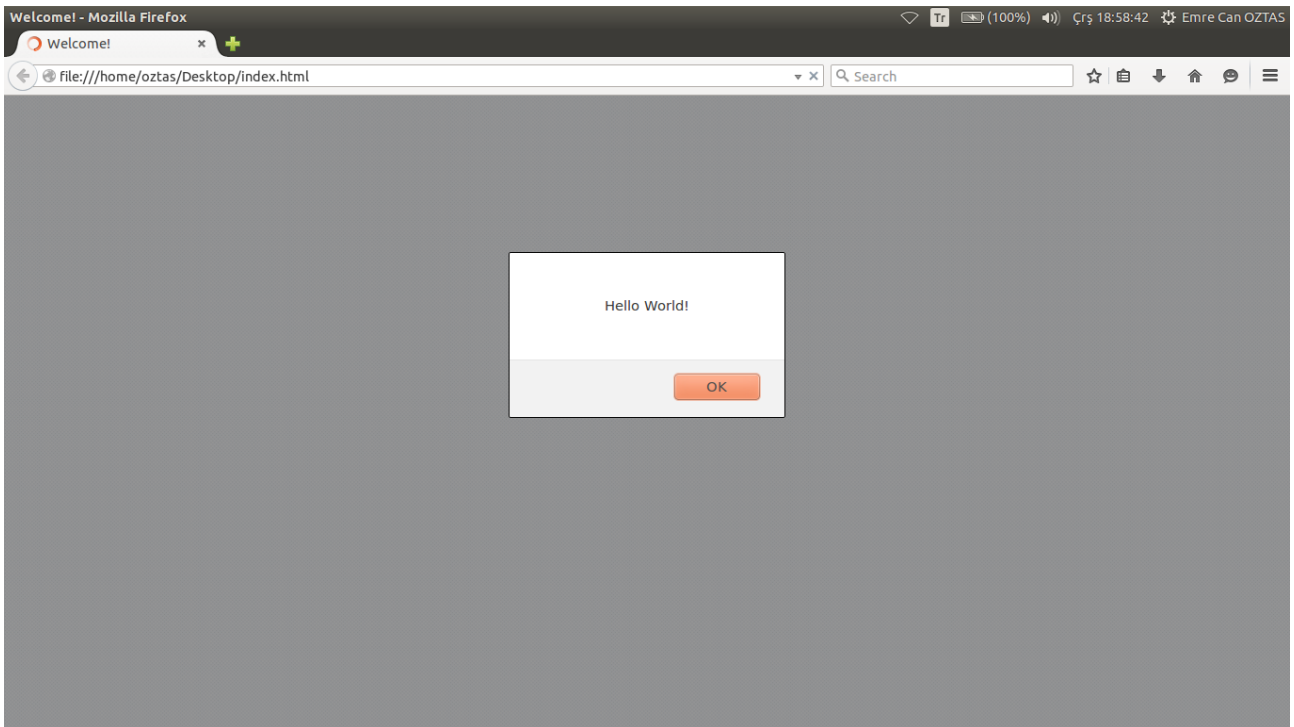
```
<!DOCTYPE html>
<html>
```

```
<head>
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    alert("Hello World!");
</script>
</body>
</html>
```



JavaScript kod satırları noktalı virgül (;) ile biter.

JavaScript kodlarımızı yazdık, her şey tamam. Şimdi HTML sayfamızı .htm veya .html uzantısıyla Desktop (Masaüstü)'a kaydedelim, erişimin kolay olması açısından. Ben index.html adıyla kaydettim. Kaydetmiş olduğumuz bu web sayfasını Browser (Tarayıcı)'da görüntüleyelim. Bunun için Geany ortamında; Menü >> Build > Execute yolunu takip ederek ya da F5 tuşuna basarak veya masaüstüne kaydettiğimiz HTML dosyasını çift tıklayarak çalıştırabiliriz.



Yukarıdaki açılan sayfamızı incelediğimizde; ekranın ortasında bir uyarı penceresiyle karşılaştık. Bu uyarı penceresini az önce biz oluşturduk. Bu da ilk öğrendiğimiz JavaScript komutudur.

3.1 alert()

JavaScript kodlarımızda kullandığımız alert() komutu uyarı vermek için kullanılan bir komuttur. alert() komutunu kullanarak başka uyarılar da verebiliriz ve birden fazla kez alert() komutunu kullanabiliriz. Örneğin alert kullanarak başka uyarılar da verdirelim.

```
<script type="text/JavaScript">
    alert("Dikkat!");
    alert("Uyarı!");
    alert("Kullanıcı girişi başarısız!");
</script>
```

Yukarıdaki kodlarımızı yine aynı şekilde HTML etiketleri arasında yazıp çalıştıralım. Sırasıyla:
Dikkat! Uyarı! Kullanıcı girişi başarısız! Şeklinde 3 ayrı uyarı penceresi art arda
sırasıyla karşımıza gelecektir.

Şu ana kadar yani ilk JavaScript kodlarımızda herhangi bir şey dikkatinizi çekti mi bilmiyorum. Bu dikkatinizi çekmesini istediğim şey; JavaScript kodlarını içeren HTML sayfaları, tarayıcıda görüntülendikleri anda JavaScript kodlarının çalışması. Evet, JavaScript kodları herhangi bir fonksiyon, blok v.s içermediği durumda web sayfası yüklendiği anda çalışır. Fakat örneğin bir fonksiyon kullandığımız zaman ve bu fonksiyonu çağırmadığımız sürece JavaScript kodları çalışmayacaktır. Buna da dikkat edelim.

BÖLÜM 4:

Variables ve Ekrana Yazdırma

Bu bölümde ayrıntılı olarak Variables (Değişkenler) konusuna değineceğiz. Bildiğiniz gibi değişkenler; verileri kısa süreli tutmak ve üzerinde işlemler yapmak için kullanılan herhangi bir hafıza bölümü veya günlük hayattan bir örnek olarak kap olarak düşünebiliriz.

JavaScript'te değişken tanımlarken belli kurallar vardır. Bu kurallara Identifiers (Belirleyiciler) denir. Bu kurallar nelermiş maddeler halinde görelim.

Identifiers (Belirleyiciler):

- JavaScript Case Sensitive (Büyük/Küçük harfe duyarlı) özelliğindedir. O yüzden tanımlanan değişkenlerin, tanımlandığı isimle kullanılması gerekir. Örneğin `degisken` ile `Degisken` farklı şeylerdir.
- Değişken tanımlaması yapılırken; değişken isimleri arasında boşluk bırakılamaz.
- Değişkenler isimlendirilirken birden fazla kelime kullanılacaksa kelimeler arasına `_` underscore (alt çizgi) konur. Fakat ben programlama hayatımda, temel olarak Java Code Convention'ı benimsediğim için iki veya daha fazla kelime kullanacaksam ikinci kelimenin baş harfini büyük yazarım. Örneğin: `toplamaIslemi`, `degiskenAdi`, `kullaniciAdi` v.s gibi. Bu kitapta da bu yöntemi takip ettim.
- Değişken isimleri rakamla başlayamaz. İlk karakterden sonra rakam ile devam edebilir. Aynı zamanda ilk karakter `_` underscore (alt çizgi) işareti olabilir.
- JavaScript için Reserved Words (Ayrılmış Kelimeler) vardır. Bu kelimeler herhangi bir değişken veya fonksiyon tanımlamasında kullanılamaz.

Reserved Words (Ayrılmış Kelimeler):

`abstract`, `alert`, `arguments`, `Array`, `blur`, `boolean`, `Boolean`, `break`, `byte`, `callee`, `caller`, `captureEvents`, `case`, `catch`, `char`, `class`, `clearInterval`, `clearTimeout`, `close`, `closed`, `confirm`, `const`, `constructor`, `continue`, `Date`, `debugger`, `default`, `defaultStatus`, `delete`, `do`, `document`, `double`, `else`, `enum`, `escape`, `eval`, `export`, `extends`, `false`, `final`, `finally`, `find`, `float`, `focus`, `for`, `frames`, `Function`, `function`, `goto`, `history`, `home`, `if`, `implements`, `import`, `in`, `infinity`, `innerHeight`, `innerWidth`, `instanceof`, `int`, `interface`, `isFinite`, `isNaN`, `java`, `length`, `location`, `locationbar`, `long`, `Math`, `menubar`, `moveBy`, `moveTo`, `name`, `NaN`, `native`, `netscape`, `new`, `null`, `Number`, `Object`, `open`, `opener`, `outerHeight`, `outerWidth`, `package`, `Packages`, `pageXOffset`, `pageYOffset`, `parent`, `parseFloat`, `parseInt`, `personalbar`, `print`, `private`, `prompt`, `protected`, `prototype`, `public`, `RegExp`, `releaseEvents`, `resizeBy`, `resizeTo`, `return`, `routeEvent`, `scroll`, `scrollbars`, `scrollBy`, `scrollTo`, `self`, `setInterval`, `setTimeout`, `short`, `static`, `status`, `statusbar`, `stop`, `String`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `toolbar`, `top`, `toString`, `transient`, `true`, `with`, `typeof`, `watch`, `unwatch`, `valueOf`, `try`, `void`, `unescape`, `while`, `window`, `var`

4.1 Değişken Tanımlama

JavaScript ortamında değişken tanımlamak için `var` anahtar kelimesi kullanılır. Bunun dışında Java veya C gibi dillerde olduğu gibi herhangi bir `data type` (veri tipi) yazılmaz.

Değişken tanımlaması aşağıdaki gibi gerçekleştirilir.

```
var degiskenAdi;
```

Değişken tanımlamasından sonra bu değişkene ilk değer ataması yapılabilir.

```
var degiskenAdi = ilkDeger;
```

Aynı satıra birden fazla değişken tanımlaması da yapılabilir.

```
var degiskenAdi1, degiskenAdi2, ... , degiskenAdiN;
```

Tanımlanan bu değişkenlere ilk değerleri de atanabilir.

```
var degiskenAdi1 = ilkDeger1, degiskenAdi2 = ilkDeger2, degiskenAdi3 =  
ilkDeger3, degiskenAdi4 = ilkDeger4;
```



JavaScript ortamında; aynı satırda veya farklı satırda değişken tanımlaması yapılabilir. Bu aşağı yukarı bütün programlama dillerinde vardır. Değişken tanımlarken her zaman tercihiniz; bir satırda yalnız bir değişken tanımlamak olsun. Böylelikle daha estetik, temiz ve anlaşılır kod yazmış olursunuz. Aynı zamanda bu yazım performansa etki edebilir.

Örnek olarak kendimiz değişkenler tanımlayalım.

```
var nameSurname = "emrecan-oztas";  
var country = "Turkey";  
var whereFrom = "Konya";  
var PI_SAYISI = 3.14;  
var year = 2015;
```

Yukarıda görüldüğü gibi çeşitli tipte değişkenlerimizi tanımladık. Burada size başka bir şey göstermek istiyorum. Yukarıdaki değişkenlerimizi aynı satırda tanımlayalım ve ilk değerlerini yine aynı şekilde verelim. Aşağıda olduğu gibi.

```
var nameSurname = "emrecan-oztas", country = "Turkey", whereFrom = "Konya",  
PI_SAYISI = 3.14, year = 2015;
```

JavaScript ortamında değişken tanımlarken; başta herhangi bir veri tipi belirtmediğimiz için aynı satırda tanımladığımız birden fazla değişkene çeşitli tipte değerler atayabiliriz.

Bu değişkenler genel / geçer değerlerdir. Yani istediğin zaman yaz sonra sil mekanizmasında kullanılırlar. JavaScript ortamında sabit değişken tanımlaması yaparak bu değişkenlerin değerlerini sonradan değiştirilmeye karşı koruyabiliriz. Sabit değişken tanımlamalarına geçmeden önce; bu değişkenlerimizi ekrana yazdıralım. Sanırım alert() komutu bu durumda işimizi görmeyecektir. Bunun yerine farklı bir komut kullanalım: `document.write()`.

4.2 Ekranaya Yazdırma

Ekranaya yazdırma işlemi için `document.write()` komutunu kullanacağız. Bu komutun standart kullanımı aşağıdaki gibidir.

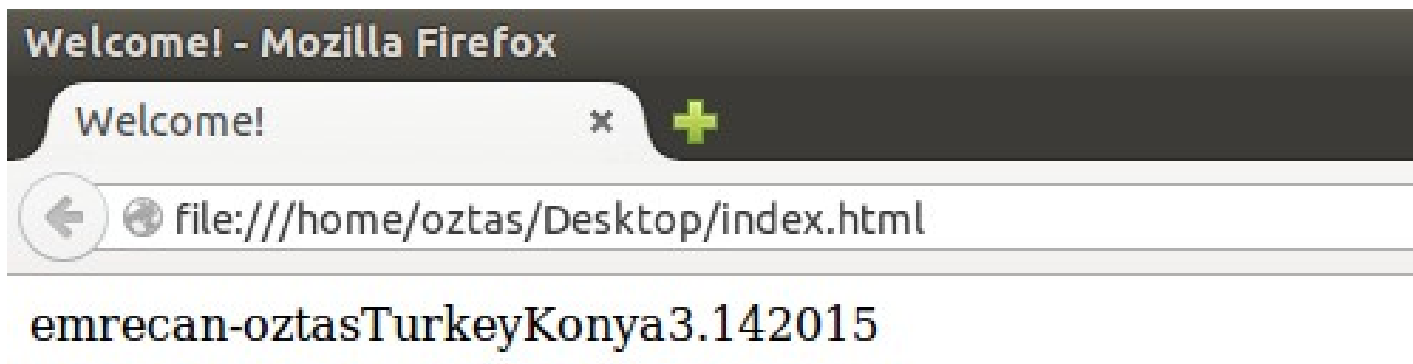
```
document.write("yazdirilacak_degerler");
```

Tanımladığımız değişkenleri document.write() komutunu kullanarak ekrana yazdıralım.

HTML sayfamızın son şekli aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var nameSurname = "emrecan-oztas";
    var country = "Turkey";
    var whereFrom = "Konya";
    var PI_SAYISI = 3.14;
    var year = 2015;
    document.write(nameSurname);
    document.write(country);
    document.write(whereFrom);
    document.write(PI_SAYISI);
    document.write(year);
</script>
</body>
</html>
```

Yazmış olduğumuz kodlarımız için ekran çıktısı alalım.



Ekran alıntısında görüldüğü gibi tanımlamış olduğumuz değişkenleri sırasıyla ekrana yazdırdık. Satır atlama veya herhangi bir boşluk bırakma yapmadığımız için sırasıyla ekrana yazdırıldı.

JavaScript kodları içerisinde HTML etiketleri de kullanılabilir. Bildiğiniz gibi
 etiketi satır atlama ve ise boşluk bırakma görevindedir. Boşluk bırakma işlemini içi boş turnaklarla (" ") da gerçekleştirebiliriz. document.write() ifadesini kullanarak değişkenlerimizi alt alta yazdırabiliriz. Şöyle ki:

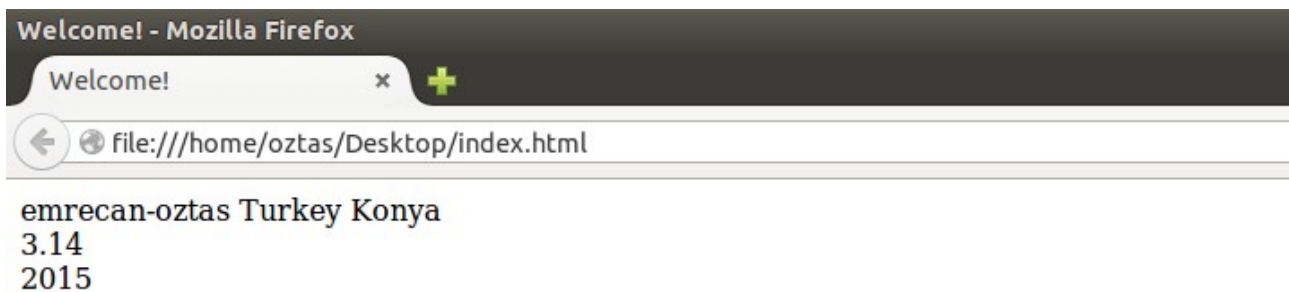

```
document.write(nameSurname + "&nbsp;" + country + " " + whereFrom + "<br>" +  
PI_SAYISI + "<BR>" + year);
```

veya

```
document.write(nameSurname);  
document.write("&nbsp;");  
document.write(country);  
document.write(" ");  
document.write(whereFrom);  
document.write("<BR>");  
document.write(PI_SAYISI);  
document.write("<BR>");  
document.write(year);
```

Şeklinde yazabiliriz.

Ekran çıktısı alalım.



Yukarıdaki ekran alıntısında görüldüğü üzere; yazmış olduğuz HTML etiketleriyle istediğimiz şekilde bir yapı oluşturabiliyoruz.

4.3 Sabit Değişken Tanımlama

JavaScript ortamında sabit değişken tanımlaması `const` anahtar kelimesi ile yapılır. `const` ile bir değişken sabit tanımlaması yapılırken mutlaka ilk değer ataması yapılmalıdır. `Const` anahtar kelimesinin standart kullanımı aşağıdaki gibidir.

```
const degiskenAdi = ilkDeger;
```

`const` anahtar kelimesi ile sabit değişkenler tanımlayalım.

```
<script type="text/JavaScript">  
  const piSayisi = 3.14;  
  const vizeCarpan = 0.6;  
  const fnaCarpan = 0.4;  
</script>
```

Yukarıdaki örnek olarak tanımlamış olduğumuz sabit değişkenler değiştirilemez değildir fakat bu sabit değişkenlere sonradan herhangi bir değer ataması durumunda bazı web browser'larda hatalar oluşacaktır. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    const piSayisi = 3.14;
    const vizeCarpan = 0.6;
    const fnaCarpan = 0.4;
    piSayisi = 3;
    vizeCarpan = 2;
</script>
```

Yukarıdaki örneğimizde; sabit olarak tanımlamış olduğumuz değişkenlere sonradan değer değiştirme işlemini gerçekleştirdik. Bu değer değiştirme işlemi sonrası çalışan JavaScript kodlarımız bazı tarayıcılarda çalışmayacaktır. Örneğin Firefox ve Chrome gibi tarayıcılar yukardaki bir durumda JavaScript betiklerini çalıştırmaz. Doğal olarak yukarıdaki kodlarımız arasına document.write() ekleyip ekran çıktısını almak istediğimiz durumda boş bir sayfa ile karşılaşırız. Buna dikkat edelim.

BÖLÜM 5:

Operators ve Comments Lines

Bu bölümde Operators (Operatörler) ve Comments Lines (Yorum Satırları) kavramlarından bahsedeceğiz. Bildiğiniz gibi operatörler ile çeşitli karşılaştırma, matematiksel işlemler v.s yapabilmekteyiz. Yorum satırları ile de kodlarımıza yorum satırları ekleyebilmekteyiz.

5.1 Operators

Operators yani Operatörler her programlama dilinde ekstra olarak artıp / azalmakla birlikte temel olarak aşağıdaki gibidir.

5.1.1 Simple Assignment Operator

Atama operatörünü bir önceki bölümde görmüştük ama yine de bahsedelim. = (eşittir) işareti genel olarak çoğu programlama dilinde atama operatörü olarak kullanılır. Atama operatörüyle ilgili bir örnek yazalım.

```
<script type="text/JavaScript">
    var sayi = 42;
    var PI_SAYISI = 3.14;
    var isim = "emrecan-oztas";
</script>
```

Bu yazdığımız değişkenleri yazdıralım.

```
document.write(sayi + "<BR>" + PI_SAYISI + "<BR>" + isim);
```

5.1.2 Arithmetic Operatorss

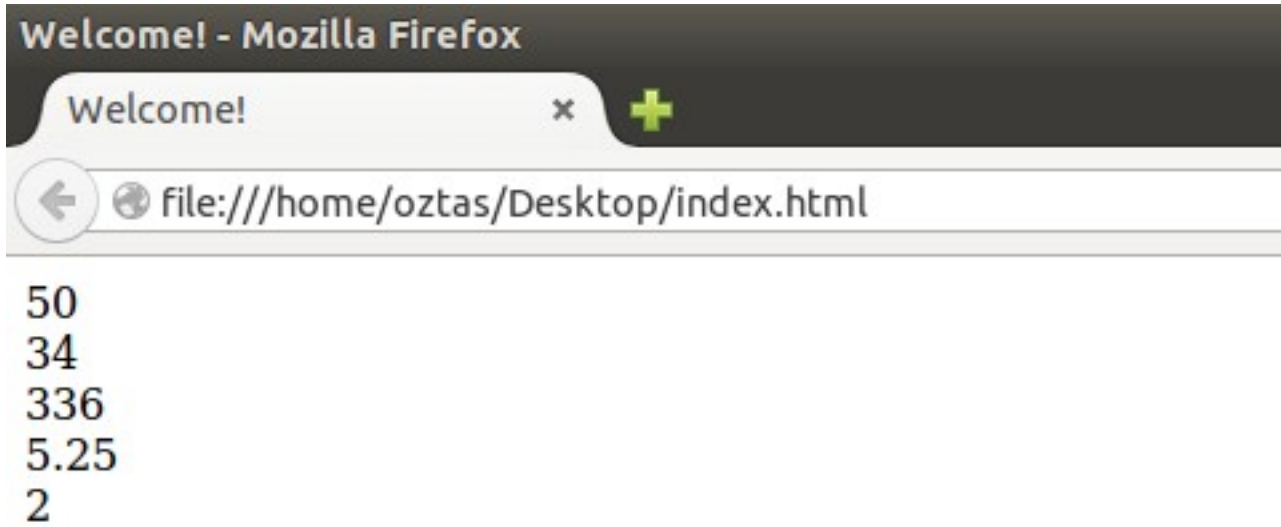
- + Toplama operatörü (Aynı zamanda metin birleştirme - Concatenation)
- Çıkarma Operatörü
- * Çarpma Operatörü
- / Bölme Operatörü
- % Mod Operatörü

Yukarıdaki operatörleri kullanarak çeşitli matematiksel işlemler gerçekleştirebiliyoruz. Şimdi bu operatörler ile örnek yapalım.

```
<script type="text/JavaScript">
    var sayi1 = 42;
    var sayi2 = 6;
    sonuc1 = sayi1 + sayi2;
    sonuc2 = sayi1 - sayi2;
    sonuc3 = sayi1 * sayi2;
    sonuc4 = sayi1 / sayi2;
    sonuc5 = sayi1 % sayi2;
```

```
</script>
```

Yukarıda yazmış olduğumuz `SONUC` satırlarını yazdıralım ve ekran çıktılarını alalım.

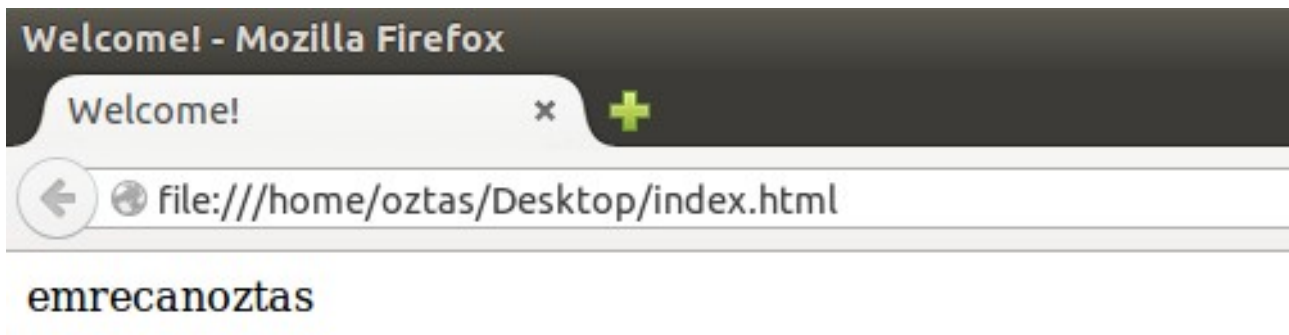


Burada bahsetmek istediğim diğer bir konu daha var. `+` operatörünü toplama operatörü olarak biliyoruz. `+` operatörü aynı zamanda concatenation (birleşme) operatörü olarakta bilinir. Daha önceki bölümlerde uyguladık fakat herhangi bir açıklama yapmadık. `+` operatörü ile **sayısal - metinsel** ve **metinsel - metinsel** ifadeleri birleştirebiliriz. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var isim = "emrecan";
    var soyad = "oztas";
    var isimSoyad = isim + soyad;
    document.write(isimSoyad);
</script>
```

Yukarıdaki örneğimizde; `isim` ve `soyad` adında tanımladığımız değişkenlere metinsel olarak değerlerini girdik. Üçüncü bir değişken olarak `isimSoyad` adında bir değişken tanımladık ve `+` operatörü ile `isim` ve `soyad` değişkenlerini, `isimSoyad` değişkeninde concatenation (birleştirme) yaptık. Sonrada `isimSoyad` değişkenini ekrana yazdırdık.

Yukarıdaki örneğimiz için ekran çıktısı alalım.



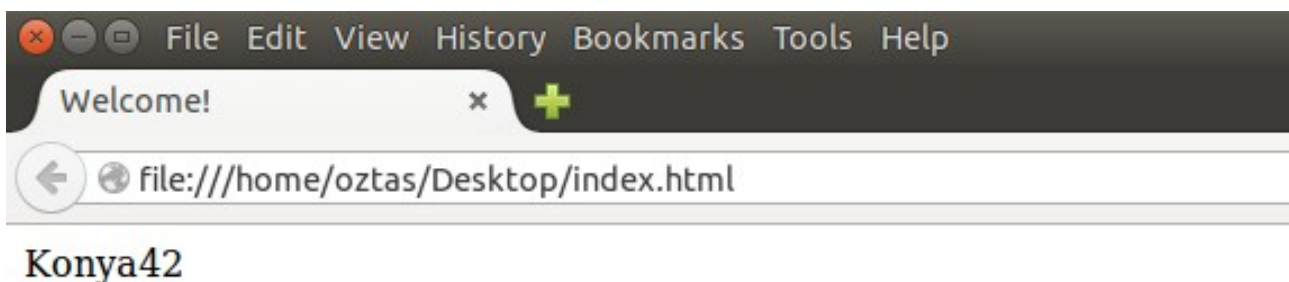
Yukarıdaki ekran çıktısında görüldüğü gibi `metinsel + metinsel` ifadeleri `+` operatörü ile birleştirebiliyoruz.

`+` operatörü ile metinsel – sayısal ifadeleri de birleştirebiliriz. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var sehir = "Konya";
    var plaka = 42;
    var sehirPlaka = sehir + plaka;
    document.write(sehirPlaka);
</script>
```

Yukarıdaki örneğimizde; `sehir` ve `plaka` adında iki tane değişken tanımladık ve `metinsel - sayısal` olarak değerlerini girdik. Bir üçüncü değişken olarak `sehirPlaka` değişkenini tanımladık ve `+` operatörü ile `sehir` ve `plaka` değişkenlerini `sehirPlaka` değişkeninde birleştirdik. Daha sonra da `sehirPlaka` değişkenimizi ekrana yazdırdık.

Yukarıdaki örneğimiz için ekran çıktısı alalım.



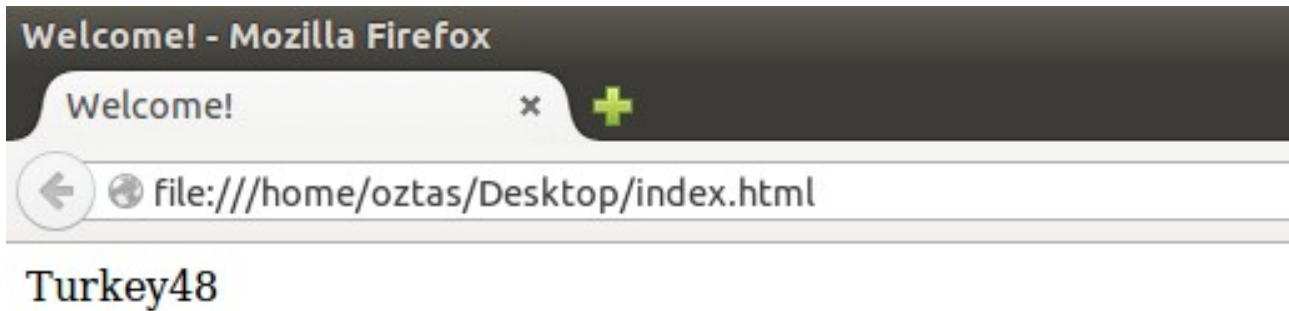
Yukarıdaki ekran çıktısında görüldüğü gibi `metinsel + sayısal` ifadeleri de `+` operatörü ile birleştirebiliyoruz.

+ operatörü ile sayısal – sayısal ifadeleri birleştirdiğimiz zaman toplama işlemini yapıyor bunu söylemeye bile gerek yok. Fakat hem metinsel hem de sayısal ifadeleri kullanıp aynı anda sayısal ifadeleri toplayıp metinsel ifadeyle birleştirmek istersek o zamanda () işaretlerini kullanacağız. Aşağıdaki örneğimize bakalım.

```
<script type="text/JavaScript">
    var sayi1 = 42;
    var sayi2 = 6;
    var isim = "Turkey";
    var total = isim + (sayi1 + sayi2);
    document.write(total);
</script>
```

Yukarıdaki örneğimizde sayi1 ve sayi2 adında iki değişken tanımlayıp sayısal değerlerini girdikten sonra üçüncü bir değişken tanımlayıp metinsel değerini girdik. + operatörü ile bu ifadeleri birleştirirken; sayısal ifadeleri toplamak istediğimiz için () işaretlerini kullandık. Kullanmasaydık bütün ifadeler birleştirilecek ve ekran çıktısı olarak: Turkey426 sonucunu alacaktık. Fakat () kullandığımız için sonuç değişti.

Yukarıdaki örnek için ekran çıktısı alalım.



Yukarıdaki ekran çıktısında görüldüğü gibi () işaretleri ile ulaştığımız sonuç.

5.1.3 Unary Operators

- + Pozitif Sayı Operatörü
- Negatif Sayı Operatörü
- ++ 1 Arttırma Operatörü
- 1 Azaltma Operatörü
- ! Mantıksal Tümlleyen Operatörü. Aynı zamanda boolean ifadelerin tersini alma işlemini gerçekleştirir.

Unary operatörlerimizden + (pozitif) ve - (negatif) operatörlerini biliyoruz. Sayıların önüne gelerek pozitif ve negatif kavramını oluştururlar. O yüzden diğer unary operatörler üzerinde duralım.

++ ve -- operatörler ile değişken değerleri 1 arttırıp ve 1 azaltabiliriz.

Örneğin elimizde bir değişken olsun.

```
var a = 5;
```


Bu değişkenin değerini 1 arttırmak veya 1 azaltmak için aşağıdaki işlemlerden herhangi birini uygulayabiliriz.

```
a = a + 1;  
a = a - 1;
```

Veya

```
a += 1;  
a -= 1;
```

Yukarıda işlemlerin aynısı ++ ve -- operatörleri ile de gerçekleştirebiliriz. Burada ++ ve -- operatörleri iki farklı yerde kullanılabilir. Ne demek istediğimizi örnekler ile açıklamaya çalışalım.

```
a++;  
a--;
```

Yukarıdaki örneğimizde a değişkenin değeri 1 arttırıp, azaltılmaktadır. Bu iki operatörleri bir arada kullanmak zorunda değilsiniz. O an hangisi ihtiyacınızı görüyorsa onu kullanmalısınız. Örneklerimiz her ikisini de bir arada içermektedir.

Örneğimize geri dönelim. Eğer yukarıdaki gibi bir ifade yazar isek değişkenlerimizin değeri bir sonraki satırda değişecektir. Yani derleme işlemi sırasında önce a sayısı okunacak sonra ++ ve -- operatörleri okunacaktır. Bir sonraki satırda da işlemler gerçekleştirilecektir. Aşağıdaki temsili derleme işlemlerini inceleyelim.

```
a = 5;  
> a++ (a = 5)  
> a = 5 + 1  
a = 6  
  
a = 5;  
> a-- (a = 5)  
> a = 5 - 1  
a = 4
```

++ ve -- operatörleri değişkenlerin önünde de bulunabilir.

```
++a;  
--a;
```

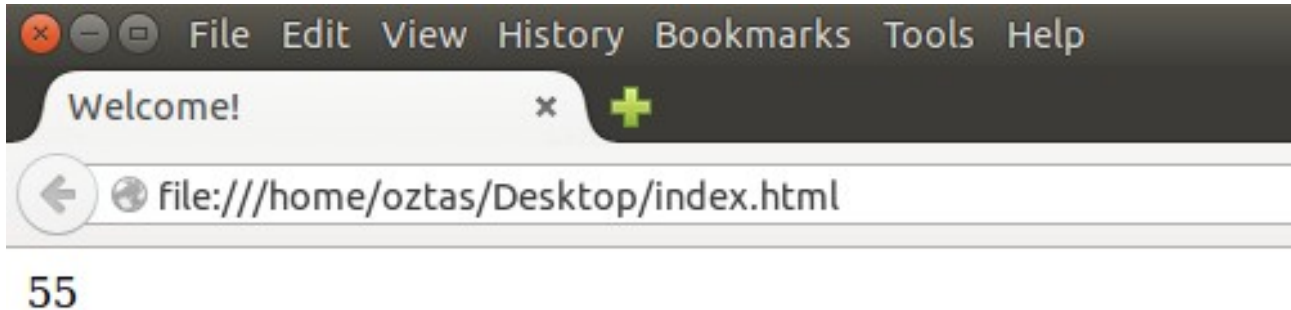
++ ve -- operatörleri değişkenin önünde bulunduğu zaman ise derleme işlemi o satıra geldiği zaman hemen değişkenin içeriği bir arttırılacak veya azaltılacaktır. Aşağıdaki temsili derlemeye işlemini inceleyelim.

```
a = 5;  
> ++a (a = 5 + 1)  
a = 6  
  
a = 5;  
> --a (a = 5 - 1)  
a = 4
```

++ ve -- operatörlerinin, bu iki kullanımları arasındaki farkı aşağıdaki örneği inceleyerek göstermeye çalışalım.

```
<script type="text/JavaScript">
  var a = 5;
  document.write(a++);
  var b = 5;
  document.write(b--);
</script>
```

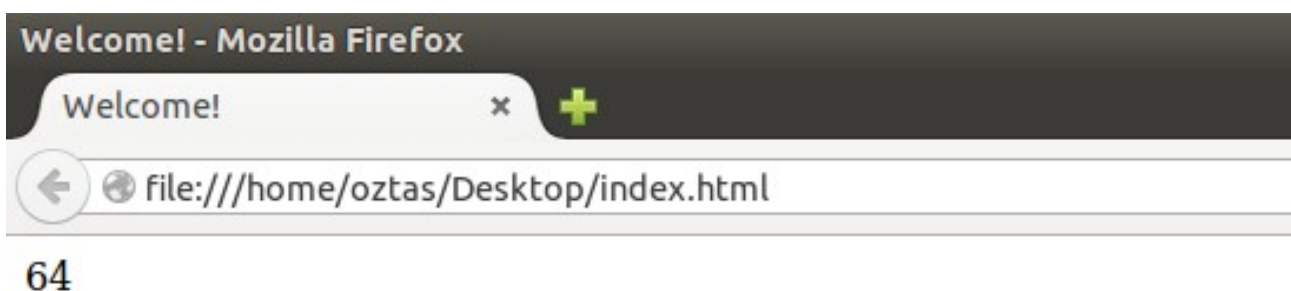
Yukarıdaki yazmış olduğumuz örneğin ekran çıktısını alalım.



Yukarıdaki ekran alıntısında görüldüğü gibi değişkenlerimizin değerleri değişmemiştir. ++ ve -- operatörlerini değişken önünde kullanarak yukarıdaki örneği bozmadan devam edelim.

```
<script type="text/JavaScript">
  var a = 5;
  document.write(++a);
  var b = 5;
  document.write(--b)
</script>
```

Yukarıdaki örneğimiz için ekran çıktısı alalım.

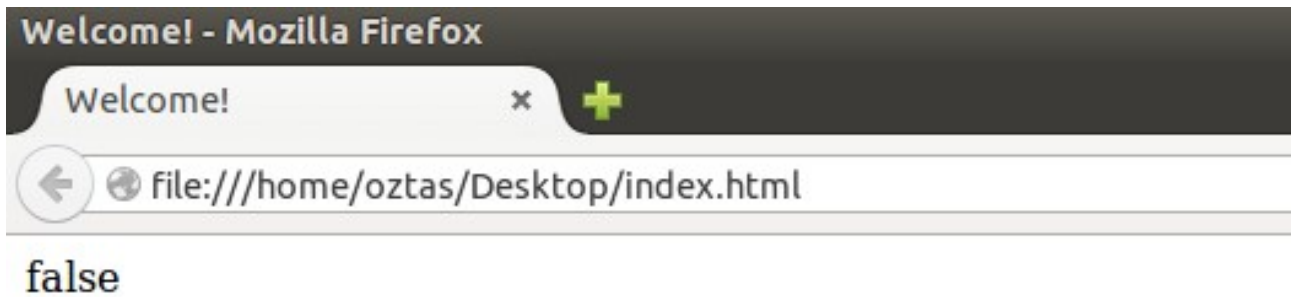


Yukarıdaki ekran çıktısında görüldüğü değişkenlerimizin değerleri değişmiş durumdadır. ++ veya -- operatörlerinin bu iki farklı kullanımına dikkat edelim.

Diğer bir unary operatörü olan ! Operatörüne geçelim. Programlama dünyasında ! işareti genellikle bir durumun değil veya tam dersi olarak bilinir. Bu durum boolean ifadeler için geçerlidir. Boolean ifadelerin true ve false değerleri olduğunu biliyoruz. ! Operatörünü ilerleyen bölümlerde özellikle control statement bölümde sıklıkla kullanacağız. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var durum1 = true;
    var durum2 = !durum1;
    document.write(durum2);
</script>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Yukarıda ekran alıntısını incelediğimizde durum1 değişkenini true olan değeri durum2 değişkeninde ! Operatörü ile false olarak değiştirilmiştir.

5.1.4 Equality and Relational Operators

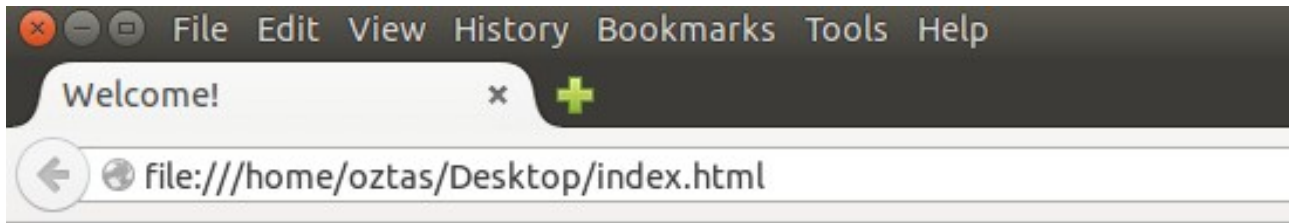
===	Denktir
==	Eşittir
!=	Eşit değildir
>	Büyüktür
>=	Büyük veya eşittir.
<	Küçüktür.
<=	Küçüktür veya eşittir.

JavaScript diğer programlama dillerinden ekstra olarak denklilik kontrolü sağlar. Mesela denktir operatörü java'da bulunmamaktadır.

Denktir operatör === işaretleriyle sağlanır. Denktir operatörü iki ifadenin hem tip hem de içeriğini kontrol eder. İki ifadenin içerikleri ve tipleri aynıysa true değilse false değerini döndürür. Denktir işareti iki ifadenin içeriğini kontrol ederken case sensitive durumunu takip eder. Yani "E" ile "e" birbirine denk değildir. Aşağıdaki örneğimize bakalım.

```
<script type="text/JavaScript">
    var kelimeBir = "Turkey";
    var kelimeIki = "Turkey";
    document.write(kelimeBir === kelimeIki);
</script>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Yukarıdaki örneğimizde `kelimeBir` ve `kelimeIki` değişkenlerinin içeriği ve tipleri aynı olduğu için sonuç `true` olarak çıkmıştır. Lakin herhangi bir değişkenin değeri “turkey” olsaydı sonuç `false` olarak çıkacaktı.

Diğer yandan metinsel değişkenler de sayısal ifadeler alabilirler. Bir de bu yönden bakalım ve bir örnek yazalım.

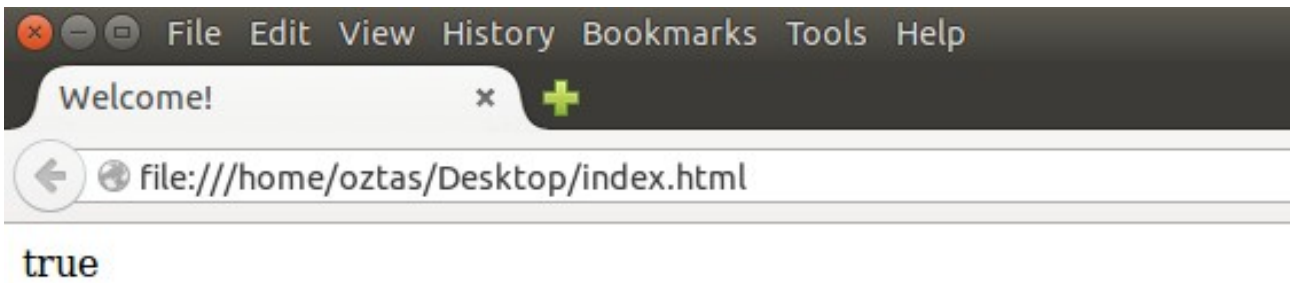
```
<script type="text/JavaScript">
    var sayi = 1453;
    var kelime = "1453";
    document.write(sayi === kelime);
</script>
```

Yukarıdaki örneğimizde içeriklerin aynı olması denklik durumuna yetmeyecektir. Tip olarak incelendiğinde `sayi` değişkeni `int` yani tam sayı bir değişken iken `kelime` değişkeni `String` yani metinsel bir değişkendir. Yazdığımız örneğin ekran çıktısını aldığımızda durumun `false` olduğunu göreceğiz.

Yukarıdaki örneğimizi `==` eşittir operatörü ile yazalım ve sonuçlarını gözlemleyelim.

```
<script type="text/JavaScript">
    var sayi = 1453;
    var kelime = "1453";
    document.write(sayi == kelime);
</script>
```

Ekran çıktısını alalım.

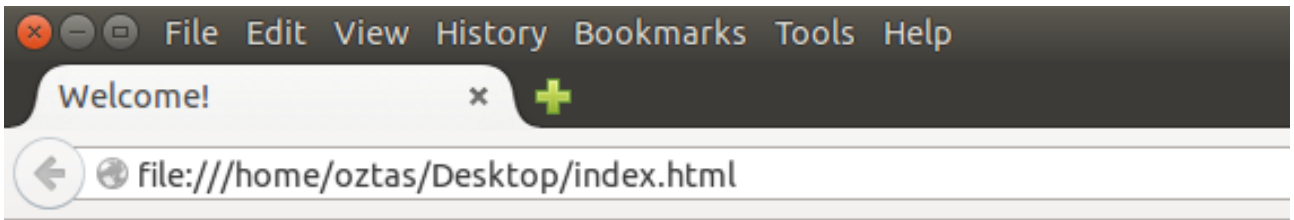


Ekran çıktımız `true` oldu. Çünkü `eşittir` operatörü `denktir` operatörünün aksine tip kontrolü yapmaz, sadece içerikleri karşılaştırır. O yüzden ekran çıktımız `true` olarak gerçekleşmiştir.

Diğer operatörlerimizin tümünü kullanarak bir örnek yazalım. Ve ekran çıktılarını inceleyelim.

```
<script type="text/JavaScript">
  var a = 5;
  var b = 6;
  document.write("a nin degeri: " + a);
  document.write("<br>");
  document.write("b nin degeri: " + b);
  document.write("<br>");
  document.write("Eşit degildir: " + (a != b));
  document.write("<br>");
  document.write("Buyuktur: " + (a > b));
  document.write("<br>");
  document.write("Buyuk veya esittir: " + (a >= b));
  document.write("<br>");
  document.write("Kucuktur: " + (a < b));
  document.write("<br>");
  document.write("Kucuk veya esittir: " + (a <= b));
</script>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



a nin degeri: 5
b nin degeri: 6
Eşit degildir: true
Buyuktur: false
Buyuk veya esittir: false
Kucuktur: true
Kucuk veya esittir: true

Yukarıdaki ekran çıktısını inceyelim.

`!=` operatörü, iki ifadenin aynı olması durumunda `false`, diğer durumda `true` değerini alacaktır. Örneğimizde de iki ifade birbirine eşit olmadığı için `true` değerini almıştır.

`>` operatörü, iki ifadeyi büyüklük bakımından kıyaslar. Eğer operatörün solundaki değer sağındaki değerden büyükse `true`, diğer durumlar da `false` değerini alacaktır. `a` değeri `b` değerinden büyük olmadığı için `false` değerini almıştır.

`>=` operatörü, iki ifadeyi büyüklük ve eşitlik bakımından kıyaslar. Eğer operatörün solundaki değer sağındaki değerden büyük veya eşit ise `true`, diğer durumda `false` değerini alacaktır. Örneğimizde `a` sayısı `b` sayısından büyük veya eşit değildir. O yüzden sonucumuz `false` çıkmıştır.

`<` operatörü, iki ifadeyi küçüklük bakımından kıyaslar. Eğer operatörün solunda değer sağındaki değerden küçük ise `true`, diğer durumlarda `false` değerini alacaktır. `a` değeri `b` değerinden küçük olduğu için sonuç `true` çıkmıştır.

`<=` operatörü, iki ifadeyi küçüklük ve eşitlik bakımından kıyaslar. Eğer operatörün solundaki sayı sağındaki sayıdan küçük veya eşit ise `true`, diğer durumda `false` değerini alacaktır. `a` sayısı `b` sayısından küçük olduğu için sonuç `true` çıkmıştır.

5.1.5 Conditional Operators

`&&` AND (VE)
`||` OR (VEYA)
`?:` if-then-else

Koşul operatörleri ifadeleri `Logic` (Mantıksal) kontrolünü yapar. Daha önce herhangi bir soyut matematik, sayısal tasarım veya ona benzer bir mantık dersi aldınız mı bilmiyorum. Bu kavramlar bu gibi derslerde sıklıkla işlenir. Daha önce bu derslerden herhangi birini almamış olsanız bile sorun yok. Hepsine ayrı ayrı değineceğiz.

Bildiğiniz gibi bilgisayarlar 1 ve 0 lardan anlarlar. 1 true, 0 false demektir. AND ve OR ifadeleri de belli bir mantığa göre bu 1 ve 0 ları çözümlerler. Aşağıdaki tabloyu inceleyelim.

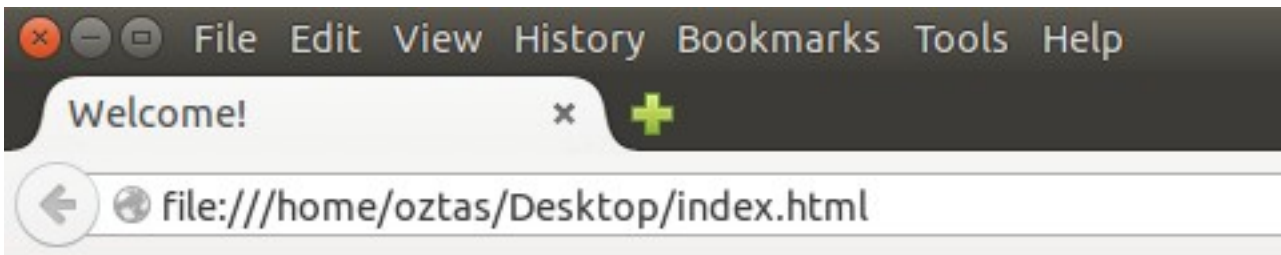
AND Tablosu:

X	Y	AND-Result
0	0	0
0	1	0
1	0	0
1	1	1

Tabloda görüldüğü gibi AND değeri her iki ifadeninde 1 yani true olması durumunda true değerini almaktadır. Bunu bir örnekle pekiştirelim.

```
<script type="text/JavaScript">
  var x1 = 1;
  var y1 = 0;
  document.write("x1: " + x1);
  document.write("<br>");
  document.write("y1: " + y1);
  document.write("<br>");
  document.write("Birinci Durum: ");
  document.write(x1 && y1);
  document.write("<br>");
  var x2 = 1;
  var y2 = 1;
  document.write("x2: " + x2);
  document.write("<br>");
  document.write("y2: " + y2);
  document.write("<br>");
  document.write("İkinci Durum: ");
  document.write(x2 && y2);
</script>
```

Örneğimiz için ekran çıktısı alalım.



x1: 1
y1: 0
Birinci Durum: 0
x2: 1
y2: 1
İkinci Durum: 1

Görüldüğü gibi tablomuzu elde ettik. Burada değişkenlere 1 veya 0 vermek yerine true ve false değerleri de verebiliriz. Aşağıdaki örnekte olduğu gibi.

```
<script type="text/JavaScript">
  var x = true;
  var y = false;
  document.write(x && y);
</script>
```

Yukarıdaki örneğimizin ekran çıktısını aldığımızda `false` değerini çıktı olarak alırız.

Bir diğer koşul operatörümüz olan OR (veya) operatörü de mantıksal işlemler gerçekleştirir demiştik. OR koşul operatörünün de tablosunun inceleyelim.

OR Tablosu:

X	Y	OR-Result
0	0	0
0	1	1
1	0	1
1	1	1

OR tablosuna göre iki ifadenin 0 olması durumunda veya false olması durumunda 0 false değerini vermektedir. Diğer durumlarda hep 1 true değerini göstermektedir. Bununla ilgili bir örnek yapalım.

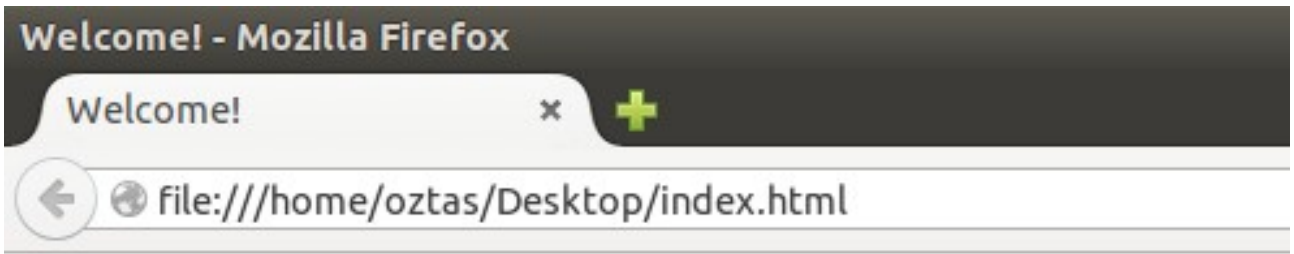
```
<script type="text/JavaScript">
  var x = true;
  var y = false;
  document.write(x || y);
</script>
```

Yukarıdaki örneğimizin çıktısını aldığımızda `true` sonucuyla karşılaşırız.

Koşul operatörlerin son olarak `?:` (if-then-else) yapısına bakalım. Bu operatörler yardımıyla `if-else` yapısı kurabiliyoruz. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
  var a = 42;
  var b = 6;
  var sonuc = (a > b) ? "Buyuktur" : "kucuktur";
  document.write(sonuc);
</script>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



a, b den buyuktur

Ekran çıktısında görüldüğü gibi “a, b den buyuktur” ifadesini aldık.

? : operatörleri hakkında biraz konuşalım ve yazdığımız örneğimizi inceleyelim. ? Operatörü sol tarafında bulunan karşılaştırma sonucu sağlanıyorsa, sağ tarafında ki değeri işletir. Aksi durumda ise : operatöründen sonraki değeri işletir. Biz sonuc adında bir değişkene çıkan değeri verdik. Doğrudan da yazdırabilirdik:

```
document.write((a > b) ? "Buyuktur" : "kucuktur");
```

?: operatörlerini kullanarak daha fazla kontrol ifadesi yazabiliriz. Bununla ilgili bir örnek yazalım.

```
<script type="text/JavaScript">
    var a = 98;
    sonuc = a < 90 ? "Dar Aci" : a > 90 ? "Genis Aci" : "Dik Aci";
    document.write(sonuc);
</script>
```

Yukarıdaki örneğimizde a değişkenine bir aç değeri girdik ve bu açının geniş – dik – dar aç bakımından kontrol ettik. Daha önce ? Operatörünün sol taraftaki durum sağlandığı zaman sağ tarafında ki durumu işletir, aksi halde : operatöründe sonra ki değeri işletir demiştik. Bu tamamen if – else yapısıdır. If – else if – else yapısını sağlamak içinde yukarıdaki formu uygulamak gerekir. Yine aynı şekilde ? operatörü sol taraftaki durum sağlandığı zaman sağ tarafında ki durumu işletiyor. Aksi bir durum ortaya çıktığı durumda ise : operatörünün sağ tarafına bakıyor. : operatörünün sağ tarafına yine bir koşul yazdık ve sonra ? Operatörüyle bir kontrol daha yaptık. Durum yine aynı şekilde devam etti. Bu yapıyı kullanarak istenildiği kadar if-else if yapısı oluşturulabilir.

5.1.6 Bitwise and Bit Shift Operators

~	tersini alma
<<	Sola Kaydırma
>>	Sağa Kaydırma
&	AND
	OR
^	XOR

Daha önce bilgisayarların 0 ve 1'lerden anladığından bahsetmiştik. Şuan kullandığımız sayı sistemi Decimal (Onluk)'tur. Fakat bilgisayarlar Binary (İkili) sayı sistemini kullanırlar. Örneğin 5 sayısının binary karşılığı 0101'dir. Bitsel işlemler içinde yukarıda vermiş olduğumuz operatörler kullanılmaktadır.

~ operatörü verilen sayıyı binary tabana çevirmekte daha sonra 1 ekleyip negatif yapmaktadır. Örneğin elimizde 5 sayısı olsun. Bu sayısı temsili olarak derlemeye tabi tutalım.

```
a = 5;
~a:
> a = 0000 0000 0000 0000 0000 0000 0000 0101
> a = 0000 0000 0000 0000 0000 0000 0000 0101 + 1
> a = 0000 0000 0000 0000 0000 0000 0000 0110
> a = 1000 0000 0000 0000 0000 0000 0000 0110
a = -6
```

<< operatörü verilen sayıyı belirtilen kadar sola iteleme yapar. İteleme yaparken sol tarafa kaydırıldığı basamak kadar 0 koyar. Aşağıdaki temsili derlemeye bakalım.

```
a = 5;
a << 2;
> a = 0000 0000 0000 0000 0000 0000 0000 0101
> a = 0000 0000 0000 0000 0000 0000 0000 1010
> a = 0000 0000 0000 0000 0000 0000 0001 0100
a = 20
```

Yukarıda görüldüğü gibi a değişkeni 20 değerini almıştır. Eğer a değişkenini 3 kez sola ötelemiş olsaydık o zamanda 40 değerini alacaktır.



int tipindeki bir değişken hafızada 32 bit uzunluğa sahiptir. Biz de yazdığımız değişkenlerde int tipini kullandık. Çünkü JavaScript değişken tanımlaması yaparken herhangi bir data type (veri tipi) yapısı tanımlamaya gerek duymamaktadır.

>> operatörü, << operatörünün yaptığı işlemin tam tersini yapmaktadır. Yani verilen sayıyı sağa doğru iteler ve itelediği miktarda sol tarafa 0 koyar. Elimizde yine 5 sayısı olsun ve bu sayıyı 2 defa sağa doğru öteleyelim. Aşağıdaki temsili derlemeye bakalım.

```
a = 5;
a >> 2;
> a = 0000 0000 0000 0000 0000 0000 0000 0101
> a = 0000 0000 0000 0000 0000 0000 0000 0010
> a = 0000 0000 0000 0000 0000 0000 0000 0001
a = 1
```

Yukarıda görüldüğü 5 sayıyı iki defa sağa doğru itelendiği zaman 1 değerini almaktadır. 5 sayısını 2'den fazla bir sayıda sağa doğru itelediğimizde her seferinde sonuç 0 olacaktır.

& operatörü verilen iki değeri bit bit karşılaştırıp AND (Ve) mantığını çalıştırmaktadır. Conditional Operators (Koşul Operatörleri) bölümünde oluşturduğumuz tabloya göre hareket eder. Örneğin elimizde 5 ve 10 sayıları olsun. Bu sayıların temsili derleme işlemine bakalım.

```
a = 5;
b = 10;
c = a & b;
a = 0000 0000 0000 0000 0000 0000 0000 0101
b = 0000 0000 0000 0000 0000 0000 0000 1010
c = 0000 0000 0000 0000 0000 0000 0000 0000
c = 0
```

Yukarıdaki örnekte görüldüğü gibi & operatörü iki sayıyı binary tabanında bit bit karşılaştırıp iki bitinde 1 yani true olması durumunda 1 diğer durumlarda 0 yani false yapmaktadır. Verdiğimiz örnekte 1 ve 1 sayıları hiç karşılaşmadıkları için tüm değerler 0 olmuştur.

| operatörü de & operatöründe olduğu gibi verilen iki sayıyı bit bit karşılaştırıp farklı olarak OR (Veya) işlemini gerçekleştirmektedir. Conditional Operators (Koşul Operatörleri) bölümünde oluşturduğumuz tabloya bakabilirsiniz. & konusunda olduğu gibi örneğimiz 5 ve 10 sayıları olsun. Bu sayıların temsili derleme işlemine bakalım.

```
a = 5 ;
b = 10 ;
c = a ^ b ;
a = 0000 0000 0000 0000 0000 0000 0000 0101
b = 0000 0000 0000 0000 0000 0000 0000 1010
c = 0000 0000 0000 0000 0000 0000 0000 1111
c = 15
```

Yukarıdaki OR işlemine tabi tuttuğumuz örnekte; sonumuz 15 çıkmıştır. Belirttiğimiz tabloyu incelediğiniz de neden böyle bir durumla karşılaştığımızı çok iyi anlayacaksınız.

^ operatörü & ve | operatörlerinde olduğu gibi verilen iki sayıyı bit bit karşılaştırıp diğerlerinden farklı olarak XOR işlemine tabi tutmaktadır. XOR yani d i ş l a m a l ı ve tablosu aşağıdaki gibidir.

X	Y	XOR-Result
0	0	1
0	1	1
1	0	1
1	1	0

Yukarıdaki tabloya göre iki değerin 1 olması durumunda 0 yani false diğer değerlerde 1 yani true değerini almaktadır. Bu durumu bir örnek bir derleme işlemiyle anlatalım. Örneğin sayılarımız 3 ve 5 olsun. Aşağıdaki örnek derleme işlemine bakalım.

```
a = 3 ;
b = 5 ;
c = a ^ b ;
a = 0000 0000 0000 0000 0000 0000 0000 0011
b = 0000 0000 0000 0000 0000 0000 0000 0101
c = 0000 0000 0000 0000 0000 0000 0000 0110
```

Yukarıdaki örnek derleme işlemini incelediğimizde ne demek istediğimizi anlayacaksınız sanıyorum.

5.2 Comment Lines

Bazen daha önce yazdığınız kodları geriye dönüp baktığınızda hatırlayamayabilirsiniz. Bu çok doğaldır. Her programcının başına gelir. Kendi kodlarınızı anlayamamanın diğer bir boyutu da yazdığınız kodları sizden sonra gelecek olan programcıların anlayamaması. İşte en büyük korkumuz budur. Çünkü yaptığınız işin devamlılığı esastır. Sizden sonra da gelecek olanlar olacaktır. Yazdığınız veya yaptığınız işlemlerin açıklayıcı olması gerekir. O yüzden kodların olabildiğince açıklayıcı olması gerekir. Hal böyle olunca her programlama dilinde yorum satırları oluşturulmuştur. Bu satırlar sayesinde yazdığımız kodların başlarına hatırlatıcı bilgilendirmeler yazabiliriz.

Yorum satırları yazıldığı zaman derleyici ya da yorumlayıcı bu satırları görmezden gelir. Doğal olarak herhangi bir problem oluşmaz. JavaScript'te iki tip yorum satırı vardır. Bunlar: single-line (tek satırlı) ve multi-line (çok satırlı) yorum satırlarıdır.

Single-Line (Tek Satırlı) yorum satırları // karakterleriyle sağlanır. Bu karakterlerden sonra o satır boyunca yorum yazabilirsiniz. Aşağıdaki örneğimize bakalım.

```
<script type="text/JavaScript">
    // a sayısına 5 degeri atandi
    var a = 5;
    //b sayısına 10 degeri atandi
    var b = 10;
    // c, a ve b degiskenlerinin toplami
    c = a + b;
</script>
```

Yukarıdaki örnekte görüldüğü gibi tek satırlı yorum satırlarımızı yazdık. Bu çok basit bir örnek yazmasanız daha evladır fakat karmaşık kodlar yazarken yorum satırları yazarsanız daha sonra kodlarınıza baktığınızda ne yaptığınızı anlamanız kolaylaşır.

Multi-Line (Çok Satırlı) yorum satırları /* */ karakter kümesiyle sağlanır. /* işaretlerini açtıktan sonra açıklama satırlarımızı yazıp sonra */ işaretiyle kapatmalıyız. Aşağıdaki örnekte olduğu gibi.

```
<script type="text/JavaScript">
    /* a sayısına 5 degeri atandi
       b sayısına 10 degeri atandi
       c, a ve b degiskenlerinin toplami
    */
    var a = 5;
    var b = 10;
    c = a + b;
</script>
```

BÖLÜM 6:

prompt ve confirm

HTML Form elementleri dışında doğrudan JavaScript kodları kullanarak kullanıcıyla bilgi alışverişinde bulunabiliriz. Pek sağlıklı bir yöntem olarak görmesem de bazı durumlarda işe yarayabilir, neden olmasın.

6.1 prompt

prompt, klavyeden bilgi girişi almak için kullanılan bir komuttur. Bu komutun iki farklı kullanım stili vardır. Bunlardan birincisi:

```
prompt("sorulacak soru");
```

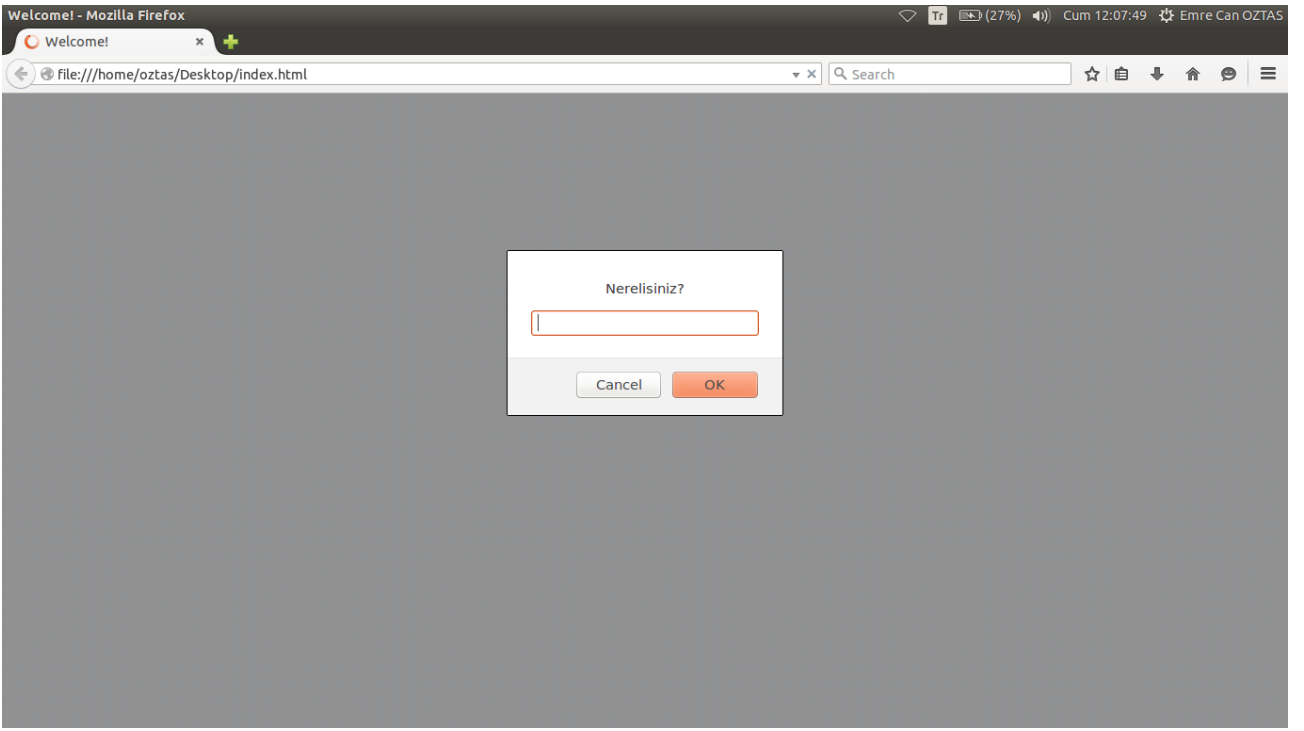
ikincisi ise:

```
prompt("sorulacak soru", "cevap ornegi");
```

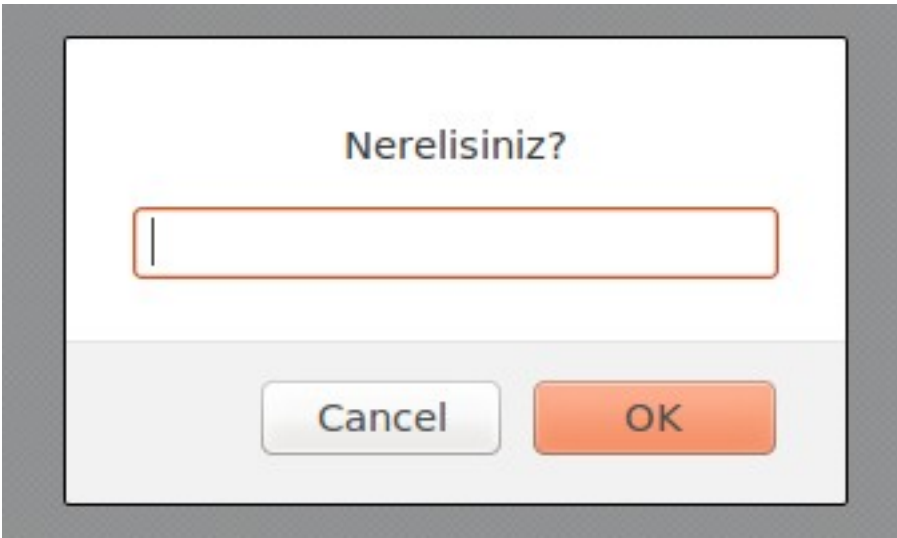
Her iki kullanım stilini kullanarak örnek yapalım. Aşağıdaki birinci örneğimizi inceleyelim.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    prompt("Nerelisiniz?");
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Ekran çıktımıza daha yakından bakalım.



Yukarıdaki ekran çıktısında gördüğünüz gibi prompt ile kullanıcıdan bilgi alabiliriz. Yalnız burada belirtmek istediğim bir şey var. Eğer prompt komutunu yukarıdaki gibi herhangi bir değişkene bağlı olmadan kullanırsanız; kullanıcıdan gelen bilgi farazi olacaktır yani uçup gider. Bilgiyi almanızın bir anlamı kalmaz. O yüzden prompt komutunu kullanırken bir değişkene bağlamamız lazım gelir. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var soru = prompt("Nerelisiniz?");
</script>
```

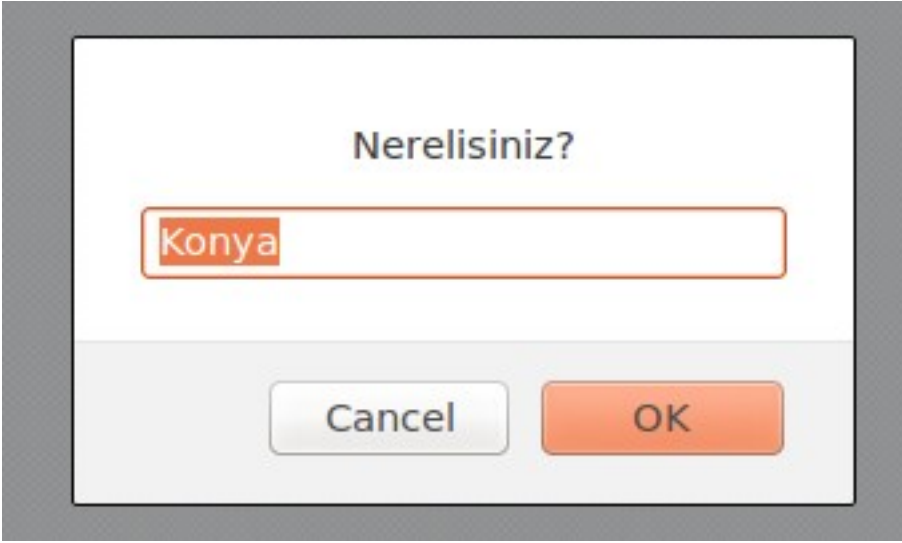
Yukarıdaki örneğimizde görüldüğü gibi prompt komutunu bir değişkene bağladık. Artık kullanıcıdan alınan değerleri istediğimiz gibi kullanabiliriz. Buna dikkat edelim. prompt komutunun ikinci kullanım stiline bakalım. Birinci stilde yazdığımız örneğimizi ikinci stilde yazalım.

```
<script type="text/JavaScript">
```



```
// prompt komutunu bir degiskene
// baglamamiz gerektigini unutmayalim.
var soru = prompt("Nerelisiniz?", "Konya");
</script>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Gördüğünüz gibi varsayılan olarak yazdığımız “Konya” yazısı bizi karşıladı. Kullanıcının verdiği cevap artık soru değişkeninde. İstedğimiz zaman kullanabiliriz.

Daha detaylı bir örnek yapalım. Örneğin kullanıcıdan vize ve final notlarını alıp gerekli hesaplamaları yapıp ekrana yazdıralım. Bunun için aşağıda yazdığımız örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var vize = prompt("Vize notunuz:");
    var fnal = prompt("Final notunuz:");
    var sonuc = (vize * 0.6) + (fnal * 0.4);
    document.write(sonuc);
</script>
```

Yukarıdaki örneğimizde kullanıcıdan vize ve final notlarını istedik. Daha sonra girilen bu notlara göre bir sonu. Hesapladık ve ekrana yazdırdık.



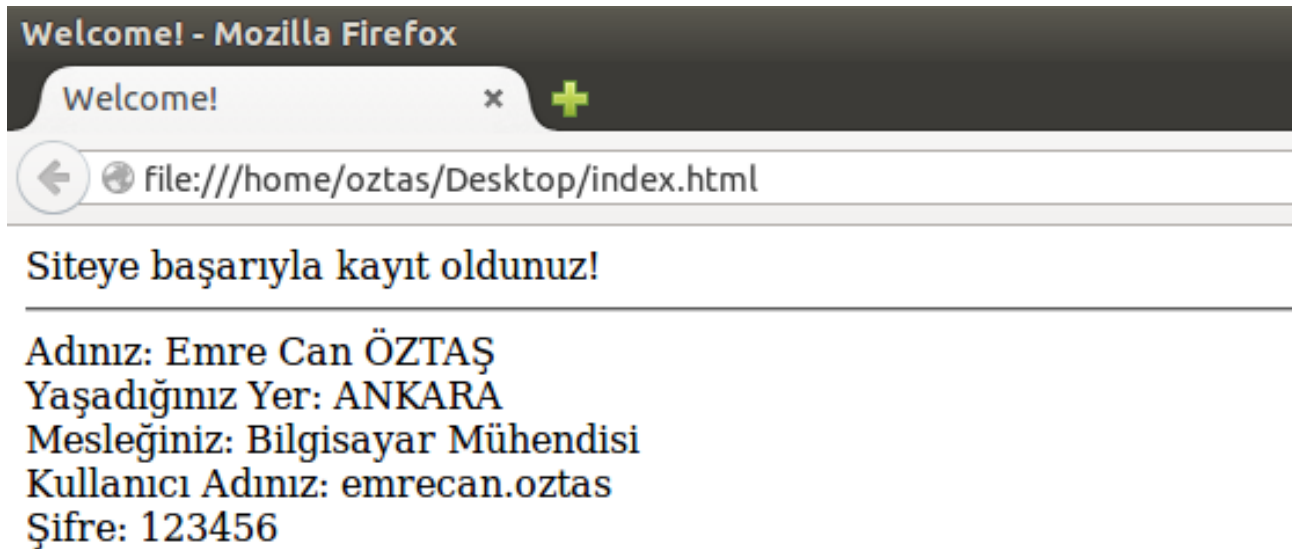
prompt komutuyla kullanıcıdan alınan değer her zaman **String** yani metinsel bir yapıdadır. Yukarıdaki örneğimizde iki değer de sayısal olduğu için herhangi bir sıkıntı çıkmamıştır ve kendi içlerinde tip dönüşümleri (**String** > **int**) yapılmıştır. Daha sağlıklı bir sonuç için tip dönüşümü yapmak gereklidir.

Akılda kalıcı olması için bir örnek daha yazalım. Örneğin bir sitemiz var ve bu siteye üye kaydı işlemleri var. Bu işlemleri de prompt ile yapmak istiyoruz (nasıl bir fantaziye). prompt komutunu kullanarak bu işlemleri yapalım ve ekrana yazdıralım.

```
<script type="text/JavaScript">
    // kullanicidan degerler aliniyor...
    var adSoyad = prompt("Adınız ve Soyadınız:");
    var yer = prompt("Yaşadığınız Yer:");
    var meslek = prompt("Mesleğiniz:");
    var kullanıcıAdi = prompt("Kullanıcı Adı:");
    var sifre = prompt("Şifre");
```

```
// ekrana yazdırılıyor...
document.write("Siteye başarıyla kayıt oldunuz!" + "<BR>");
document.write("<HR>");
document.write("Adınız: " + adSoyad + "<BR>");
document.write("Yaşadığınız Yer: " + yer + "<BR>");
document.write("Mesleğiniz: " + meslek + "<BR>");
document.write("Kullanıcı Adınız: " + kullanıcıAdi + "<BR>");
document.write("Şifre: " + sifre + "<BR>");
</script>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım. prompt ile gelen alanları doldurduğumuz da aşağıdaki gibi bir ekran çıktısı elde etmiş olacağız.



Yukarıdaki örneğimizle birlikte prompt ile bir sıkıntımız kalmadığını varsayarak, bir sonraki konumuz olan `confirm` konusuna geçelim.

6.2 confirm

Bazı durumlarda kullanıcıdan onay almak gerekebilir. Örneğin Windows işletim sisteminde yaptığınız her adımda kullanıcıdan bir onay istenir. İşte bu onay işlemleri JavaScript ortamında da `confirm` komutuyla sağlanır.

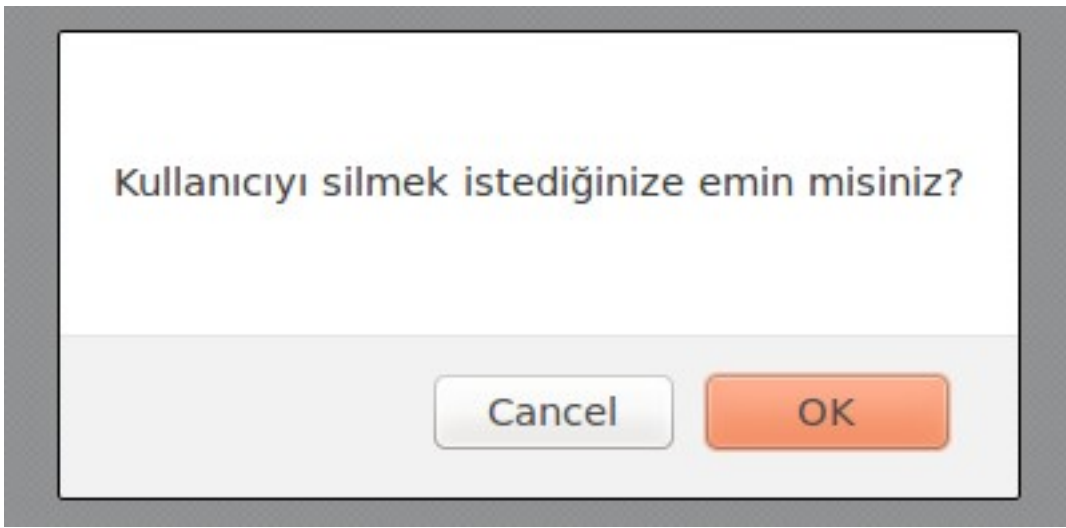
Confirm komutunun standart kullanımı aşağıdaki gibidir.

```
confirm("Onay konusu");
```

`confirm` komutuyla ilgili bir örnek yazalım.

```
<script type="text/JavaScript">
    confirm("Kullanıcıyı silmek istediğinize emin misiniz?");
</script>
```

Yukarıdaki yazmış olduğumuz örneğimizin ekran çıktısını alalım.



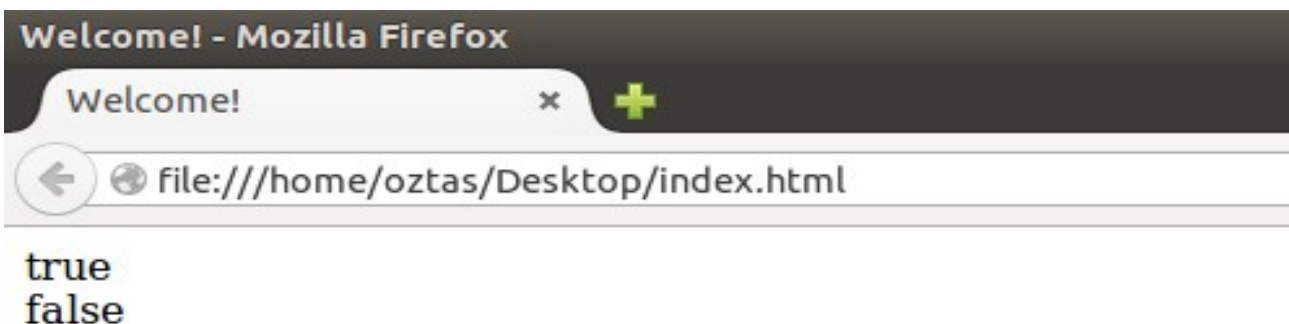
Yukarıdaki ekran çıktısıyla ilgili herhangi bir sorun bulunmamaktadır. Lakin `confirm` komutunu herhangi bir değişkene bağlamadığımız durumda kullanıcının cevabı havada kalır. Yani uçar gider. O yüzden prompt komutunda olduğu gibi `confirm` komutunu da bir değişkene bağlamamız gerekir. Aşağıdaki kullanımda `confirm` komutu bir değişkene bağlanmıştır.

```
<script type="text/JavaScript">
    var onay = confirm("Kullanıcıyı silmek istediğinize emin misiniz?");
</script>
```

Burada size daha önemli bir konudan bahsetmek istiyorum. `confirm` ile alınan onay işleminde; eğer onay verilmiş ise (OK veya Tamam seçilmiş ise) dönen değer `True` olur. Onay verilmemiş ise (Cancel veya İptal seçilmiş ise) dönen değer `False` olur. Bu dediklerimizi bir örnek ile kanıtlayalım.

```
<script type="text/JavaScript">
    // onay1 i onaylıyalım.
    var onay1 = confirm("Birinci durum");
    document.write(onay1);
    // onay2 i onaylamayalım.
    var onay2 = confirm("İkinci durum");
    document.write(onay2);
</script>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında görüldüğü gibi; `confirm` komutunun sonucu: `true` veya `false` olmaktadır.

confirm komutundan dönen değere göre işlem yaptırmakta sizlere kalıyor.

confirm komutunu daha iyi anlamak için daha detaylı bir örnek yazalım.

```
<script type="text/JavaScript">
    var onay = confirm("Silinsin mi?");
    durum = (onay == true) ? "Silindi!" : "Silinmedi!";
    document.write(durum);
</script>
```

Yukarıdaki örneğimizde; ilk başta kullanıcıya “Silinsin mi?” sorusunu confirm komutuyla sorduk. Kullanıcının verdiği cevaba göre (Cancel veya OK)?: operatörleriyle bir denetim yaptık. Confirm komutundan dönen değer true ise “Silindi!”, false ise “Silinmedi!” şeklinde bir çıktı verdik.

BÖLÜM 7:

Data Types ve Type Casting

JavaScript'te değişken tanımlarken `var` anahtar kelimesini kullanıyorduk, biliyorsunuz. Değişkenler bölümünde değişkenlerin veri tipleri konusundan bahsetmedik. Çünkü JavaScript, değişken tanımlarken herhangi bir veri tipi kullanımına gerek görmemektedir. Dolayısıyla değişken tiplerini ve tip dönüşüm işlemlerinden, bu bölümde bahsetmenin daha doğru olacağı kanısındayım.

7.1 Data Types (Veri Tipleri)

JavaScript'in sahip olduğu temel veri tiplerinden bahsedeceğiz. JavaScript yapısı gereği 4 tür veri tipini desteklemektedir. Bunlar: `int`, `String`, `boolean` ve `float`. Diğer programlama dillerinde, örneğin Java'da 8 tür primitif veri tipi vardır, `String`'i de sayarsanız 9 tiptir. Bunlar: `int`, `short`, `char`, `boolean`, `float`, `double`, `long`, `byte` ve `String`'tir. Fakat JavaScript; sayısal veri türleri için: `number`, metinsel veri türleri için: `string` ve mantıksal veri tipi için de: `boolean` demektedir. Aşağıdaki tabloyu inceleyelim.

General Data Types	JavaScript Data Types
<code>int</code>	number
<code>byte</code>	
<code>short</code>	
<code>long</code>	
<code>float</code>	
<code>double</code>	string
<code>char</code>	
<code>string</code>	boolean
<code>boolean</code>	

Bir değişkenin tipini anlamak için `typeof()` komutu kullanılır. `typeof()` komutunu daha detaylı öğrenmek için aşağıdaki örneği inceleyelim.

```
<script type="text/JavaScript">
  var a = "yesil vadi";
  var b = 3.14;
  var c = 'e';
  var d = true;
  var e = 1453;
  document.write(typeof(a) + "<BR>");
  document.write(typeof(b) + "<BR>");
  document.write(typeof(c) + "<BR>");
  document.write(typeof(d) + "<BR>");
  document.write(typeof(e) + "<BR>");
</script>
```

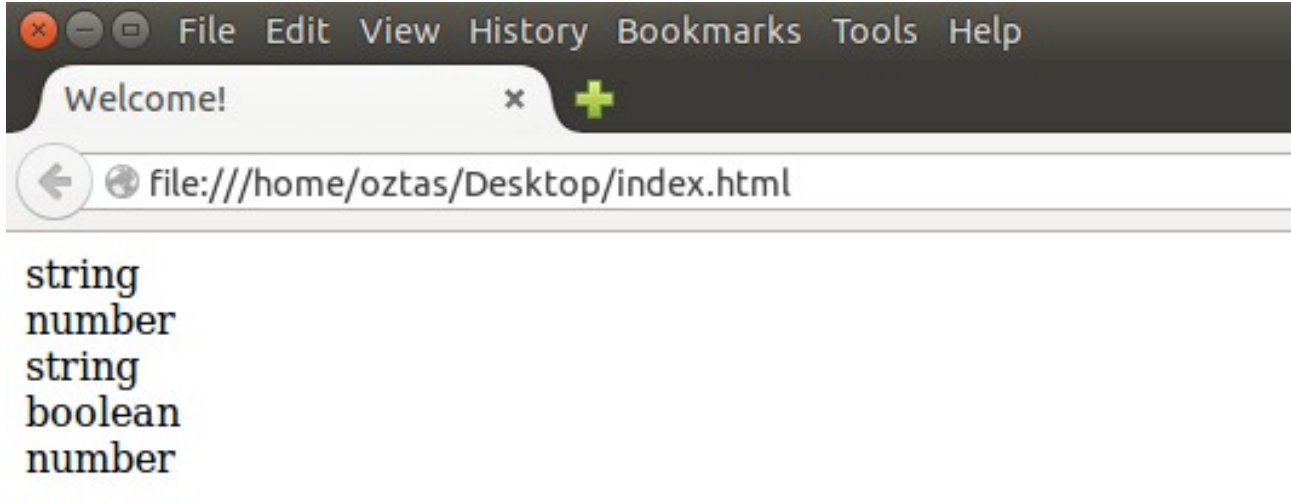
Yukarıdaki örneğimizde `typeof()` komutunun çıktısı bir değişkene de atanabilirdik fakat biz doğrudan yazdırdık. Değişkenlerimize çeşitli değerleri atadık ve `typeof()` komutuyla tipleri nelermiş öğrenmek için ekrana yazdırma işlemi yaptık.



`typeof(degiskenAdi)` komutu `typeof degiskenAdi` şeklinde de kullanılabilir. Daha açıklayıcısı olması için:

```
var toplamDeger = 639;  
typeof(toplamDeger);  
// veya  
typeof toplamDeger;
```

Yukarıdaki örneğimize ilişkin ekran çıktısını alalım.



Yukarıdaki ekran alıntısında görüldüğü gibi JavaScript'te veri tipleri 3 grupta toplanmıştır.

Öte yandan bahsetmiş olduğumuz diğer programlama dillerinde bulunan veri tipleri de `Reserved Words` (Ayrılmış Kelimeler) tablosunda bulunmaktadır.

7.2 Type Casting (Tip Dönüşümü)

Type Casting (Tip Dönüşümü), farklı veri tipindeki değişkenlerin birbirleri arasında dönüşümüdür. Yalnız dönüşüm yapılacak olan verilerin benzer nitelikte olmaları gerekir. Örneğin aşağıdaki gibi bir yapımız olsun.

```
<script type="text/JavaScript">  
    var degisken = "4206";  
</script>
```

Yukarıdaki örneğimizde `degisken` adındaki değişkenimiz `String` bir yapıda olmasına rağmen bir sayısal ifade içermektedir. Böyle durumlarla sıkça karşılaşabiliriz ki hatırlarsanız prompt ile aldığımız değerler de metinsel bir yapıdaydı. O zaman da tip dönüşümü yapmamız gerekir.

JavaScript'te 3 tip dönüşümü vardır. `float`, `int` ve `String` veri türleri arasında dönüşüm yapılabilir. Burada belirtmekte yarar var. `String` bir değişken sayısal ifade içeriyorsa dönüşümü yapılabilir, metinsel ifade içeriyorsa sonuç `NaN` olur. Bir diğer konuda `float` tipindeki kayan noktalı bir sayıyı `int` tipine çevirmek isterseniz; virgülden sonra ki sayılar kaybolur.

Diğer önemli bir konudan bahsetmek istiyorum. Bir önceki bölümde `prompt` komutunda kullanıcıdan alınan değer `String` tipinde olduğundan bahsetmiştik ve `prompt` ile kullanıcıdan değer aldıktan sonra kullanılacak tipe göre tip dönüşümünün yapılması gerekliliğinden bahsetmiştik. Şimdi bu konuya daha detaylı olarak bakalım.

`prompt` komutuyla kullanıcıdan bir değer alalım. Bu değer `int`, `float`, `String` ve `boolean` olması önemli değil. Aldığımız bu değeri `typeof()` komutuyla tipine bakalım.

```
<script type="text/JavaScript">
    var abc = prompt("Bir değer giriniz: ");
    var tip = typeof abc;
    document.write(tip);
</script>
```

Yukarıdaki örnek kodlarımızı çalıştırdığınızda, hangi değeri girerseniz girin tip olarak `String` dönecektir.

7.2.1 `parseInt()`

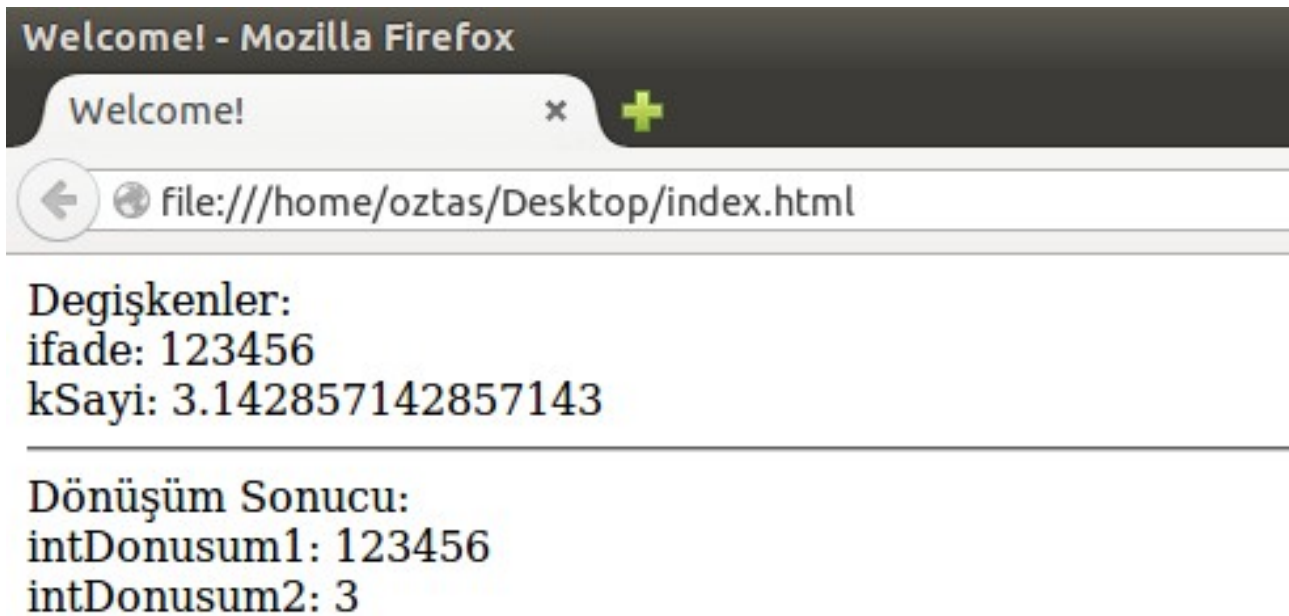
`parseInt()` komutu `float` veya `String` bir yapıda ki değer `int` tipine çevrilmesini sağlar. `parseInt()` komutunun standart kullanımı aşağıdaki gibidir.

```
// normal gosterim sekli.
parseInt(cevrilecek_deger);
// cikan sonuc bir degiskene baglanirsa deger kaybolmamis olur.
var de = parseInt(cevrilecek_deger);
```

`parseInt()` komutuyla ilgili aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var ifade = "123456";
    var kSayi = 3.142857142857143;
    // int tipine donusturuluyor...
    var intDonusum1 = parseInt(ifade);
    var intDonusum2 = parseInt(kSayi);
</script>
```

Yukarıdaki örneğimize daha yakından bakmak için; `document.write()` ekrana yazdırma komutunu ekleyelim ve `intDonusum1` ve `intDonusum2` değişkenlerinin durumlarına bakalım.



Yukarıdaki ekran alıntısını incelediğimiz de daha önce değindiğimiz bazı konuların ekrana yansıdığını gördük. Buradan çıkacak olan ders; `float` veri tipini `int` tipine çevirirken virgülden sonrası kesilip atılır. Yani bir değer kaybı söz konusudur. Bu konuda dikkatli olalım.

7.2.2 `parseFloat()`

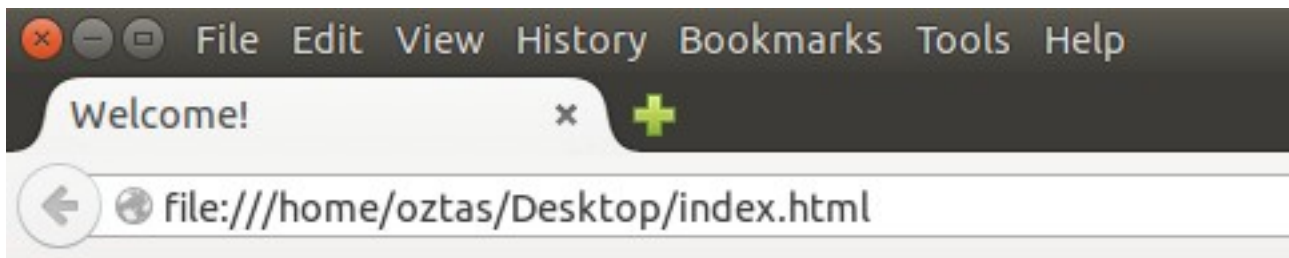
`parseFloat()` komutuyla `int` ve `String` tipteki değerler `float` veri tipine çevrilebilir. `int` veri tipinden ziyade kayan noktalı sayılar içeren `String` veri tipindeki değerlerin, veri kaybına uğramadan çevrilebilmesi noktasında önemli bir komuttur. `parseFloat()` komutunun standart kullanımı aşağıdaki gibidir.

```
// normal gosterim sekli.  
parseFloat(cevrilecek_deger);  
// cikan sonuc bir degiskene baglanirsa deger kaybolmamıs olur.  
var de = parseFloat(cevrilecek_deger);
```

Her zaman olduğu gibi yine bir örnekle bu komutumuzu açıklamaya çalışalım.

```
<script type="text/JavaScript">  
    var ab = "3.142857142857143 ";  
    var cb = parseFloat(ab);  
</script>
```

Yukarıdaki örneğimizin ekran çıktısını alalım. Ekran çıktısı almak için `document.write(cb)` komutunu eklememiz gerektiğini unutmayalım.



3.142857142857143

Yukarıdaki ekran alıntısında görüldüğü gibi; String tipindeki değerimizi float tipine herhangi bir veri kaybı olmadan çevirdik. Dilerseniz `typeof(cb)` komutu ile tip dönüşümünde kullandığımız değişkenimizin tipini alabiliriz.

7.2.3 toString()

`toString()` komutu belirtilen değeri `String` tipine çevirmektedir. Örneğin elimizde bulunan `float` veya `int` tipindeki değerleri String'e çevirmek isteyebiliriz. Böyle durumlarda `toString()` komutu kullanılır. `toString()` komutunun standart kullanımı aşağıdaki gibidir.

```
degiskenAdi.toString();
```

Her zaman olduğu gibi yine bir örnek üzerinden konumuzu anlatmaya çalışalım.

```
<script type="text/JavaScript">
    var sayi = 2015;
    var cevir = sayi.toString();
</script>
```

Yukarıdaki örneğimizde `int` tipinde bir tam sayı değişken olan `sayi` değişkenini `toString()` komutunu kullanarak String'e çevirdik. Eğer `typeof(cevir)` komutunu kodlarımız arasında ekleyip ekran çıktısı alırsak, `int` tipindeki değişkenimizin değer String tipine çevrilmiş olduğunu görebiliriz.

BÖLÜM 8:

Control Statement (Kontrol Yapıları)

Kontrol yapıları; program akışı sırasında bazı verilerin belli kısıtlara göre kontrol edilmesi ve daha sonra bu kontrolden geçen bilgilere, belli işlemler uygulamak için kullanılan yapılardır. Örneğin herhangi bir x sitesine üye olurken sizden alınan bilgiler belli kriterlere göre değerlendirilir ve verdiğiniz bilgiler yetersiz veya uygun değilse sizden tekrar giriş ister. İşte bu gibi durumları kontrol eden yapılar; web sayfalarında genellikle JavaScript ortamında, control statement (kontrol yapıları) ile sağlanır.

Çoğu programlama dilinde olduğu JavaScript ortamında da iki adet kontrol yapısı vardır. Bunlar: if -else ve switch - case yapılarıdır.

8.1 if – else

if, Türkçe'deki anlamıyla (“eğer”) aynı işi yapıyor desek yeridir. Çünkü if ile verilen ifadeler veya veriler belli kısıtlara göre kontrol edilir. Yani; eğer sonuç bu ise şunu yap gibi bir yapıdır. Bu yapıda nasıl bir kontrol kullanmış isek veriler ona göre kontrol yapılır ve yapılan bu sonuca göre; if'in scope alanı içerisinde yani etki alanı içerisindeki işlemler yürütülür. Eğer yapılan işlem sonucu if ifadesinde belirtilen duruma uygun değilse else yapısına uyar demektir ve else'nin scope alanı içerisindeki işlemler yürütülür. If'in standart kullanımı aşağıdaki gibidir.

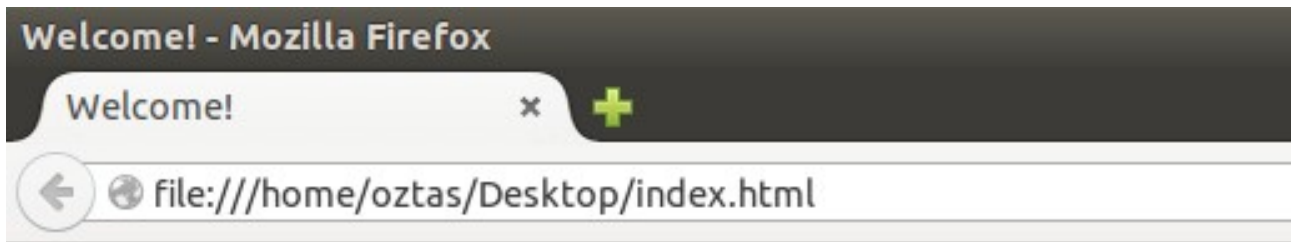
```
<script type="text/JavaScript">
    if (kosul) {
        // yurutulecek islemler.
    }else{
        // yurutulecek islemler.
    }
</script>
```

If – else yapısını kullanarak bir örnek yapalım. Bu örneğimizde; prompt konusunda yaptığımız, örneği kullanalım. Kullanıcıdan vize ve final notlarını yine prompt komutuyla alalım, tip dönüşümü yapalım ve kullanıcının geçme / kalma durumunu öğrenelim. Yapacağımız örnek aşağıdaki gibi olacaktır.

```
<script type="text/JavaScript">
    // kullanicidan degerleri alalım.
    var vize = prompt("Vize notunuz:");
    var fnal = prompt("Final notunuz:");
    // tip donusumu yapalım.
    vize = parseInt(vize);
    fnal = parseInt(fnal);
    // girilen notları hesaplayalım.
    var sonuc = (vize * 0.6) + (fnal * 0.4);
    // degerleri ekrana yazdıralım
    document.write("Vize: " + vize + " ");
    document.write("Final: " + fnal + " ");
    document.write("Sonuç: " + sonuc + " ");
    /*
    gecti/kaldi durumunu if-else ile kontrol edelim
```

```
ve durumu ekrana yazdıralım
*/
if(sonuc < 60){
    document.write("Kaldınız!");
}else {
    document.write("Geçtiniz!");
}
</script>
```

Yukarıdaki kodlarımızın üstlerine, açıklama satırları ile hangi işlemleri yaptığımızı yazdık. If – else yapısıyla sonuc değişkenini kontrol ettik. Eğer sonuc değişkenindeki değer 60'tan küçük ise “Kaldı” değerini yazacak, aksi durumda yani sonuc değişkenindeki değer 60'tan büyükse “Geçtiniz!” yazacaktır. JavaScript programımızı çalıştıralım. Vize ve final notlarımızı girelim. Sonucuna bakalım doğru hesaplama yapıyor mu öğrenelim.



Vize: 52 Final: 65 Sonuç: 57.2 Kaldınız!

Yukarıdaki ekran alıntısında görüldüğü gibi; dersten kalmışım!

If – else yapısının çalışması hakkında biraz konuşalım. if (kosul) ifadesinde koşul sağlanıyorsa if scope alanındaki işlemler gerçekleştirilir aksi halde else scope alanındaki işlemler gerçekleştirilir demistik. if(kosul) ifadesinde herhangi bir değere göre kontrol yapabilirsiniz. Buradan dönecek değer boolean bir değerdir. Yani if(kosul) = boolean bir değerdir. Eğer dönen boolean değer true ise if scope alanındaki değerler işletilir. Dönen değer false ise else scope alanında ki değerler işletilir. Bunu unutmayalım.



Konunun başından beri hep scope alanı (etki alanı) olarak bahsettik. Peki scope alanı nedir? Akıllarda bu soru oluşabileceği için açıklama yapalım istedim. Scope alanı, etki alanı veya blok olarak adlandırılan bir kavramdır ve { } küme parantezi (güzel parantez, kıvrıkcık parantez, süslü parantez artık nasıl adlandırıyorsanız) ile sağlanır. { } parantezleriyle oluşturulan her alan scope alanıdır. Burası ayrı bir dünyadır ve dış dünyayla (scope alanı dışındaki alan) pek bir alakası yoktur. Ne vererseniz o yürütülür.



Eğer if – else yapısından sonra yürütülecek işlemler tek satırlık ise küme parantezleri açıpta scope alanı oluşturmaya gerek yoktur. Aşağıdaki örnekte olduğu gibi.

```
var sonuc = 42;
```

```
if (sonuc < 60) document.write("Kaldı!");  
else document.write("Geçti!");
```

If – else yapısı sade iki değer için kontrol yapar. Yukarıdaki örneğimizde olduğu iki geçti / kaldı veya doğru / yanlış gibi. Peki daha çok kontrol yapısı oluşturmak istiyorsak o zaman ne yapmamız lazım? Paniğe gerek yok. If – else if – else yapısı bu gibi durumlar için oluşturulmuştur. If – else if – else yapısının standart kullanımı aşağıdaki gibidir.

```
if(kosul1){  
    //yurutulecek islemler  
} else if(kosul2){  
    //yurutulecek islemler  
} else if(kosul3){  
    //yurutulecek islemler  
}  
...  
...  
...  
  
else {  
    //yurutulecek islemler  
}
```

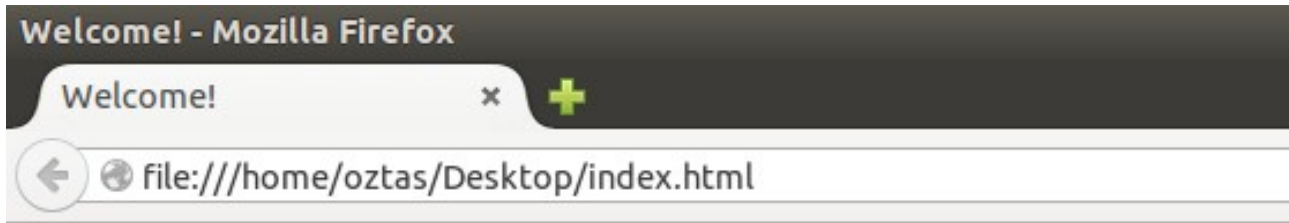
Son satırdaki else yapısı geriye kalan tüm değerleri kapsamak içindir.

If – else if – else yapısını bir örnekle daha iyi kavramaya çalışalım. Bu örneğimizde; yukarıdaki örneğe harf notu da ekleyelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<script type="text/JavaScript">  
    var vize = prompt("Vize notunuz:");  
    var fnal = prompt("Final notunuz:");  
    vize = parseInt(vize);  
    fnal = parseInt(fnal);  
    var sonuc = (vize * 0.6) + (fnal * 0.4);  
    document.write("Vize: " + vize + " ");  
    document.write("Final: " + fnal + " ");  
    document.write("Sonuç: " + sonuc + " ");  
    if(sonuc < 25){  
        document.write("FF");  
    }else if (sonuc >= 25 && sonuc < 30){  
        document.write("FD");  
    } else if (sonuc >= 30 && sonuc < 35){  
        document.write("DD");  
    } else if (sonuc >= 35 && sonuc < 45){  
        document.write("DC");  
    } else if (sonuc >=45 && sonuc < 55){  
        document.write("CC");  
    } else if (sonuc >= 55 && sonuc < 65){  
        document.write("CB");  
    } else if (sonuc >=65 && sonuc < 75){  
        document.write("BB");  
    } else if(sonuc >=75 && sonuc < 85){  
        document.write("BA");  
    } else{  
        document.write("AA");  
    }  
</script>
```

If ile kontrol ettiğimiz koşul sağlanmıyorsa, else – if yapısının koşuluna bakılır. Eğer ilk else – if yapısının koşulu da sağlanmıyorsa kendisinden sonra gelen else – if yapısının koşuluna bakılır ve bu böyle devam eder. Eğer if ve else – if yapılarından herhangi birisinin koşulu sağlanmıyor ise son olarak else yapısına gidilir ve else bloğunda hangi işlem varsa o yürütülür. Yani else bloğu son bakılan yerdir.

If – else yapısında, notlarımı hesaplamıştım ve dersten kalmıştım. Bir de harf notumu alıyım ve durumuma bir kez daha bakayım. Belki de geçmişimdir.



Vize: 52 Final: 65 Sonuç: 57.2 CB

Yukarıdaki ekran alıntısında görüldüğü gibi dersten geçmişim hemde CB ile!

8.2 Switch – Case

switch – case yapısı, if – else yapısıyla aynı kategoridedir ve ikisi işi yapmaktadır. Switch ile değer alınır ve case yapılarıyla da alınan bu durum kontrol edilir. Switch – case yapısına ait standart kullanım aşağıdaki gibidir.

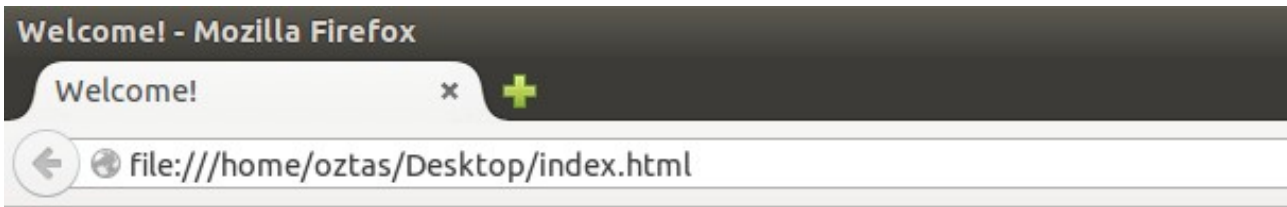
```
switch(kontrol_edilecek_deger){
    case kosul1:
        // yurutulecek islemler
        break;
    case kosul2:
        // yurutulecek islemler
        break;
    case kosul3:
        // yurutulecek islemler
        break;
    . . .
    . . .
    . . .
    default:
        // yurutulecek islemler
}
```

Yukarıdaki standart kullanımı verilmiş olan switch – case yapısı hakkında biraz konuşalım. switch() ifadesinde; parantezler içerisine kontrolü yapılacak olan değer alınır. case yapıları ile bu alınan değer kontrol edilir. Eğer koşul sağlanıyorsa yapılacak işlemler yürütülür. Aksi halde diğer case kontrolüne bakılır. Bu iki case arasındaki geçişler break ile sağlanır. Eğer break koyulmaz ise case kontrolündeki koşul sağlansın veya sağlanmasın sağlanmış gibi gösterilir ve o işlemler yürütülür. Her case kontrolünden sonra break kullanılmıyorsa kontrollerimiz yanlış çalışır.

Her zaman olduğu gibi yine örnek bir program yazalım. Bu örneğimizde kullanıcıdan 1 – 10 arası bir değer alalım ve bu değerın metinsel karşılığı ekrana yazalım.

```
<script type="text/JavaScript">
    var sayi = prompt("1 - 10 arasında bir değer giriniz.");
    sayi = parseInt(sayi);
    document.write("Girilen sayi: " + sayi + ". ");
    switch(sayi){
        case 1:
            document.write("Bir");
            break;
        case 2:
            document.write("İki");
            break;
        case 3:
            document.write("Üç");
            break;
        case 4:
            document.write("Dört");
            break;
        case 5:
            document.write("Beş");
            break;
        case 6:
            document.write("Altı");
            break;
        case 7:
            document.write("Yedi");
            break;
        case 8:
            document.write("Sekiz");
            break;
        case 9:
            document.write("Dokuz");
            break;
        case 10:
            document.write("On");
        default:
            document.write("Hileci! 1 - 10 arası bir sayı girmedin!");
    }
</script>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Girilen sayı: 25. Hileci! 1 - 10 arası bir sayı girmedin!

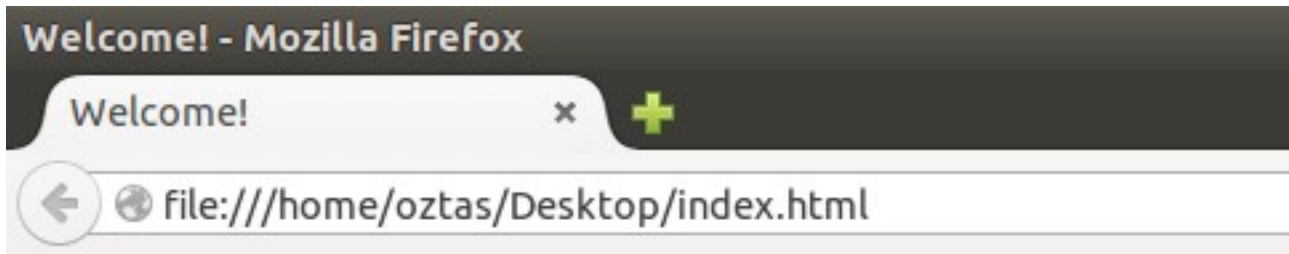
Yukarıdaki ekran çıktısında; kodlarımın ne kadar asi olduğunu görüyorsunuz.

Son bir örnek ile switch – case yapısını bitirelim. Aşağıdaki örneğimizi dikkatle inceleyelim.

```
<script type="text/JavaScript">
    var sehir = prompt("Şehrinizi giriniz: ");
    switch(sehir){
        case "KONYA":
        case "konya":
            document.write("Girilen il: " + sehir + ". ");
            document.write("Plaka Kodu: " + 42);
            break;
        case "İSTANBUL":
        case "istanbul":
            document.write("Girilen il: " + sehir + ". ");
            document.write("Plaka Kodu: " + 34);
            break;
        case "İZMİR":
        case "izmir":
            document.write("Girilen il: " + sehir + ". ");
            document.write("Plaka Kodu: " + 35);
            break;
        case "ANKARA":
        case "ankara":
            document.write("Girilen il: " + sehir + ". ");
            document.write("Plaka Kodu: " + 06);
            break;
        default:
            document.write("Girilen il: " + sehir + ". ");
            document.write("Girilen il sistemde kayıtlı değildir!");
    }
</script>
```

Yukarıdaki örneğimizde göstermek istediğim iki kavram var. bunlardan birincisi; switch – case yapısı sadece sayısal bir değeri değil String yani metinsel bir ifadeyi de kontrol edebilmektedir. Aynı zamanda boolean bir değeri de kontrol edebilir. Orası ayrı mesele. İkinci göstermek istediğim şey ise case yapılarını art arda kullanarak birden fazla kontrol yaptırabilirsiniz. Mesela biz örneğimizde iki tane aynı durumu kontrol ettik. Kullanıcı büyük harfte girebilir, küçük harfte girebilir diye varsaydık. Eğer birden fazla benzer yapıda kontrol edilecek durum var ise case yapılarını yukarıdaki örneğimizde olduğu gibi kullanabilirsiniz. Buna dikkat edelim.

Örneğimize ait ekran çıktısını alalım.



Girilen il: konya. Plaka Kodu: 42

BÖLÜM 9:

Loops ve break & continue Deyimleri

Loops (döngüler), birden fazla yapılacak olan aynı işi gerçekleştirmek üzere oluşturulmuş yapılardır. Örneğin ilkokuldayken öğretmenlerimiz bizlere üzerinde kelimelerin yazılı olduğu fişleri verirler ve on kere, yüz kere defterimize yazmamızı isterlerdi. Bu tabi bizim hem yazma hem de öğrenme kabiliyetimizi geliştirmek için yapılan bir şeydi. Programlama hayatımızda böyle olmasa da buna benzer durumlar ortaya çıkabiliyor.

JavaScript'te 3 tip döngü yapısı vardır. Bunlar: `for`, `while` ve `do...while` idir. Bir de `for` döngü yapısının daha kısa formu olan `enhanced for` ya da `for...in` olarak adlandırabileceğimiz bir döngü yapısı daha var. Bu yapıyı bu bölümde değil, Arrays (Diziler) bölümünde inceleyeceğiz.

9.1 for Döngüsü

Her döngünün yapısı birbirinden farklıdır ve aynı zamanda her döngü, her döngüye ihtiyaç olunan yerde kullanılacak diye bir şey söz konusu değildir. For döngüsünün, programlama dünyasında yeri diğer döngülerden ayrı olup daha çok kullanılıyor desek herhalde yanlış olmaz. For döngüsünün standart kullanımı aşağıdaki gibidir.

```
<script type="text/JavaScript">
    for(baslangic_degeri; sonlanma_kriteri; dongu_degerini_duzenle){
        // yapılacak islemler
    }
</script>
```

Yukarıdaki standart formunu vermiş olduğumuz `for` yapısını daha yakından inceleyelim. For yapısının 3 bölümü vardır ve bu bölümler birbirleriyle ilişkilidir.

```
for(baslangic_degeri; sonlanma_kriteri; dongu_degerini_duzenle) {}
```

For yapısının döngüye başlayabilmesi için bir başlangıç değeri olmak zorundadır. Bu başlangıç değerine taban değeri ve for yapısına yukarıya doğru çıkan bir asansör diyebiliriz. Döngüye başlangıç değerini `baslangic_degeri` olarak adlandırdığımız alanda vermek zorundayız. Bu döngü başlangıç değeri for yapısının parantezleri içerisinde tanımlanabileceği gibi for yapısının dışında da tanımlanabilir. Aşağıda olduğu gibi.

```
var i = 1;
for(i; sonlanma_kriteri; dongu_degerini_duzenle) {}
//veya
for(var i = 1; sonlanma_kriteri; dongu_degerini_duzenle) {}
```

Her döngüsünün bir sonlanma kriteri bulunmak zorundadır aksi halde sonsuz bir döngü olur. Sonlanma kriteri `sonlanma_kriteri` olarak belirttiğimiz alana yazılmak zorundadır. Bu alanda bir kontrol ifadesi olmalıdır. Döngü bu kritere ulaştığı zaman otomatik döngüden çıkılır.

```
for(var i = 0; i < 10; dongu_degerini_duzenle) {}
```

Bu başlangıç değerinin update (güncelleme) edilmesi gerekir. Döngünün kuruluş yapısına göre arttırma, azaltma, çarpma, bölme işlemi yapılabilir. Bu güncelleme işlemi döngünün başlangıç değeri üzerinde gerçekleştirilir.

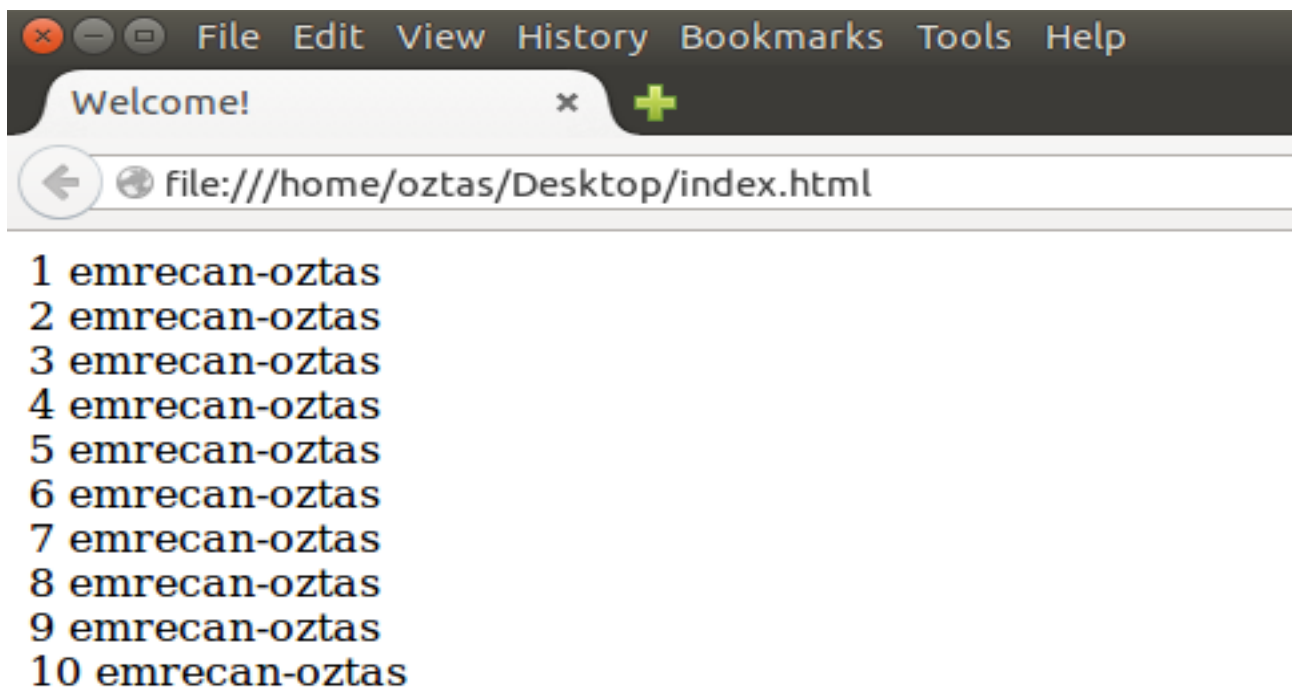
```
for(var i = 0; i < 10; i++) {}
```

For döngüsü doğru bir şekilde kurulduktan sonra for döngüsüne ait scope alanı içerisinde yapılmak istenen işlemler yazılır.

Verdiğimiz bu bilgiler ışığında örnek bir uygulama yapalım. Bu uygulamamızda ekrana 10 kere ismimizi yazdıralım. Tabi herkes kendi ismini yazdırsın.

```
<script type="text/JavaScript">
    for(var i = 1; i <=10; i++){
        document.write(i + " emrecan-oztas" + "<BR>");
    }
</script>
```

Programımız yazdık. Browser'da çıktımıza bakalım.



Görüldüğü gibi programımız doğru çalışmaktadır. Döngümüzü tersten de kurabiliriz. Aşağıda olduğu gibi.

```
<script type="text/JavaScript">
    for(var i = 10; i >= 1; i--){
        document.write(i + " emrecan-oztas" + "<BR>");
    }
</script>
```

Döngüyü nasıl kuracağınız tamamen sizin inisiyatifinize kalmış durumdadır. for yapısıyla pek çok değişik döngü oluşturabilirsiniz.



Eğer for sope alanı içerisinde yapılacak işlem tek satırlık bir işlem ise o zaman scope alanlarını açmaya gerek yoktur. Aşağıdaki yapıda olduğu gibi.

```
for(var i = 1; i <= 10; i++) document.write("emrecan-oztas");
```

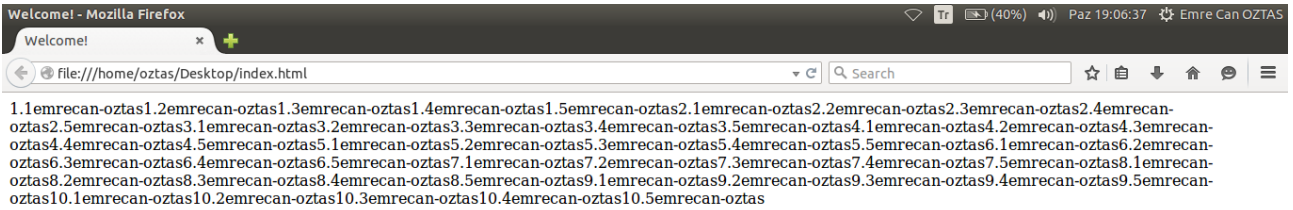
for yapılarını iç içe kullanarak döngüler oluşturabiliriz. Her for döngüsünün scope alanına istenildiği kadar iç içe for döngüsü oluşturulabilir. Bu yapıya Nested for yapısı denilmektedir. Nested for yapısının kullanımı aşağıdaki gibidir.

```
for(var i = 1; i <= 10; i++){
    for(var j = 1; j <= 5; j++){
        // yapılacaklar
        // for yapıları acılabilir
    }
}
```

Yukarıdaki yapıyı kullanarak bir örnek yazalım. Tek for yapısıyla yaptığımız örneği içi içe iki for yapısıyla yeniden yazalım.

```
<script type="text/JavaScript">
    for(var i = 1; i <= 10; i++){
        for(var j = 1; j <= 5; j++){
            document.write(i + "." + j + "emrecan-oztas");
        }
    }
</script>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Yukarıdaki ekran çıktısını incelediğimizde iç içe for yapılarının çalışma mantığını anlayabiliriz. Örneğimizde ilk for yapısı 10 kez, ikinci for yapısı 5 kez çalışacaktır. Toplamda $10 \times 5 = 50$ kere çalışacaktır.

İç içe for yapılarının çalışma mantığından bahsedelim. İlk for döngüsündeki değer çalıştığı anda içteki for yapısına bakacaktır. İçteki for yapısı çalışıp, bitirme kriteri sağlandığı zaman dıştaki for yapısı yeniden

çalışmaya başlayacaktır. Örneğimizde dıştaki for yapısının birinci değerinde içteki for yapısı 5 kez çalışacaktır. Aynı şekilde dıştaki for yapısının ikinci değerinde içteki for yine 5 kez çalışacaktır ve bu dıştaki for yapısı bitene kadar bu durum böyle devam edecektir. Buradan çıkan sonuç içteki for dıştaki for yapısının her adımına bağlıdır.

Yukarıdaki örneğimizde olduğu gibi iç içe birden fazla for yapısı oluşturabiliriz ve her for yapısında ayrı işlemler gerçekleştirebiliriz. Örneğin iç içe beş for yapısı oluşturmak istersek kodlarımızı aşağıdaki gibi olacaktır.

```
<script type="text/JavaScript">
  for(var i = 1; i <= 10; i++){
    for(var j = 1; j <= 5; j++){
      for(var k = 1; k <= 20; k++){
        for(var x = 1; x < 20; x++){
          for(var y = 100; y > 0; y--){
            }
          }
        }
      }
    }
  }
</script>
```

9.2 while

while, bir diğer döngü yapısıdır. for döngüsünden farklı bir yapıdadır. while, belirtilen koşul sağlandığı anda çalışmaya başlar. Eğer belirtilen koşul sağlanmıyor ise hiç çalışmaz. Ayrıca for yapısından farklı olarak, başlangıç değerini güncelleme while'ın gövdesinde yapılır. while'ın standart kullanımı aşağıdaki gibidir.

```
<script type="text/JavaScript">
  while(kosul){
    // yapılacaklar
  }
</script>
```

while yapısını kullanarak bir örnek yapalım. Bu örneğimiz for yapısında kullandığımız örnek olsun. Bu örnek; iki döngüyü karşılaştırmamıza yardımcı olacaktır.

```
<script type="text/JavaScript">
  var i = 1;
  while(i <= 10){
    document.write("emrecan-oztas" + "<BR>");
    i++;
  }
</script>
```

Yukardaki örneğimizi inceleyelim. i adında bir değişken tanımladık. i değişkeni bizim başlangıç değerimiz olacaktır. while (i <= 10) ifadesinde; başlangıç değerimiz olan i, 10 sayısından küçük veya eşit olana kadar çalışacaktır. Bu bizim koşul ifademizdir. Yani i burada kontrol edilecektir. while'ın gövdesinde yani scope alanında ekrana ismimizi yazdırdık. Başka işlemlerde yapabiliriz. Daha sonra başlangıç değerimiz olan i değerine +1 ekledik. Eğer eklemese idik; döngümüz sonsuz bir döngü olacaktı ve biz browser'ı kapatana kadar durmadan çalışacaktı.

Başka bir örnekle while yapısını anlamaya çalışalım. Bu örneğimizde kullanıcıdan bir değer alalım. Kullanıcının girmiş olduğu bu değere kadar olan ikiye ve üçe bölünebilen sayıları bulup ekrana

yazdırmaya çalışalım. Örnek kodlarımız aşağıdaki gibi olacaktır.

```
<script type="text/JavaScript">
    var sayi = prompt("Bir sayi giriniz: ");
    sayi = parseInt(sayi);
    document.write("Girilen sayi: " + sayi + "<BR>");
    var sayac = 1;
    while(sayac <= sayi){
        if(sayac % 2 == 0){
            if(sayac % 3 == 0){
                document.write(sayac + ". ");
            }
        }
        sayac++;
    }
</script>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



“Bir sayı giriniz: ” sorusuna 100 sayısını girdim. 1'den 100'e kadar olan, 2 ve 3'e bölünebilen sayılar yukarıdaki ekran alıntısında olduğu listelenmiş durumdadır.

Yukarıdaki örneğimizde; başlangıç ve bitiş değerini kullanıcıdan alarakta, kullanıcının istediği aralıkta 2 ve 3'e bölünebilen sayıları bulabiliriz. Aşağıdaki kodlar inceleyelim.

```
<script type="text/JavaScript">
    var baslangic = prompt("Başlangıç: ");
    var bitis = prompt("Bitiş: ");
    baslangic = parseInt(baslangic);
    bitis = parseInt(bitis);
    document.write("Başlangıç: " + baslangic + "<BR>" + "Bitiş: " + bitis +
"<BR>");
    while(baslangic <= bitis){
        if(baslangic % 2 == 0){
            if(baslangic % 3 == 0){
                document.write(baslangic + ". ");
            }
        }
        baslangic++;
    }
</script>
```

9.3 do...while

do...while yapısı, while yapısıyla benzer yapıda olan bir döngü deyimidir. While konusunda, belirtilen koşul sağlandığı durumda çalışmaya başlar demiştik. Fakat do...while, belirtilen koşul sağlansa da sağlanmasa da en az 1 kez çalışır. Böyle davranmasının nedeni yapısı gereğidir. do...while yapısının standart kullanımı aşağıdaki gibidir.

```
<script type="text/JavaScript">
    do{

        // yapılacaklar

    }while(kosul);
</script>
```

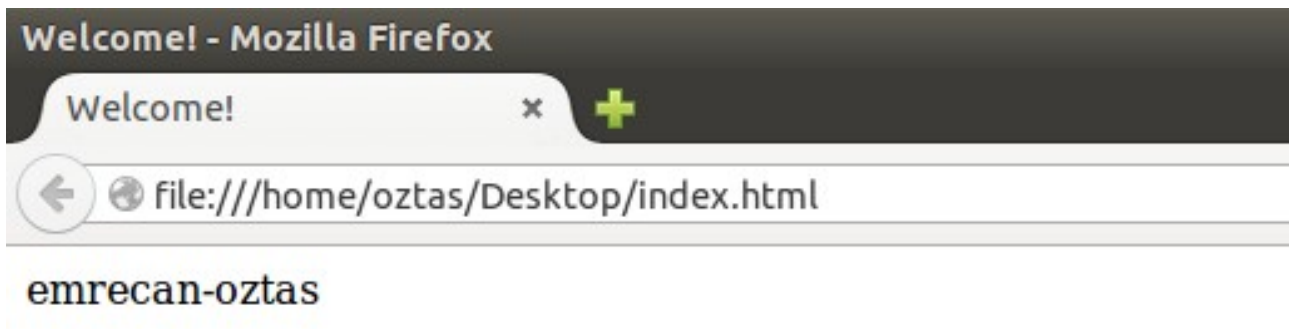
Yukarıdaki standart kullanımı gösterilen do...while yapısında görüldüğü gibi koşula en son bakılmaktadır. Başta herhangi bir koşul ifadesi olmadığı için döngüye en az 1 kez girilir. Her zaman olduğu gibi yine örnek bir uygulama üzerinden gidelim. For ve while yapısında kullandığımız örneğimizi do...while yapısında tekrar yazalım.

```
<script type="text/JavaScript">
    var i = 1;
    do{
        document.write("emrecan-oztas" + "<BR>");
        i++;
    }while(i <= 10);
</script>
```

do...while yapısında, sağlanmayan bir koşul tanımlayalım ve do...while yapısının koşulu sağlanmasa bile en az 1 kez çalıştığını kanıtlayalım.

```
<script type="text/JavaScript">
    var i = 1;
    do{
        document.write("emrecan-oztas" + "<BR>");
        i++;
    }while(i >= 10);
</script>
```

İlk yazdığımız örneğimizdeki koşulun operatörlerini tersine çevirdik. Yukarıdaki ikinci örneğimizi browser'da görüntülediğimizde sadece tek bir değer göreceğiz. Aşağıdaki ekran çıktısında olduğu gibi.



9.4 break & continue Deyimleri

Buraya kadar olan döngüler kısmında hep döngüyü bitirme kriterinin kontrol edilmesi gerektiğinden aksi halde sonsuz döngüye girileceğinden bahsettik. Peki döngüyü istediğimiz zaman bitiremez miyiz? Evet, bitirebiliriz. Döngüyü istediğimiz zaman kırabilir ve döngüden çıkabiliriz. Bu işlem için `break` deyimini kullanırız. Öte yandan bazı durumlarda da bazı değerlerin görmezden gelinmesini veya atlatılmasını isteyebiliriz. Böyle durumlarda da `continue` deyimini kullanırız.

9.4.1 break

Yukarıda bazı açıklamalarda bulunduk. `break` deyimi döngüyü kırmak veya döngüden çıkmak için kullanılan bir deyimdir. Şöyle düşünelim. Bazı adımlarda kullanıcıdan bilgiler alabilir ve kullanıcı devam etmek istemediği durumlarda döngünün kırılması gerekebilir. İşte böyle durumlarda `break` deyimini kullanırız. `break` deyimini üç döngü deyiminde de kullanabiliriz.

Daha akılda kalıcı olması için bir örnek yapalım. Örneğin kullanıcıdan bir takım bilgiler isteyelim ve kullanıcı devam etmek istediği durumda döngüyü devam ettirelim. Kullanıcı devam etmek istemediği durumda döngüyü kuralım. Örneğimiz aşağıdaki gibi olacaktır.

```
<script type="text/JavaScript">
    var durum = true;
    var sayac = 0;
    while(durum){
        sayac++;
        var adSoyad = prompt("Öğrenci adı:");
        var vize = prompt("Vize: ");
        var fnal = prompt("Final: ");

        vize = parseInt(vize);
        fnal = parseInt(fnal);

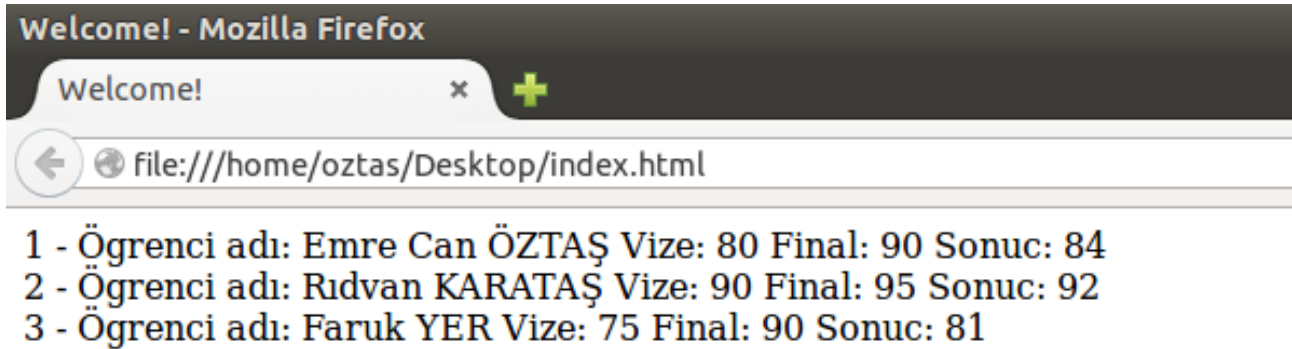
        document.write(sayac + " - ");
        document.write("Öğrenci adı: " + adSoyad + " ");
        document.write("Vize: " + vize + " ");
        document.write("Final: " + fnal + " ");

        var sonuc = (vize * 0.6) + (fnal * 0.4);

        document.write("Sonuc: " + sonuc + "<BR>");
```

```
        var cikis = confirm("Devam etmek ister misiniz?");
        if(cikis == false) break;
    }
</script>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Örneğimize geri dönelim. Sırasıyla; prompt komutuyla kullanıcıdan değerleri aldık ve sonuç notunu hesaplattık. Aldığımız ve hesapladığımız değerleri ekrana yazdık. Daha sonra confirm komutuyla kullanıcıya devam etmek isteyip istemediğini sorduk. Kullanıcı devam etmek istemediği durumda break deyimiyle döngüyü bitirdik.

Yukarıdaki kodlarımızın hemen hepsini anlayabilirsiniz, buraya kadar anlattıklarımızla. Bir yerde takılabilirsiniz. O da; while (durum) ifadesi. Daha önce if yapısında if'in parantezlerinin içinin boolean yapıda olduğundan bahsetmiştik. Bu durum döngü ifadeleri için geçerlidir. Yani while(durum) ibaresi bir boolean ifadedir. durum değişkeni true olduğu sürece döngümüz çalışır. Bizim örneğimizde de durum ifadesi true'dur.

Döngülerin başında bir şey söylemiştim. Her döngü ifadesi her yerde kullanılmaz diye. Eğer bu örneğimizde for yapısını kullansaydık; döngünün başlangıç ve bitiş değerlerini girmemiz gerekirdi. Yine kullanıcıdan devam edip etmeyeceğini sorabilirdik fakat döngü aralığını belirlediğimiz için o aralıklarda sormak zorunda kalırdık. Sonu belli olmayan döngü durumlarında; while ve do...while kullanılır. Buna dikkat edelim.

while ve do...while döngüleri birbirine benzer olduğu için ki aralarında ki tek fark do...while döngüsünün koşul sağlanmasa da 1 kez çalıştığıdır, do...while ile örnek yazmıyorum. Break deyimi için bir de for ile yapı kuralım.

Örneğin elimizde; 1 – 10 arasındaki sayıları ekrana yazdıran bir if döngüsü olsun. Döngü değeri 7 olunca döngüden çıkalım. Örnek kodlarımız aşağıdaki gibi olacaktır.

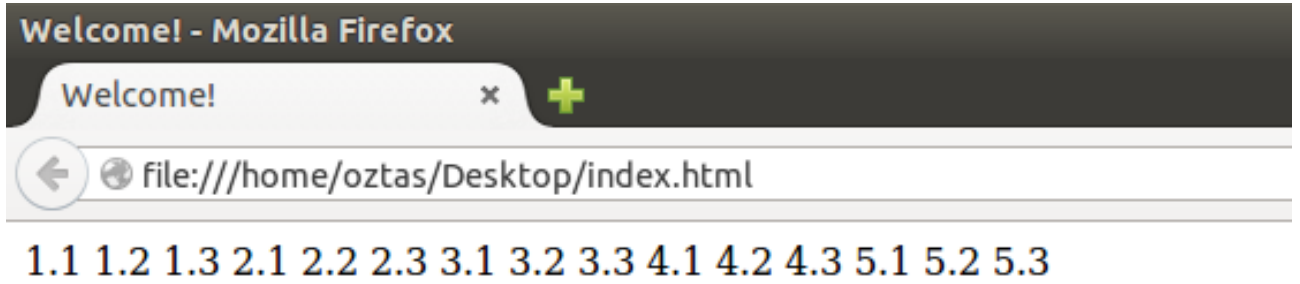
```
<script type="text/JavaScript">
    for(var i = 1; i <= 10; i++){
        document.write(i + ".");
        if(i == 7) break;
    }
</script>
```

Yukarıdaki kodlarımız basit olarak döngü değişkeni 7 olunca döngüden çıkmak üzerine kurulmuştur. Daha önce de belirttiğimiz gibi break deyimi üç döngüde de kullanımı aynıdır. Sadece döngüyü nerde ne zaman kırmamız gerektiğini bilmeniz yeterlidir.

For yapısıyla ilgili daha önemli bir konudan bahsedelim. İç içe kullanılmış for yapılarında içteki for yapısında kullandığımız break deyimiyle tüm döngülerden çıkar mıyız? Yoksa sadece içteki döngüden mi çıkarız? İçteki for yapısında kullanılan break deyimi sadece içinde bulunduğu döngüyü kırabilir. Dışta döngü içteki döngünün break deyimiyle kırılmaz. Bunu bir örnek ile açıklamaya çalışalım.

```
<script type="text/JavaScript">
  for(var i = 1; i <= 5; i++){
    for(var j = 1; j <=5; j++){
      document.write(i + "." + j + " ");
      if(j == 3) break;
    }
  }
</script>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Örneğimizde dıştaki döngü 5 kez, içteki döngü de 5 kez çalışacaktı. İki döngü basit bir hesaplama; $5 \times 5 = 25$ kez çalışacak. Yukarıdaki ekran çıktısını incelediğimizde 15 kez çalıştığı görüyoruz. Bunun nedeni break deyimidir. İçteki for döngüsünde if ile döngü değerini her adımda kontrol edip, değer 3 olduğu zaman döngüyü break deyimi ile kıldık. Gördüğünüz gibi iç içe döngü ifadelerinde; break deyimi sadece kullanıldığı döngüyü kırabilir.

Son söz olarak şunları söylemek isterim size. Break deyimini gerekli olmadığı sürece kullanmayın hatta hiç kullanmayın desem yeridir. Bazen başınızı fena halde ağrıtabilir.

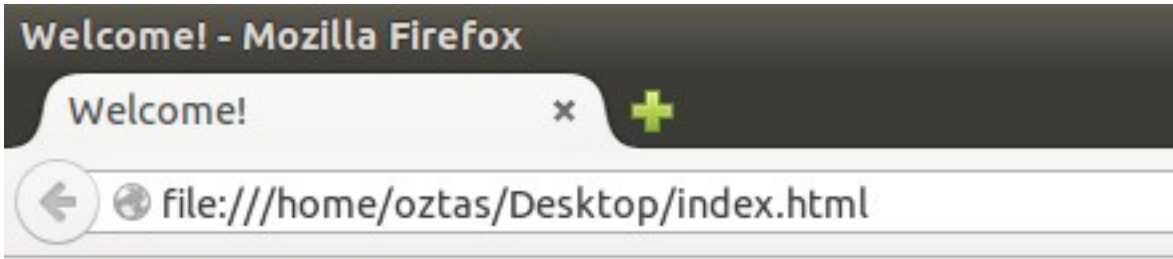
9.4.2 continue

continue deyimi bazı durumlarda bazı değerlerin görmezden gelinmesi veya atlatılması gibi durumlar kullanılır. Continue belirtilen değere gelindiği zaman o değer atlatılmasını ve döngünün kaldığı yerden devam etmesini sağlar. Aşağıdaki basit örneğimizi incelediğinizde ne demek istediğimi anlayacaksınız.

```
<script type="text/JavaScript">
  for(var i = 1; i <= 10; i++){
    if((i % 3) == 0) continue;
    document.write(i + ", ");
  }
</script>
```

Yukarıda yazmış olduğumuz örneğimizde; 3 ve 3'ün katlarını continue deyimiyle atlادık. Yani görmezden gelerek döngünün devam etmesini sağladık.

Yukarıdaki örneğimize ait ekran çıktısını alalım.



1, 2, 4, 5, 7, 8, 10,

Ekran alıntısında görüldüğü üzere; 3, 6 ve 9 sayıları atlanmış ve döngüye devam edilmiştir.

BÖLÜM 10:

Arrays ve for...in Yapısı

10.1 Arrays

Arrays (Diziler) konusunu, bu bölümde detaylı olarak incelemeye çalışacağız. Öncelikle bir dizi nedir ne işe yarar bundan bahsedelim. Dizi dediğimizde Türk mantığı olarak aklımıza hemen TV'de çıkan diziler gelebilir, normaldir. Fakat programlama dünyasında diziler bambaşka bir konudur.

Değişken yapısını bir kap gibi düşünelim demiştik. Bu kabımız sadece 1 tane değeri taşıyabiliyordu. Diziyi ise bir kitaplık gibi düşünebiliriz. Kitaplıkta aynı türden kitaplar sıralı olarak yan yana bulunur. Dizilerde bu mantıkta çalışır. Sıralı olarak art arda dizilmiş aynı türdeki veriler bir dizide saklanırlar. Daha sonra ihtiyaç olduğu anda istenilen dizi elemanı alınıp kullanılabilir. Dizilerin standart kullanımı aşağıdaki gibidir.

```
var dizi_adi = new Array(dizi_boyutu);
```



Arrays (Diziler)'de başlangıç indisi (adres i) 0'dır. Yani diziyeye değerler 0 nolu indisten başlayarak yerleştirilir.

Yukarıdaki standart gösterimi verilmiş olan dizinin başta boyutu vermek istersek; `dizi_boyutu` olarak belirtilen alana yazmak zorundayız. Lakin başta dizinin herhangi bir boyunu vermek gibi bir zorunluluğumuz yok.

```
var dizi_adi = new Array();
```

Başta dizi boyutunu vermediğimiz dizimize yeni elemanlar ekleyerek, dizi boyunu kendimize göre ayarlayabiliriz.

Yukarıdaki dizi yapısını kullanarak bir örnek yapalım.

```
<script type="text/JavaScript">
  var dizi = new Array(10);
  for(var i = 0; i < 10; i++){
    dizi[i] = prompt("Değer: ");
  }
  for(var i = 0; i < 10; i++){
    document.write( dizi[i] + ". ");
  }
</script>
```

Yukarıdaki örneğimizde dizi adında oluşturmuş olduğumuz dizinin boyutu 10'dur. Uyarı kısmında belirttiğimiz gibi diziler 0 nolu indisten başlayarak verileri yerleştirmeye başlarlar. Doğal olarak son indis değeri 9 olacaktır. Yukarıdaki for yapısında yaptığımız 10 değer kontrolü bu yüzden. Bunun yerine `i <= 9` ifadesini de kullanabilirdik.

Örneğimize geri dönelim. İlk for yapısında, kullanıcıdan değerler istenilmekte ve bu değerler sırayla diziye yerleştirilmektedir. Bu yerleştirme işlemi; dizinin indisini belirtilerek yapılmaktadır. Dizinin indisini `i` değişkeni ile tutmaktayız. Yani `dizi[i]` ifadesi, dizinin bir indisini göstermektedir. Bunu bir piyano gibi düşünersek `i` değişkeniyle piyanonun tuşlarına sırasıyla basıyoruz. Burada `i` basacağımız tuşu gösteriyor gibi bir düşünce daha açıklayıcı olabilir. İkinci for yapısında ise dizinin indislerindeki değerleri okuma işlemi yapmaktayız. `i` değişkeni yine burada dizinin indislerini tutmaktadır. Birinci for yapısında dizinin indisleri boş idi. İkinci for yapısında ise bu indisler dolu durumdadır. `dizi[i]` ifadesiyle dizinin indislerindeki değerleri okuyoruz ve `document.write()` ifadesiyle de yazdırıyoruz.

Yukarıdaki örneğimize ait ekran çıktısını alalım ve dizimize rastgele değerlerle dolduralım.



Ekran çıktısında görüldüğü gibi prompt komutuyla sorulan değerlere sırasıyla cevap verdik ve dizimizi doldurduk. Daha sonra dizimize yazdığımız bu değerleri ekrana yazdırdık.

Örneğimizde yazdığımız bazı değerleri daha sonra değiştirmek isteyebiliriz. Böyle durumlarda yukarıda belirttiğimiz gibi indis değerlerine göre dizinin elemanlarını yeniden tanımlayabiliriz. Ayrıca aşağıdaki yapıyı kullanarak; for döngüsüne gerek kalmadan dizimizi değerlerle doldurabiliriz.

```
dizi[5] = 30;
dizi[9] = 1453;
dizi[0] = 42;
```

Tanımlamış olduğumuz dizinin başka bir kullanım stilleri de vardır. Bu kullanımda ise dizinin elemanları başta belirtilir. Bu formun standart kullanımı aşağıdaki gibidir.

```
var dizi_adi = new Array(deger1, deger2, deger3, ...);
```

Ya da

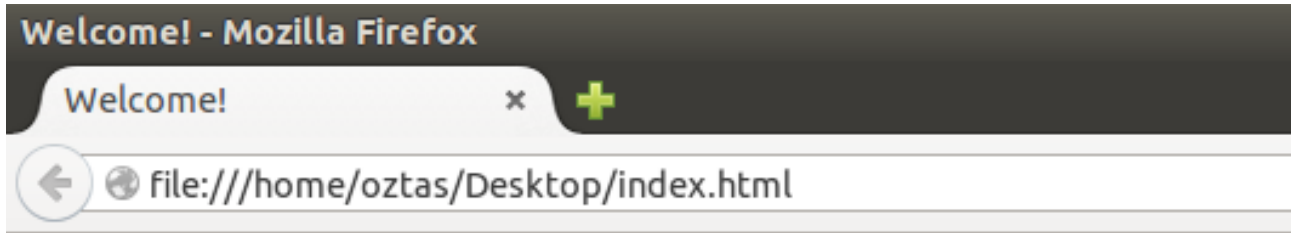
```
var dizi_adi = [deger1, deger2, deger3, ...];
```

Yukarıdaki dizi formumuzda `Array()` parantezleri içerisine değerleri varsayılan olarak tanımlayabiliriz. Bu formumuza ait aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var buyukSehirler = new Array("Ankara", "İstanbul", "İzmir", "Bursa",
    "Adana", "Antalya", "Konya");
    // veya asagidaki gibi dizimizi tanimlayabiliriz.
    /*
        var buyukSehirler = ["Ankara", "İstanbul", "İzmir", "Bursa",
    "Adana", "Antalya", "Konya"];
    */
```

```
document.write("7 Büyük Şehrimiz: " + "<BR>");
for(var i = 0; i < 7; i++){
    document.write(buyukSehirler[i] + ", ");
}
</script>
```

Yukarıda örneğimize ait ekran çıktısını alalım.



7 Büyük Şehrimiz:
Ankara, İstanbul, İzmir, Bursa, Adana, Antalya, Konya,

Oluşturmuş olduğumuz diziminin boyutunu bilmediğimiz durumlarda `length` anahtar kelimesini kullanabiliriz. `Length` anahtar kelimesinin kullanımı aşağıdaki gibidir.

```
dizi_adi.length;
```

`Length` anahtar kelimesinin kullanılmasıyla beraber dönecek olan değer tam sayı bir değerdir. Bunu bir değişkende tutmamız daha doğru olacaktır.

```
dizi_boyutu = dizi_adi.length;
```

Yukarıdaki örneğimize yeni değerler ekleyebiliriz. Bu işlem için dizinin boyutunu bilmemiz yeterlidir. Aşağıdaki örneğe bakalım.

```
<script type="text/JavaScript">
    var buyukSehirler = new Array("Ankara", "İstanbul", "İzmir", "Bursa",
    "Adana", "Antalya", "Konya");
    buyukSehirler[buyukSehirler.length] = "Bayburt";
    buyukSehirler[buyukSehirler.length] = "Siirt";
    buyukSehirler[buyukSehirler.length] = "Rize";
    buyukSehirler[buyukSehirler.length] = "Çanakkale";
    buyukSehirler[buyukSehirler.length] = "Muğla";
    document.write(buyukSehirler.length + " Büyük Şehrimiz: " + "<BR>");
    for(var i = 0; i < buyukSehirler.length; i++){
        document.write(buyukSehirler[i] + ", ");
    }
</script>
```

Yukarıdaki örneğimizde `length` anahtar kelimesiyle, tanımlamış olduğumuz dizinin sonuna yeni değerler ekledik.

Bir diğer konudan bahsetmet istiyorum. JavaScript, değişken veya dizi tanımlaması yaparken herhangi bir `data type` (veri tipi) istemediği için bir dizide istenilen veri türünde değerler tutabilmektedir.

```
<script type="text/JavaScript">
```

```
var dizi = new Array("Konya", 123, 3.14, true);  
</script>
```

JavaScript'te Arrays (Diziler)'in bir takım özellikleri vardır. Bunlar dizi fonksiyonlarıdır. Henüz fonksiyonlar konusuna girmediğimiz için bunları komut olarak adlandırabiliriz. Şimdi bu komutlar neler onları görelim.

10.1.1 reverse()

reverse() fonksiyonu, tanımlanan dizinin elemanlarının, dizi içindeki sıralamalarını tersine çevirir. reverse() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
dizi_adi.reverse();
```

Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">  
    var dizi = new Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
    dizi.reverse();  
    for(var i = 0; i < dizi.length; i++){  
        document.write(dizi[i] + ". ");  
    }  
</script>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Ekran alıntısında da görüldüğü gibi dizinin elemanları, sondan başa doğru yer değiştirmiş durumdadır.

10.1.2 sort()

sort() fonksiyonu ile sıralı olmayan diziler, küçükten büyüğe doğru sıralı hale getirilir. sort() fonksiyonunun standart kullanımı aşağıdaki gibidir.

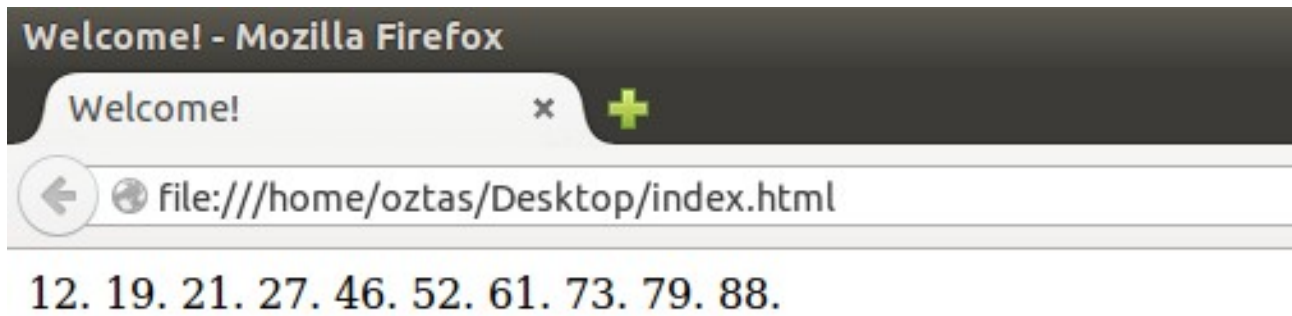
```
dizi_adi.sort();
```

sort() fonksiyonu ilgili olan aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">  
    var dizi = new Array(73, 46, 52, 61, 19, 88, 12, 27, 21, 79);  
    dizi.sort();  
    for(var i = 0; i < dizi.length; i++){  
        document.write(dizi[i] + ". ");  
    }  
</script>
```

```
</script>
```

Yukarıdaki yazmış olduğumuz örneğimizin ekran çıktısını alalım.



Ekran alıntısını incelediğimizde; sıralı olmayan dizimizin `sort()` fonksiyonu ile sıralı bir hale getirildiğini görüyoruz.

10.1.3 `shift()`

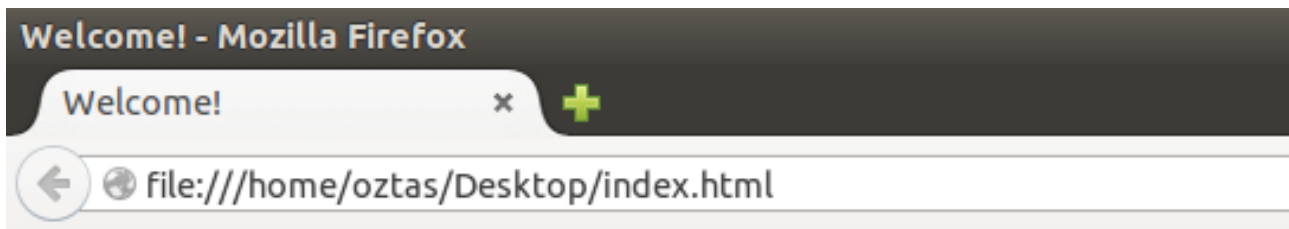
`shift()` fonksiyonu ile belirtilen dizi 1 basamak sola kaydırılır. Bu kaydırma sırasında soldan ilk sırada bulunan değer kesilip atılır ve dizinin boyutu 1 azalır. `shift()` komutunun standart kullanımı aşağıdaki gibidir.

```
dizi_adi.shift();
```

`shift()` fonksiyonu ile ilgili olan aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var dizi = new Array(73, 46, 52, 61, 19, 88, 12, 27, 21, 79);
    document.write("Dizinin ilk boyutu: " + dizi.length + "<BR>");
    dizi.shift();
    document.write("Dizinin yeni boyutu: " + dizi.length + "<BR>");
    for(var i = 0; i < dizi.length; i++){
        document.write(dizi[i] + ". ");
    }
</script>
```

Örneğimizin ekran çıktısını alalım.



Dizinin ilk boyutu: 10
Dizinin yeni boyutu: 9
46. 52. 61. 19. 88. 12. 27. 21. 79.

Ekran çıktımızı incelediğimizde; dizinin soldaki ilk değerinin (73) kesilip atıldığını görebiliyoruz.

10.1.4 slice()

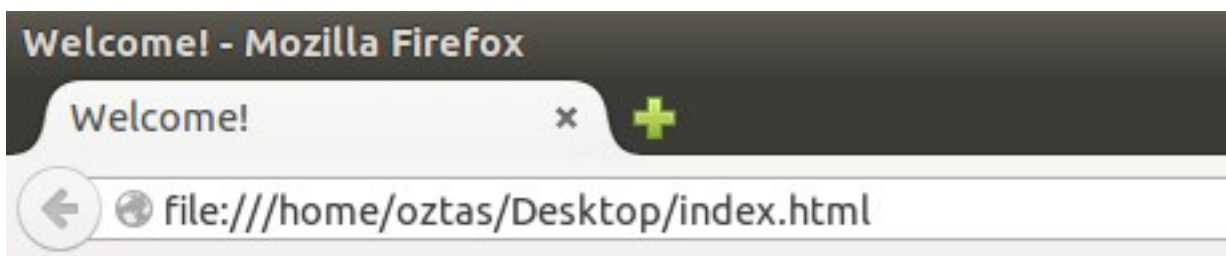
`slice()` fonksiyonu, dizinin belirtilen aralıkta elemanlarını almak için kullanılır. Bu alınan elemanlar diziden çıkarılmaz. Alınan elemanlar ya yeni bir diziye atılmalıdır ya da bir değişkene değer olarak atanmalıdır. `slice(x, y)` fonksiyonunda `x`: başlangıç değeri, `y`: bitiş değeridir. `slice(x, y)` fonksiyonun standart kullanımı aşağıdaki gibidir.

```
dizi_adi.slice(x, y);
```

`slice(x, y)` fonksiyonu içeren örneğimizi bakalım.

```
<script type="text/JavaScript">
    var dizi = new Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    document.write("Dizinin ilk boyutu: " + dizi.length + "<BR>");
    var alinanDegerler = dizi.slice(3, 7);
    document.write("Alınan değerler: " + alinanDegerler + "<BR>");
    document.write("Dizinin yeni boyutu: " + dizi.length + "<BR>");
    for(var i = 0; i < dizi.length; i++){
        document.write(dizi[i] + ". ");
    }
</script>
```

Örneğimize ait ekran çıktısını alalım.



Dizinin ilk boyutu: 10
Alınan değerler: 4,5,6,7
Dizinin yeni boyutu: 10
1. 2. 3. 4. 5. 6. 7. 8. 9. 10.

10.1.5 hasOwnProperty()

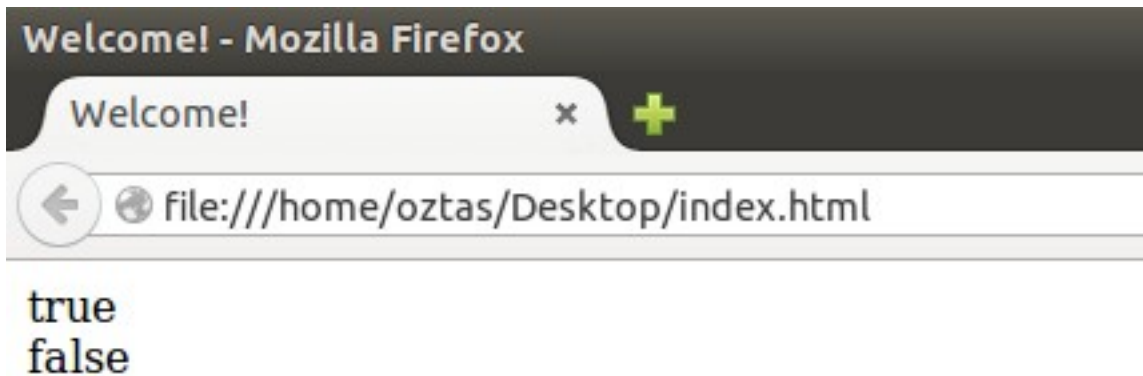
hasOwnProperty(x) fonksiyonu, belirtilen x değerini dizi içerisinde arar. Eğer belirtilen x değeri dizinin bir elemanı ise true, değilse false değerini döndürür. hasOwnProperty(x) fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
dizi_adi.hasOwnProperty(x); // x dizi icinde aranacak deger
```

hasOwnProperty(x) fonksiyonuna ait aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var dizi = new Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    document.write(dizi.hasOwnProperty(6));
    document.write("<BR>");
    document.write(dizi.hasOwnProperty(17));
</script>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



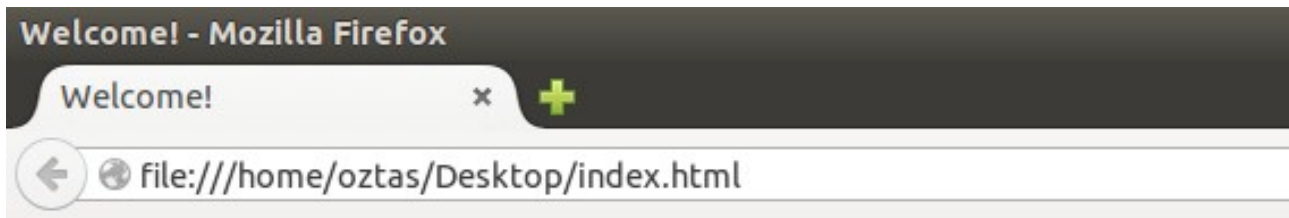
10.1.6 push()

Dizileri bir stack (yığıl) gibi düşünersek, bu fonksiyonun ne iş yaptığını anlayabiliriz. Bildiğiniz gibi stack'ler LIFO (Last In First Out) mantığına göre çalışır. push() fonksiyonuyla da dizinin sonuna eleman ekleyebiliriz ve diziyi genişletebiliriz. push() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
dizi_adi.push(x); // x: dizinin sonuna eklenecek eleman
```

push() fonksiyonuna ait aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var dizi = new Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    dizi.push(50);
    dizi.push(3.14);
    dizi.push('e');
    dizi.push(true);
    dizi.push("emrecan-oztas");
    for(var i = 0; i < dizi.length; i++){
        document.write(dizi[i] + ". ");
    }
</script>
```



1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 50. 3.14. e. true. emre-can-oztas.

Yukarıdaki örneğimizin ekran çıktısını alalım.

10.1.7 pop()

push() fonksiyonu dizinin sonuna eleman eklerken, pop() fonksiyonu dizinin sonundan elemanları koparır. pop() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
dizi_adi.pop();
```

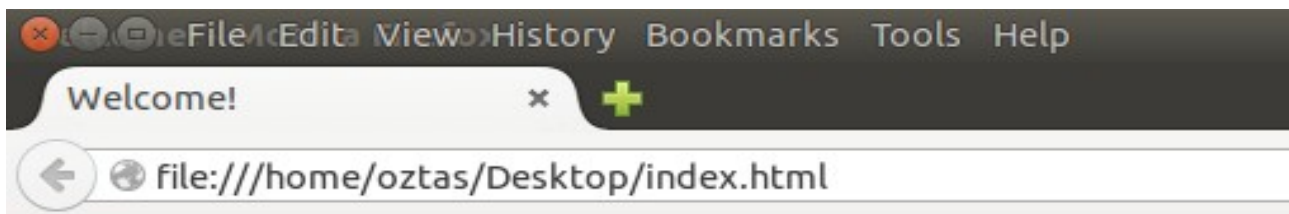
pop() fonksiyonu yukarıdaki gibi kullanılırsa, dizinin sonundaki elemanı koparıp alır fakat bu eleman uçar gider. Çünkü herhangi bir değişkene bağlı değil. O yüzden dizinin sonundaki elemanı alıp kullanacaksak pop() fonksiyonunu bir değişkene bağlamalıyız.

```
var degisken_adi = dizi_adi.pop();
```

pop() fonksiyonuna ait aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    var dizi = new Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    document.write("Dizinin ilk hali: ");
    document.write(dizi);
    document.write("<BR>");
    var d1 = dizi.pop();
    var d2 = dizi.pop();
    document.write("Dizinin son hali: ");
    document.write(dizi);
    document.write("<BR>");
    document.write("Koparılan değerler: " + d1 + ", " + d2);
</script>
```

Yukarıdaki programımıza ait ekran çıktısını alalım.



Dizinin ilk hali: 1,2,3,4,5,6,7,8,9,10

Dizinin son hali: 1,2,3,4,5,6,7,8

Koparılan değerler: 10, 9

Yukarıdaki örneğimizle birlikte diziler konusunun sonuna geldik.



Arrays (Diziler) bölümünün sonuna geldik. Buraya kadar olan konularımızda hep dizileri for döngüsüyle ekrana yazdırdık. Son örneğimizde de görebileceğiniz gibi dizileri for döngüsü yerine `document.write(dizi_adi)` komutuyla da yazdırabiliriz.

Hep for döngüsü kullandım çünkü alışkanlık kazanmanızı istedim. İlerleyen zamanlarda bu alışkanlığı kazandığınızda, istediklerinizi daha rahat yapabileceksiniz.

10.2 for...in Yapısı

`for...in` yapısı, C# bilenler için `foreach` yapısı, Java bilenler içinde `Enhanced for` yapısıyla ve yaklaşık olarak aynı formatla kullanılan bir yapıdır. `for...in` yapısı, aslında `for` yapısının daha kısa halidir ve bazı durumlarda `for` yapısından daha kullanışlıdır. Ne demek istediğimizi örneklerimizi incelediğinizde daha iyi anlayacaksınız. `for in` yapısının standart kullanımı aşağıdaki gibidir.

```
for(degisken in dizi){  
    // yapılacaklar.  
}
```

Yukarıdaki yapımızı biraz açıklamaya çalışalım daha sonra da bir kaç örnek vererek konumuzu daha iyi anlayalım. `for...in` yapısında `degisken` olarak belirttiğimiz alanda herhangi bir değişken tanımlamalıyız veya daha önce tanımlanmış değişkeni yazmalıyız. Bu yazacağımız değişken `in` yapısı yardımıyla `dizi` olarak belirttiğimiz dizi içerisinde daha doğrusu dizinin indisleri arasında sırasıyla gezinecek. Gezinirken de dizi değerlerini bize verecek.

`for...in` yapısı ile yararlı bir örnek yapalım. Örneğin ekrana yazdırmak için `document.write()` komutunu kullanıyorduk. Bu yapımızda `document` bir nesnedir ve bu nesnenin değişik fonksiyonları vardır. `write()` fonksiyonu da `document` nesnesinin bir fonksiyonudur. Fonksiyonlar konusundan henüz bahsetmedik fakat işlerimizi basitleştirip kolaylaştıran kod parçaları diyebiliriz. İşte şimdi `document` nesnesinin fonksiyonlarını sırasıyla `for...in` yapısı ile alalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome!</title>  
<meta charset="utf-8">  
</head>  
<body>  
<script type="text/JavaScript">  
    for(var i in document) document.write(i + "<br>");  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimizin ekran çıktısını aldığımız zaman `document` nesnesinin fonksiyonlarını daha rahat görebiliriz.

BÖLÜM 11:

Functions

Bu bölümde `function` (fonksiyon) kavramına değineceğiz ve detaylı olarak incelemeye çalışacağız. Öncelikle fonksiyon nedir? Ne işi yarar? Bu soruların cevaplarını arayalım. Programlama yaparken bazen bir çok yerde aynı işlemleri tekrarlamak zorunda kalırız. Örneğin bir kullanıcının sisteme kayıt işlemlerinde form kontrollerini düşünelim. Kullanıcıdan verilerin alınması o verilerin kontrol edilmesi gibi durumlar vardır. Böyle durumlarda her seferinde aynı kodları tekrar tekrar yazmak; hem programcı için zulümdür hemde web sayfasının yükünü artırır. Böyle durumlarda fonksiyonlar imdadımıza yetişir. Fonksiyonlar, kod parçalarıdır. Bu kod parçaları kod karmaşıklığını önler ve aynı zamanda yapılacak olan aynı tip işler için kodun tekrar edilmesini engeller. Fonksiyonlar, programlama dillerinin sihirli değnekleridir desek yeridir. JavaScript ortamında standart fonksiyon kullanımı aşağıdaki gibidir.

```
<script type="text/JavaScript">
    function fonksiyon_adi(parametreler){
        // yapılacak işlemler
    }
</script>
```

Fonksiyonlar `function` anahtar kelimesiyle tanımlanırlar. Her fonksiyonun bir adı vardır ve bu fonksiyonlar parametreler olarak belirtilen parantez içlerinde değerleri alabilirler. Aldıkları bu değerler üzerinde işlem yapma yetkisine sahiptirler.

JavaScript konularımızın başlarında, `<script type="text/JavaScript"></script>` etiketlerinin fonksiyon içermesi durumunda `<HEAD></HEAD>` etiketleri arasında açılmasının daha doğru olacağını belirtmiştik. Bizde artık bu bölümden sonra (buradan sonraki bir çok konumuz fonksiyonlar üzerine kurulu olacak) bu metodu uygulayacağız.

Kollarımızı sıvayıp basit fonksiyonlar tanımlamaya başlayalım ve fonksiyonların inceliklerini anlamaya çalışalım. Aşağıdaki gibi basit bir örnek yazalım.

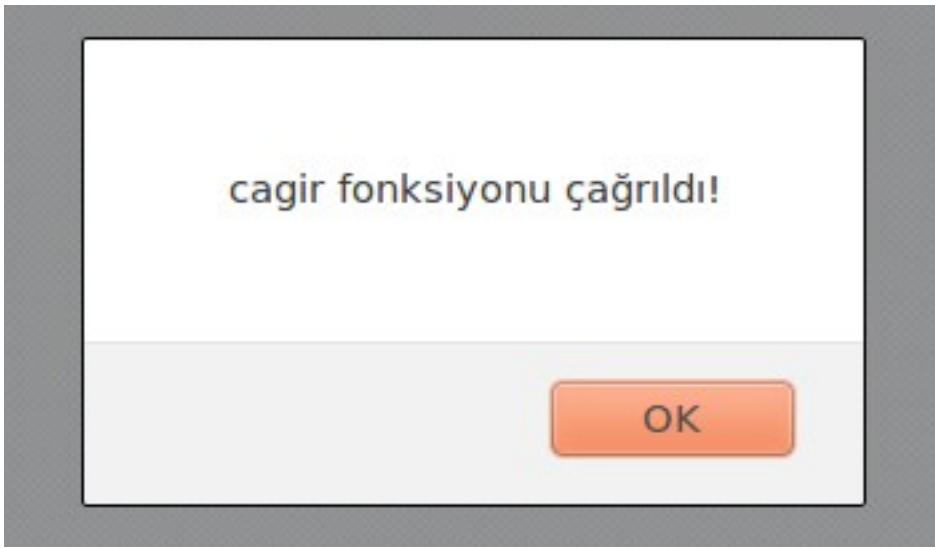
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function cagir(){
        alert("cagir fonksiyonu çağrıldı!");
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    cagir();
</script>
</body>
</html>
```

Yukarıdaki kodlarımızı inceleyelim. <HEAD></HEAD> etiketleri arasında <script type="text/JavaScript"></script> etiketlerini açtık ve içerisine bir fonksiyon yazdık. <BODY></BODY> etiketleri arasında bu fonksiyonu çağırdık. <script type="text/JavaScript"></script> etiketleri bir web sayfasında istenildiği gibi açılıp kapatılabilir. Ayrıca bu etiketler içerisindeki değişkenler, fonksiyonlar birbirleriyle ilişkilidir. Örneğimizde de olduğu gibi iki script etiketi birbirleriyle ilişkili durumda ve script etiketindeki fonksiyonu diğer script etiketinde çağırabiliriz.

Fonksiyon çağırma işlemi, fonksiyonun adının yazılması ve parantezlerin açılıp kapatılmasıyla gerçekleşir.

```
// fonksiyon cagirma
// gonderilecek degerler parametreler kismina yazilir.
fonksiyon_adi(parametreler);
```

Örneğimize ilişkin ekran çıktısını alalım.



Yukarıdaki ekran çıktısında olduğu gibi fonksiyonumuz doğru çalışıyor. Başka bir örnek yazalım. Bu örneğimizde yazdığımız iki fonksiyonu çağıralım.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function ilkFonksiyon(){
        document.write("Burası ilkFonksiyon");
        document.write("<BR>");
    }
    function ikinciFonksiyon(){
        document.write("Burası ikinciFonksiyon");
        document.write("<BR>");
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    // fonksiyonlar cagrildi.
    ilkFonksiyon()
```

```
        ikinciFonksiyon()
</script>
</body>
</html>
```

Yukarıdaki örneğimizde iki tane fonksiyon tanımlayıp daha sonra bu fonksiyonları çağırdık. İstedığımız kadar fonksiyon tanımlayabiliriz. Fonksiyon içinde fonksiyon da çağırabiliriz. Aşağıdaki örneğimizde olduğu gibi.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function ilkFonksiyon(){
        document.write("Burası ilkFonksiyon");
        document.write("<BR>");
        // ikinciFonksiyon isimli fonksiyon cagrildi.
        ikinciFonksiyon();
    }
    function ikinciFonksiyon(){
        document.write("Burası ikinciFonksiyon");
        document.write("<BR>");
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    // ilkFonksiyon isimli fonksiyon cagrildi.
    ilkFonksiyon()
</script>
</body>
</html>
```

Fonksiyonlar konusunda değinmemiz gereken bir başka konu var. Hatırlarsanız; yazdığımız JavaScript kodları web sayfalarının browser'de görüntülenmeleriyle hemen çalışmaya başlıyordu. Lakin fonksiyonlar için durum böyle değildir. Bir fonksiyonu çağırmadığımız sürece o fonksiyon asla çalışmaz.

Yukarıdaki örneklerimiz basit bir fonksiyon yazma ve çağırma işlemiydi. Daha karmaşık örnekler yaparak fonksiyonlar konusunu daha iyi anlayalım.

Aşağıdaki örneğimizde; ilk olarak kullanıcıdan alınacak değerlerin tutulacağı değişkenleri tanımlayalım. İlk fonksiyonda kullanıcıdan değerleri alalım ve tanımlanan değişkenlere bu değerleri atayalım. İkinci fonksiyonda ise bu değerleri yazdıralım. Örneğimiz aşağıdaki gibi olacaktır.

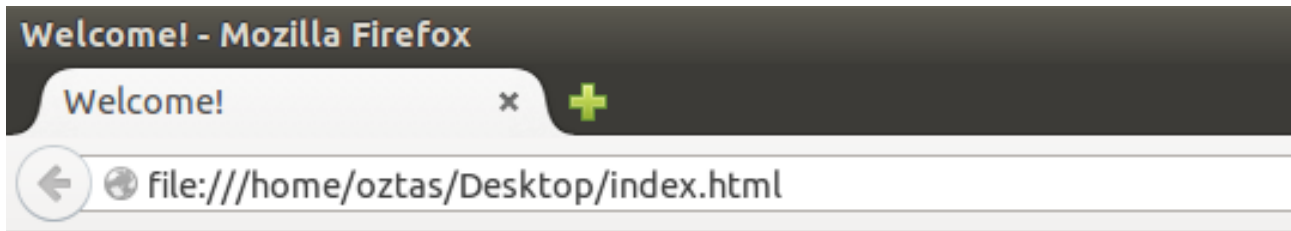
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    // degiskenler tanimlandi.
    var kullanıcıAdı;
    var kullanıcıSoyadı;
    var yer;
    var meslek;
```

```

function degerAl(){
    kullanıcıAdi = prompt("Adınız: ");
    kullanıcıSoyadi = prompt("Soyadınız: ");
    yer = prompt("Yaşadığınız yer: ");
    meslek = prompt("Mesleğiniz yer: ");
    degerYazdir();
}
function degerYazdir(){
    document.write("Adınız: " + kullanıcıAdi);
    document.write("<BR>");
    document.write("Soyadınız: " + kullanıcıSoyadi);
    document.write("<BR>");
    document.write("Yaşadığınız yer: " + yer);
    document.write("<BR>");
    document.write("Mesleğiniz: " + meslek);
}
</script>
</head>
<body>
<script type="text/JavaScript">
    // degerAl fonksiyonu cagrildi.
    degerAl();
</script>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısı aşağıdaki gibi olacaktır.



Adınız: Emre Can
 Soyadınız: ÖZTAŞ
 Yaşadığınız yer: ANKARA
 Mesleğiniz: Bilgisayar Mühendisi

Birinci fonksiyonda prompt komutuyla sorulan sorulara verdiğim cevaplar listelenmiş durumdadır.

Buraya kadar bölümde fonksiyonlara herhangi bir parametre göndermeden çağırdık ve kullandık. Fonksiyonlar, dışarıdan ek değerlerde alabilirler. Aldıkları bu değerleri kullanıp sonuç üretebilirler. Fonksiyonların dışarıdan aldığı değerlere parametreler denir. Bir fonksiyon istenildiği zaman parametre alabilirler.



Parametrelili fonksiyonlarla ilgili bir örnek yapalım. Örneğin kullanıcıdan ismini soralım. Kullanıcının ismini aldıktan sonra bunu fonksiyona gönderelim ve kullanıcıya bir mesaj verdirelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<html>
<head>

```

```

<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function selamla(kullaniciAdi){
        document.write("Merhaba, " + kullaniciAdi);
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    var ad = prompt("Kullanıcı adınız: ");
    selamla(ad);
</script>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında görüldüğü gibi, prompt ile sorulan soruya verdiğim cevap fonksiyona parametre olarak gönderilmiş ve bu fonksiyonda kendisine gelen değeri ekrana yazdırmıştır.

Parametrelili fonksiyonlara devam edelim. Daha karmaşık bir örnek yazalım. Bu örneğimizde kullanıcıdan vize ve final notlarını alalım ve hesaplayalım. Başka bir fonksiyon tanımlayalım. Bu fonksiyonda kullanıcının harf notunu hesaplasın. Diğer bir fonksiyonda harf notu CC olanlara geçti, notu CC'den küçük olanlara kaldı yazdırsın. Örnek kodlarımız aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    var sonuc;
    function notHesap(vize, fnal){
        sonuc = (vize * 0.6) + (fnal * 0.4);
        document.write("Vize: " + vize + "<BR>");
        document.write("Final: " + fnal + "<BR>");
        document.write("Not: " + sonuc + "<BR>");
        harfHesap(sonuc);
    }
    function harfHesap(sonuc){
        var harf;

```



```

        if(sonuc < 25){
            harf = "FF";
            durumHesap(harf)
        }else if (sonuc >= 25 && sonuc < 30){
            harf = "FD";
            durumHesap(harf)
        } else if (sonuc >= 30 && sonuc < 35){
            harf = "DD";
            durumHesap(harf)
        } else if (sonuc >= 35 && sonuc < 45){
            harf = "DC";
            durumHesap(harf)
        } else if (sonuc >=45 && sonuc < 55){
            harf = "CC";
            durumHesap(harf)
        } else if (sonuc >= 55 && sonuc < 65){
            harf = "CB";
            durumHesap(harf)
        } else if (sonuc >=65 && sonuc < 75){
            harf = "BB";
            durumHesap(harf)
        } else if(sonuc >=75 && sonuc < 85){
            harf = "BA";
            durumHesap(harf)
        } else{
            harf = "AA";
            durumHesap(harf)
        }
    }

    function durumHesap(harf){
        if(harf == "FF") document.write("Kaldi!");
        else if(harf == "FD") document.write("Kaldi!");
        else if(harf == "DD") document.write("Kaldi!");
        else if(harf == "DC") document.write("Kaldi!");
        else document.write("Geçti!");
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    var vize = prompt("Vize: ");
    var fnal = prompt("Final:");
    notHesap(vize, fnal);
</script>
</body>
</html>

```

Yukarıdaki kodlarımızı incelediğimizde; fonksiyonlara herhangi bir veri tipinde değer gönderebildiğimizi görebilirsiniz.

Örneğimizin ekran çıktısını alalım.



Vize: 4
Final: 6
Not: 4.8000000000000001
Kaldi!

Fonksiyonlar, ürettikleri sonuçları ekrana yazdırmak zorunda değildirler. Buraya kadar olan bölümde hep fonksiyonlara birtakım işler yaptırıp sonuçlarını ekrana yazdırdık. Fonksiyonların görevleri sadece bu kadar değildir. Fonksiyonlar aynı zamanda değer de döndürebilme yeteneğine sahiptirler. Fonksiyonların değer döndürme sırrı `return` anahtar kelimesinde saklıdır. Şimdi `return` anahtar kelimesini anlamaya çalışalım.

11.1 return

Fonksiyonları ikiye ayırmak istesek; değer döndüren ve değer döndürmeyen fonksiyonlar olarak ayırabiliriz. Buraya kadar olan kısımda tanımlamış olduğumuz fonksiyonlar değer döndürmeyen fonksiyonlar idi. `Return` anahtar kelimesi başlığı altında değer döndüren fonksiyonları incelemeye çalışacağız.

Fonksiyonlar, işleyen bir makinanın çarkları gibidir. Nasıl bir makina çarkları olmadan çalışmaz ya da zorlanarak çalışır, işte fonksiyonsuz programlar da böyle durumlardadır. Büyük projelerde fonksiyonlara sıkça başvurulur. Hele ki web ortamlarında fonksiyonların önemi tartışılmaz. Fonksiyonların ürettikleri değerler bazen tek başlarına bir anlam ifade etmeyebilir lakin başka bir fonksiyon için gerekli olabilirler. Değer döndüren fonksiyonlar ürettikleri değerleri `return` anahtar kelimesiyle ihraç ederler. Bir fonksiyonda `return` anahtar kelimesinin standart kullanımına bakalım.

```
<script type="text/JavaScript">
    function fonksiyon_adi(parametreler){
        // deger donduruluyor...
        return dondurulecek_deger;
    }
</script>
```

Çağırdığımız fonksiyondan dönen değer ise aşağıdaki şekilde alınır.

```
<script type="text/JavaScript">
    // return den donen degerler aliniyor...
    fonksiyonName(); // varsa parametreler gönderilir.
</script>
```

Normal olarak fonksiyon çağırırken kullandığımız yöntem ile `return` anahtar kelimesinden dönen değerler alınır. Yukarıdaki kullanımda değerler geldi fakat havada asılı kaldı. Çünkü herhangi bir değişkene bağlı değil. Durum böyle olunca değerler uçar gider. O yüzden dönen değerleri almak için bu satırı bir

değişkene bağlamamız gerekmektedir. Aşağıda olduğu gibi.

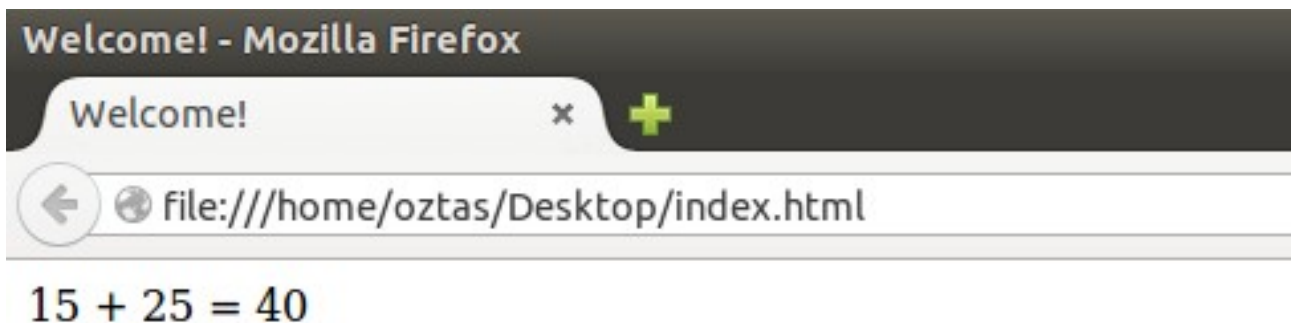
```
<script type="text/JavaScript">
    // return den donen degerler aliniyor...
    degisken_adi = fonksiyonName();
</script>
```

Yukarıdaki kullanımda artık dönen değerler bir değişkende tutulmaktadır.

Durumu daha iyi anlayabilmek için basit bir örnek yazalım. Fonksiyona iki sayı gönderelim, gönderdiğimiz fonksiyon bu sayıları toplayıp bize return ile döndürsün. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    var sonuc = 0;
    function topla(birinciSayi, ikinciSayi){
        sonuc = birinciSayi + ikinciSayi;
        //deger donduruluyor...
        return sonuc;
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    var toplamSonuc = topla(15, 25);
    document.write(" 15 + 25 = " + toplamSonuc);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Ekran çıktısında da görüldüğü gibi fonksiyondan dönen değer 40'tır.

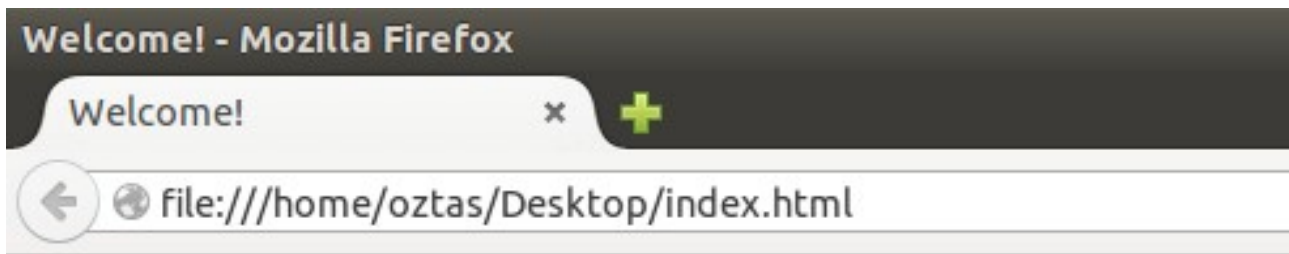
Burada bir şeyden daha bahsedelim. Return anahtar kelimesi diziler dışında sadece 1 değer döndürür. Bildiğiniz gibi diziler verileri sıralı olarak tutan yapılardır. Dolayısıyla bir dizi içerisinde birden fazla

değer olabilir. Lakin dizi dışındaki değerler bakımından return anahtar kelimesini kapasitesi 1 değerdir. Ayrıca bir fonksiyonda yalnızca bir kez return anahtar kelimesi kullanılabilir.

Yukarıdaki örneğimiz çok basit bir örnektir. Daha karmaşık bir örnek yapalım ve değer döndüren fonksiyonları anlama kat sayımız artsın. Bu örneğimizde; kullanıcıdan iki sayı alalım. Tanımlayacağımız fonksiyonlar ile dört işlemi gerçekleştirelim. Çıkan sonuçları return anahtar kelimesiyle döndürelim ve ekrana yazdıralım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function toplama(s1, s2){
        var toplam = s1 + s2;
        return toplam;
    }
    function cikarma(s1, s2){
        var cikar = s1 - s2;
        return cikar;
    }
    function carpma(s1, s2){
        var carp = s1 * s2;
        return carp;
    }
    function bolme(s1, s2){
        var bol = s1 / s2;
        return bol;
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    var birinciSayi = prompt("Birinci Sayi: ");
    var ikinciSayi = prompt("İkinci Sayi: ");
    birinciSayi = parseInt(birinciSayi);
    ikinciSayi = parseInt(ikinciSayi)
    islem1 = toplama(birinciSayi, ikinciSayi);
    islem2 = cikarma(birinciSayi, ikinciSayi);
    islem3 = carpma(birinciSayi, ikinciSayi);
    islem4 = bolme(birinciSayi, ikinciSayi);
    document.write("Girilen Sayilar: " + birinciSayi + " ve " +
    ikinciSayi);
    document.write("<HR>");
    document.write("a + b = " + islem1 + "<BR>");
    document.write("a - b = " + islem2 + "<BR>");
    document.write("a * b = " + islem3 + "<BR>");
    document.write("a / b = " + islem4 + "<BR>");
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Girilen Sayılar: 65 ve 13

$a + b = 78$

$a - b = 52$

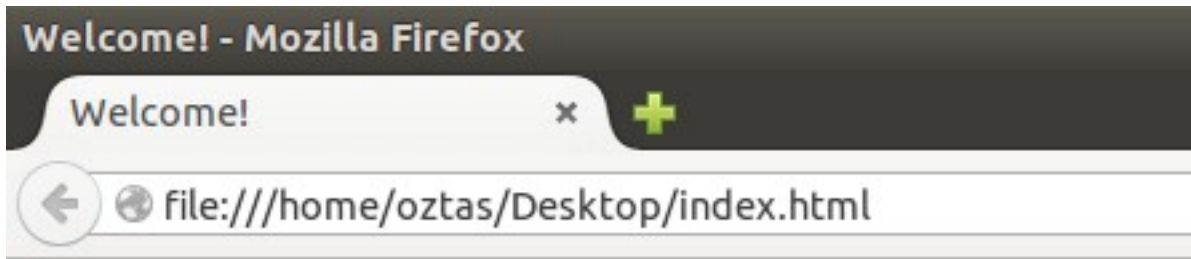
$a * b = 845$

$a / b = 5$

Fonksiyonlarımıza hep değişken tipinde değerler gönderdik fakat dizi hiç göndermedik. Peki diziler bir fonksiyona değer olarak nasıl gönderilir? Bir fonksiyona dizi göndermek, değişken göndermekten farksızdır. Aynı şekilde gönderilip alınır. Aşağıdaki örneğimizi fonksiyonlara dizi göndermeye ayıralım. Örneğimizi inceleyelim.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function diziAl(dizi){
        for(var j = 0; j < dizi.length; j++){
            document.write(dizi[j] + "<BR>");
        }
    }
</script>
</head>
<body>
<script type="text/JavaScript">
    var dizi = new Array(5);
    for(var i = 0; i < 5; i++){
        var diziDeger = prompt("Diziye eklenecek değer: ");
        dizi[i] = diziDeger;
    }
    diziAl(dizi);
</script>
</body>
</html>
```

Yukarıdaki örneğimizde fonksiyona bir dizi gönderdik. Gelen diziyi alan fonksiyonumuz bu diziyi ekrana yazdırdı. Yazmış olduğumuz kodlarımızı incelediğinizde, normal bir değişkenin fonksiyona gönderilmesi ile bir dizinin gönderilmesinin farklı bir yapısı olmadığını görebiliriz.



Java
Python
JavaScript
PHP
Ruby

Yukarıdaki örneğimize ait ekran çıktısını alalım.

BÖLÜM 12:

Events

Events (Olaylar), konusunu bu bölümde incelemeye çalışacağız. Olay nedir? Ne işe yarar bu soruların cevaplarını bulmaya çalışalım. Bir web sayfasına bağlandığımızı düşünelim. Bu sayfada yaptığımız her hareket bir olaydır. Örneğin üye formunu doldurma, herhangi bir linke veya butona tıklamak hatta web sayfasına bağlanmak veya web sayfasından ayrılmak bile bir olaydır. Bu olaylar JavaScript ile kontrol edilebilir.

Web sayfası üzerinde yaptığımız her şey bir olaydır dedik. Buradan hareketle bir çok olay vardır. Olay deyimleri HTML etiketleri ile birlikte kullanılır ve kullanılan bu olay deyimleri JavaScript kodlarını tetikler. Tetiklerden kastımız yazmış olduğumuz fonksiyonların çalıştırılması ya da başka bir deyişle aktif hale getirilmesidir.

Web sayfasında sıkça meydana gelen olayları görelim ve bu olayları JavaScript ile kontrol edelim. Sırasıyla olayları incelemeden önce temel HTML bilgisine sahip olmamız gerektiğini unutmayalım.

Olaylar ile yazacağımız örneklerimizi olabildiğince basit tutacağız. Çünkü henüz çok önemli bir mesafe kat edemedik. İlerleyen bölümlerde burada geçen olaylardan en çok kullanılanları sıkça tekrarlayacağız.

12.1 onload

onload, herhangi bir web sayfasının yüklenirken meydana gelen olaydır. Yani kısaca web sayfasının web tarayıcıda ilk görüntülendiği andır. Örneğin kullanıcı web sayfamızın url'ini yazdı ve server'a istek gönderdi. Web sayfamız ilk açıldığı anda herhangi bir mesaj verebiliriz veya aklınıza gelen, JavaScript ile yapılabilecek herhangi bir şey de olabilir.

onload olayı <BODY> etiketi ile birlikte kullanılır. Çünkü web sitemizin tüm içeriği <BODY></BODY> etiketleri arasında yer almaktadır. onload etiketinin kullanımı aşağıdaki gibidir.

```
<body onload="tanımlanan_olay">
```

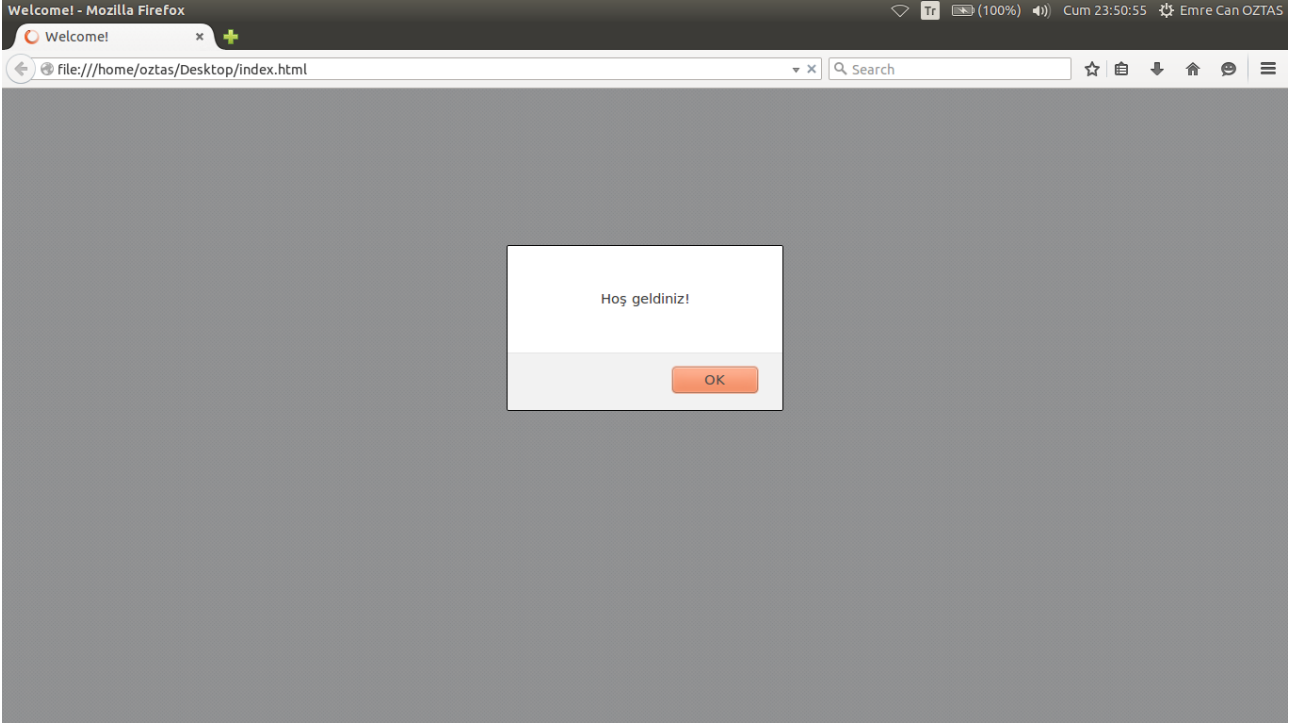
onload olayından sonra; tanımlanan herhangi bir fonksiyon çağrılabilir veya JavaScript kodu yazılabilir.

Örneğin web sayfamıza bağlanan kullanıcıya “Hoş geldiniz!” diyelim. Bunun için bir fonksiyon yazalım ve onload olayında bu fonksiyonu çağıralım. Örnek kodlarımız aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function selam(){
        alert("Hoş geldiniz!");
    }
</script>
</head>
```

```
<body onload="selam()">
</body>
</html>
```

Yukarıdaki kodlarımızın ekran çıktısı aşağıdaki gibi olacaktır.



Yukarıdaki ekran çıktımızda görüldüğü gibi web sitemiz tarayıcıda görüntülediği anda “Hoş geldiniz!” mesajı verdi. Ayrıca sayfamızı her refresh (yenile) ettiğimizde aynı mesajla karşılaşacağız.

Böyle “Hoş geldiniz!” gibi küçük mesajlar verdirmek için ya da küçük işlemler yapmak için fonksiyon tanımlamaya ihtiyaç yoktur. Aşağıdaki kodlarla da yukarıda yaptığımız işlemi gerçekleştirebiliriz.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
</head>
<body onload="alert('Hoş geldiniz!')">
</body>
</html>
```

12.2 onclick

onclick, herhangi bir nesneye tıklandığında oluşan olaydır. Örneğin bir butona tıklandığında mesaj verdirmek isteyebiliriz ya da bir formu göndermek isteyebiliriz. Bu ve buna benzer durumlarda onclick olayını kullanabiliriz. JavaScript ortamında en çok kullanılan olaylardan bir tanesidir. İlerleyen bölümlerde DOM (Document Object Model) konusuna girdiğimizde çeşitli formların doldurulması ve kontrol edilmesi işleminde onclick olayını sıklıkla kullanacağız. Onclick olayının örnek kullanımı aşağıdaki gibidir.

```
<input type="button" name="b1" value="Click" onclick="tanımlanan_olay"/>
```


OnClick olayı sadece bir buton'da değil, herhangi bir HTML elementinde de kullanılabilir. Lakin en sık olarak buton etiketiyle birlikte kullanılır. Örneğin aşağıdaki örnek kodda olduğu gibi bir resim ekleme etiketine de uygulanabilir.

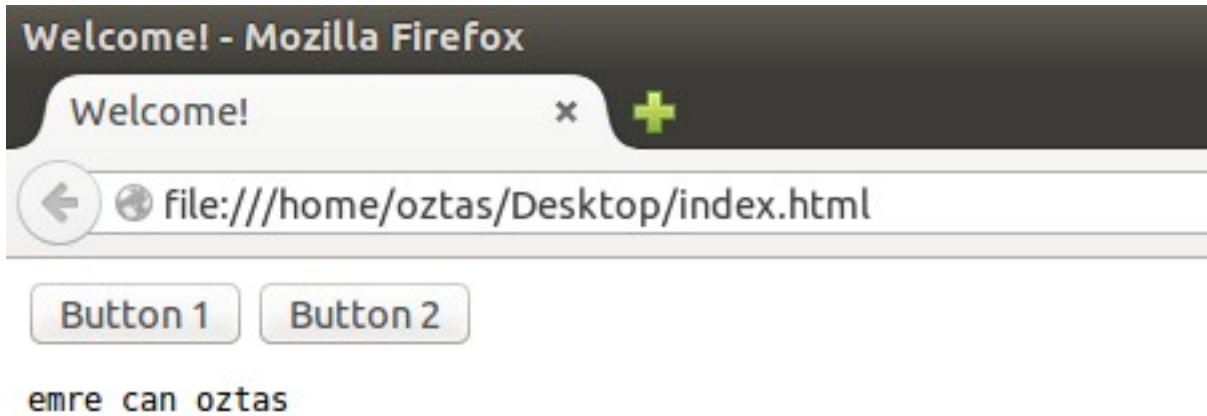
```

```

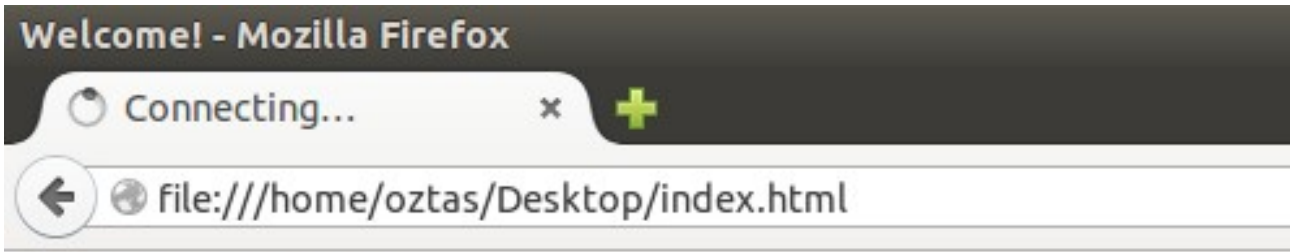
onclik olayı ile ilgili bir örnek yapalım. Örneğin formumuzda iki tane buton olsun ve kullanıcı bu butonlara tıkladığında çeşitli mesajlar verdirelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function btn1(){
        document.write("Button 1 tıklandı!");
    }
    function btn2(){
        document.write("Button 2 tıklandı!");
    }
</script>
</head>
<body>
<input type="button" name="b1" value="Button 1" onclick="btn1()"/>
<input type="button" name="b2" value="Button 2" onclick="btn2()"/>
<pre>emre can oztas</pre>
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısı aşağıdaki gibi olacaktır.



Yukarıdaki örnek çıkmızda olduğu gibi herhangi bir butona tıklayalım.



Button 1 tıklandı!

Button1'e tıkladığımız zaman ekranda bulunan herşey silindi ve “Button 1 tıklandı!” yazısı geldi. Neden böyle bir şey oldu? `document.write()` komutu ekranda bulunan herşeyi siler ve belirtilen şeyi ekrana yazar. Buna dikkat edelim. `document.write()` komutu yerine HTML etiketlerinin kullanılması gerekir. HTML etiketlerinin kullanımı da DOM konusunda detaylı olarak inceleyeceğiz.

12.3 ondblclick

`ondblclick`, herhangi bir nesneye çift tıklanıldığında oluşan olaydır. Örneğin `onclick` olayı butona veya herhangi belirtilen bir nesneye, fare ile bir kere tıklanıldığı durumda oluşan bir olaydır. `ondblclick` olayının buton etiketi ile kullanımı aşağıdaki gibidir.

```
<input type="button" name="b1" value="Click" ondblclick="tanımlanan_olay"/>
```

`OnClick` olayındaki örneğimizi yeniden yazacak olursak;

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset = "utf-8">
<script type="text/JavaScript">
    function btn1(){
        document.write("Button 1 çift tıklandı!");
    }
    function btn2(){
        document.write("Button 2 çift tıklandı!");
    }
</script>
</head>
<body>
<input type="button" name="b1" value="Button 1" ondblclick="btn1()"/>
<input type="button" name="b2" value="Button 2" ondblclick="btn2()"/>
<pre>emre can oztas</pre>
</body>
</html>
```

Yukarıdaki kodlarımızı kaydedip oluşan web sayfamızı tarayıcıda görüntülediğimizde; iki butondan herhangi birini çift tıkladığımızda oluşturmuş olduğumuz fonksiyonlar aktif hale gelecektir.

12.4 onmouseover

onmouseover, fare işaretçisinin belirlenen herhangi bir nesne üzerine geldiği durumda oluşan olaydır. Bu nesne buton, resim veya herhangi başka bir şey olabilir. Biz, örnek bir resim üzerinden gideceğiz. `img` etiketiyle birlikte onmouseover olayının kullanımı aşağıdaki gibidir.

```

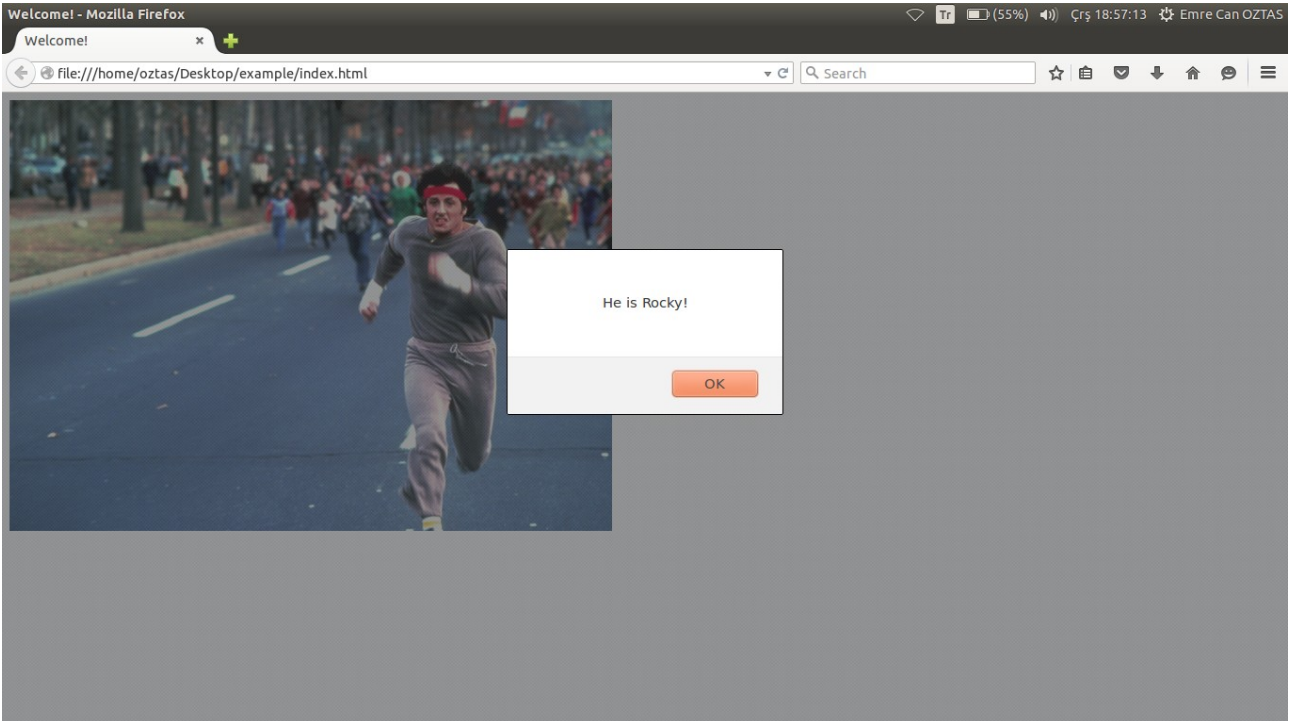
```

Her zaman olduğu gibi yine bir örnek üzerinden gidelim. Bir `file` (dosya) oluşturalım ve bu dosyanın içerisinde hem bir resim hem de web sayfamızı yerleştirelim. Siz de herhangi bir resim dosyasıyla çalışabilirsiniz. Örnek kodlarımız aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8"/>
<script type="text/JavaScript">
    function resim(){
        alert("He is Rocky!");
    }
</script>
</head>
<body>

</body>
</html>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Yukarıdaki örnek ekran alıntımızda görüldüğü gibi fare ile resmin üzerine geldiğimizde, `alert` komutu ile mesaj verebilmekteyiz. Daha önce belirttiğimiz gibi `alert` komutu yerine buraya kadar olan bölümde öğrendiğimiz herhangi bir JavaScript komutunu da kullanabiliriz. Burada herşey bize kalmış durumdadır. Örneğin `prompt` komutuyla kullanıcıdan herhangi bir bilgi alabiliriz.

onmouseover olayını herhangi bir nesne üzerinde kullanabiliriz dedik ve resim ekleme etiketini kullandık. Bunun yerine örneğin bir butona da onmouseover olayı verebiliriz. Bir butonun üzerine fare ile gelindiği zaman onmouseover olayını aşağıdaki gibi kullanabiliriz.

```
<input type="button" name="unnamed" value="click" onmouseover="alert('This is button!')" />
```

12.5 onmouseout

onmouseout olayı, onmouseover olayından tam tersi nitelikte yani fare işaretçi ile belirtilen nesnenin dışına çıkılması durumunda oluşan olaydır. onmouseout olayı da herhangi bir nesneye verilebilir. Onmouseout olayının herhangi bir etikette kullanımı (örneğin resim etiketinde) aşağıdaki gibidir.

```

```

onmouseover olayında kullandığımız örneğimizi tekrar edecek olursak;

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8"/>
<script type="text/JavaScript">
    function resim(){
        alert("Rocky's picture!");
    }
</script>
</head>
<body>

</body>
</html>
```

Yukarıdaki örneğimizi kaydedip, web sayfamızı tarayıcıda görüntülediğimiz de; fare ile resmin dışına çıkıldığı durumda alert ile uyarı mesajı verilecektir.

Daha önce de dediğimiz gibi bu olayların kullanımı tamamen bize kalmış durumdadır.

12.6 Other Events

En çok kullanılan olaylardan bahsettik fakat diğer kullanılan olaylardan henüz bahsetmedik çünkü çok önemli bir mesafe kat etmedik dolayısıyla diğer olayları açıklayacak seviyeye gelmedik. İlerleyen konularda diğer kullanmadığımız ve kullandığımız olayları açıklamaya çalışacağız. Burada bir açıklamada bulunalım. HTML5 ile gelen yeniliklerden birisi de olayların yelpazesinin genişlemesidir.

Diğer kullanmadığımız olayların listesi ve açıklamaları aşağıdaki gibidir.

| Event (Olay) | Description (Açıklama) |
|--------------|--|
| offline | İnternet bağlantısının kopması durumu |
| onabort | Sayfa yüklenirken kullanıcının; sayfayı iptal etmesi durumunda oluşan olay |
| onafterprint | Sayfa yazdırıldıktan sonra oluşan olay |

| | |
|------------------|---|
| onbeforeunload | Sayfa yüklenmeden önce oluşan olay |
| onbeforeprint | Sayfa yazdırılmadan önce oluşan olay |
| onblur | Odağın pencereden farklı bir yere kayması |
| oncanplay | Medya çalınmaya başlanması |
| oncanplaythrough | Medya çalınmaya başlanıp, sonuna kadar gelinmesi |
| onchange | Herhangi bir element içeriği (input alanlar) değiştirilmesi durumu |
| oncontextmenu | contextmenu seçilmesi |
| ondrag | Herhangi bir elementin sürüklenmesi durumu |
| ondragend | Herhangi bir elementin sürüklenip bırakılması |
| ondragenter | Herhangi bir elementin belirtilen hedefe sürüklenmesi |
| ondragleave | Herhangi bir element belirtilen hedefe sürüklenip bırakıldığında oluşan olay |
| ondragover | Herhangi bir element sürüklenip belirlenen hedefin üzerine gelindiğinde oluşan olay |
| ondragstart | Sürükleme işleminin başlaması |
| ondrop | Sürüklenen elementin bırakılması |
| ondurationchange | Medya uzunluğunun değiştirilmesi |
| onemptied | Medya kaynağının alınamaması |
| onended | Medya çalınmasının bitmesi |
| onerror | Herhangi bir hata meydana geldiğinde |
| onfocus | Pencereye odaklanması |
| onformchange | Form değiştirildiğinde |
| oninput | Herhangi bir elemente kullanıcı girişi durumu |
| oninvalid | Herhangi bir element geçersiz olduğu durum |
| onkeydown | Herhangi bir karakter girildiğinde (input alanlar) |
| onkeypress | Herhangi bir tuşa basıldığında (input alanlar) |
| onkeyup | Herhangi bir tuşa basılıp, bırakıldığı durum |
| onloadeddata | Medya verisi yüklendiğinde |
| onloadedmetadata | Herhangi bir medya yürütülürken diğer medyanın açılması durumunda oluşan olay |
| onloadstart | Medya verisi yürütülmeye başlandığında |
| onmessage | Herhangi bir mesaj başlatılması |
| onmousedown | Fare ile tıklanması |
| onmousemove | Fare işaretçisinin hareket ettirilmesi |
| onmousewheel | Farenin topunun çevrilmesi |
| onoffline | Web sayfasının kapatılması |
| ononline | Web sayfasının açılması |

| | |
|--------------------|--|
| onpagehide | Pencerenin gizlenmesi |
| onpageshow | Pencere görünür olduğunda |
| onpause | Medya verisinin durdurulması |
| onplay | Medya verisinin yürütülmeye başlanması |
| onplaying | Medya verisi yürütülmeye başlandığında |
| onpopstate | Pencere geçmişi değiştirildiği zaman |
| onprogress | Tarayıcı, medya verisine ulaştığı zaman |
| onratechange | Medya verisinin yürütme sayısı değiştiğinde |
| onreadystatechange | Ready – State durumu değiştirilmesi |
| onredo | Döküman içeriğindeki değişiklik geri verildiğinde |
| onresize | Pencere boyutu değiştirildiğinde |
| onscroll | Kaydırma çubuğu kullanıldığında |
| onseeked | Medya elementinin arama değeri gerekli uzunlukta değilse ve arama işlemi durdurulmuşsa oluşan olay |
| onselect | Element seçildiği durum |
| onstalled | Medya verisi alınırken meydana gelen herhangi bir hata durumunu |
| onstorage | Herhangi bir döküman yüklendiğinde |
| onsubmit | Form gönderilmesi |
| onsuspend | Tarayıcı, medya verisini alırken, alınan verinin durdurulması durumunda oluşan olay |
| ontimeupdate | Medya yürütülürken, zaman durumunun değiştirilmesi |
| onundo | Döküman içerisindeki içerik değiştirilip geri alındığı durumda oluşan olay |
| onunload | Kullanıcının sayfadan ayrılması durumu |
| onvolumechange | Medya yürütülürken ses değerinin değiştirilmesi |
| onwaiting | Medya yürütülürken durdurulması ve devam etmek için onay beklemesi durumu |

BÖLÜM 13:

HTML DOM

HTML DOM yani HTML Document Object Model, bir yapıdır. Bu yapı ile web sayfasında; HTML elementlerinin tüm özelliklerine ulaşılabilir ve değiştirilebilir. Örneğin üyelik kaydının alındığı bir sayfa düşünelim. Bu üyelik kaydında kullanıcının çeşitli bilgileri alınır. Bu alınan bilgiler belli bir yapıda olmalıdır. Dolayısıyla kullanıcıdan bilgileri alınırken, bu bilgiler belli bir kriterlere göre kontrol edilmelidir. Bu işlem de DOM (Document Object Model) ile yapılır. DOM ile sadece form kontrolü dışında daha bir çok değişik yapılar oluşturulabilir. Hatta bir web sayfasının CSS yapısı bile değiştirilebilir.

DOM bir W3C (Web Wide Consortium) standardıdır. W3C DOM standardı üçe ayrılır. Bunlar:

- Core DOM: Döküman tipleri için standart bir modeldir.
- XML DOM: XML sayfaları için standart bir modeldir.
- HTML DOM: HTML sayfaları için standart bir modeldir.

XML yapısı da W3C tarafından oluşturulmuş bir standarttır. Bu da ekli bilgi olarak aklımızda bulunsun.

Şuan bizim ihtiyacımız olan HTML DOM. Dolayısıyla HTML DOM konusunu detaylı olarak incelemeye çalışacağız. HTML DOM'a göre bir web sayfasının tüm elementlerine JavaScript ile erişilebilir. Bizde sırasıyla bir web sayfanın elementlerine erişmeye ve çeşitli işlemler yapmaya çalışacağız.

13.1 Elementlere Erişmek

HTML elementlerine erişmek için çeşitli yollar vardır. Bu yollardan herhangi birisini kullanarak bu elementlere erişebiliriz. HTML elementlerine erişim sağlandıktan sonra `.value` anahtar kelimesi ile bu elementin değeri okunur. Fakat bu kullanıcıdan giriş alınırken kullanılan bir anahtar kelimedir. Mesela herhangi bir `text` alanının değeri `.value` anahtar kelimesi ile çekilebilir. Bu ve buna benzer durumlarda; kullanıcının doğrudan doldurduğu veya değiştirdiği alanlarda `.value` anahtar kelimesi kullanılır. Bunu kesinlikle unutmayalım.

Kullanıcının doğrudan müdahale edemediği elementler de vardır. Mesela `<p>`, `<label>`, `<code>` v.s gibi kısaca biz bunlara gömülü HTML elementleri diyelim. İşte bu gömülü HTML elementlerinin değerleri alınırken `.value` anahtar kelimesi çalışmaz. Bunun yerine farklı bir anahtar kelime kullanılmalıdır. Bu anahtar kelime de `.innerHTML` anahtar kelimesidir.

Bu bilgiler şimdilik aklımızın bir köşesinde bulunsun.

13.1.1 getElementById()

`getElementById`, `document` nesnesinin bir fonksiyonudur. Bu fonksiyon ile herhangi bir HTML elementinin `id`'si kullanılarak, bu HTML elementinin özelliklerine erişilebilir veya özellikleri değiştirilebilir.

Kullanıcının doğrudan değiştirebildiği bir HTML elementinin özelliklerine ulaşmak için aşağıdaki standart kullanım uygulanmalıdır.

```
document.getElementById("elementId").value;
```

Yukarıdaki yapımızı, herhangi bir değişkene bağlamalıyız aksi halde her zaman söylediğimiz gibi aldığımız değer havada kalır, uçup gider. Aşağıdaki gibi kullanım daha uygundur.

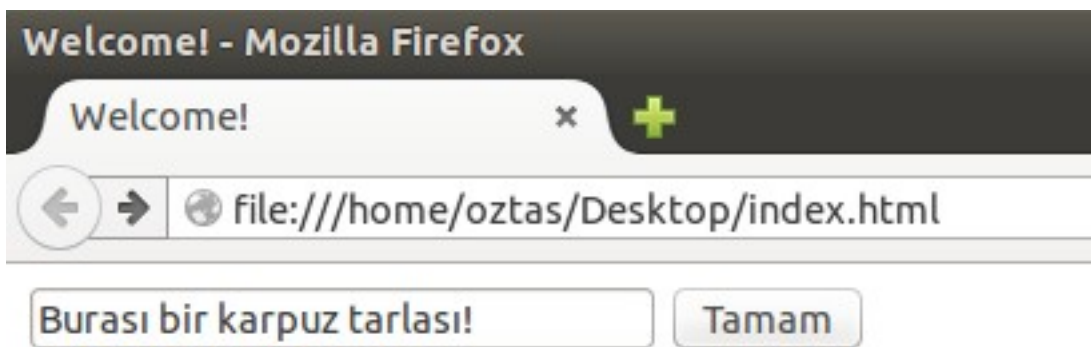
```
var deger = document.getElementById("elementId").value;
```

getElementById() yapısını daha iyi anlamak için basit bir örnek yapalım. Örnek bir web sayfası tasarlayalım. Bu web sayfasında bir text alanı ve bir buton olsun. Butona basıldığı anda text alanındaki değeri yakalayalım ve yazdıralım. Örneğimiz aşağıdaki gibi olacaktır.

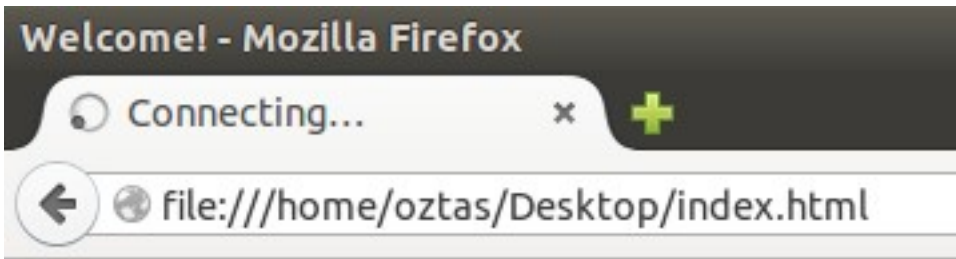
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function getir(){
        // id'si kelime olan text alaninin degerini alalim.
        var cumle = document.getElementById("kelime").value;
        document.write(cumle);
    }
</script>
</head>
<body>
<input type="text" name="kelime" id="kelime" />
<input type="button" name="ok" value="Tamam" onclick="getir()" />
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını almadan önce biraz inceleyelim. Bir elemente herhangi bir id verilebilir. id'ye verdiğimiz değerin akılda kalıcı olması için name ile aynı ismi vermeye özen gösterelim. JavaScript fonksiyonlarımızı da tetiklemek için Events (Olaylar) kullanacağız.

Şimdi yukarıdaki örneğimizin ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü herhangi bir cümleyi text alanı yazdık. Şimdi butona tıklayalım ve yazmış olduğumuz text alanındaki değeri yakalayalım. Ekran çıktımız aşağıdaki gibi olacaktır.



Burası bir karpuz tarlası!

text alanına yazdığımız cümlemizi doğru bir şekilde yakaladık ve ekrana yazdırdık. Şimdilik document.write() komutuyla yetineceğiz. Bu bölümün ikinci başlığında yani “Elementleri Değiştirmek” kısmında ekrandan herhangi bir element kaybolmadan çıktılarımızı verebileceğiz.

getElementById() fonksiyonunu daha iyi anlamak için başka bir örnek yapalım. Bu örneğimizde basit bir üye kayıt sayfası tasarlayalım. Daha sonra kullanıcının girdiği bilgileri, JavaScript ortamında kontrol edelim. İlk olarak web sayfamızı tasarlayalım. Örnek web sayfamız aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<form>
<table>
<tr>
<td colspan="2"><b>ÜYE KAYIT</b></td>
</tr>
<tr>
<td colspan="2"><hr></td>
</tr>
<tr>
<td><label>Adınız:</label></td>
<td><input type="text" name="ad" id="ad"/></td>
</tr>
<tr>
<td><label>Soyadınız:</label></td>
<td><input type="text" name="soyad" id="soyad"/></td>
</tr>
<tr>
<td><label>Yaşadığınız Yer:</label></td>
<td><input type="text" name="yer" id="yer"/></td>
</tr>
<tr>
<td><label>Mesleğiniz:</label></td>
<td><input type="text" name="meslek" id="meslek"/></td>
</tr>
<tr>
<td><label>e-Mail:</label></td>
<td><input type="text" name="mail" id="mail"/></td>
</tr>
<tr>
<td><label>Kullanıcı Adı:</label></td>
<td><input type="text" name="kadi" id="kadi"/></td>
</tr>
<tr>
<td><label>Şifre:</label></td>
```

```

<td><input type="password" name="sifre1" id="sifre1"/></td>
</tr>
<tr>
<td><label>Şifre (Tekrar):</label></td>
<td><input type="password" name="sifre2" id="sifre2"/></td>
</tr>
<tr>
<td colspan="2"><hr></td>
</tr>
<tr>
<td><input type="button" value="Kaydı Tamamla" onclick="kontrolEt()"/></td>
<td><input type="reset" value="İptal Et"/></td>
</tr>
</table>
</form>
</body>
</html>

```

Yukarıda gördüğünüz gibi web sayfamızı tasarladık. Şimdi kodlamaya geçelim. İlk olarak; alanların boşluk / doluluğunu kontrol edelim. Daha sonra geçerli bir e-mail adresi girilmiş mi buna bakalım. Son olarakta şifre alanlarına girilen değerlerin eşitliğini kontrol edelim ve şifre alanının en az 6 karakterli olup olmadığını kontrol edelim.

Web sayfamızla birlikte yazmış olduğumuz JavaScript kodlarımız aşağıdaki gibidir.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function kontrolEt(){
        var uyeAd = document.getElementById("ad").value;
        var uyeSoyad = document.getElementById("soyad").value;
        var uyeYer = document.getElementById("yer").value;
        var meslek = document.getElementById("meslek").value;
        var mail = document.getElementById("mail").value;
        var kullanıcıAdi = document.getElementById("kadi").value;
        var sifre1 = document.getElementById("sifre1").value;
        var sifre2 = document.getElementById("sifre2").value;

        if(uyeAd == "" || uyeSoyad == "" ||
            uyeYer == "" || meslek == "" ||
            mail == "" || kullanıcıAdi == "" ||
            sifre1 == "" || sifre2 == "")
            document.write("Lütfen boş alan bırakmayınız!");
        else if(mail.indexOf("@") < 0)
            document.write("Lütfen geçerli bir mail adresi giriniz!");
        else if(sifre1.length < 6)
            document.write("Şifre alanı en az 6 karakter olmalıdır!");
        else if(sifre1 != sifre2)
            document.write("Şifre alanları uyuşmamaktadır!");
        else{
            document.write("Kayıt Başarılı!" + "<BR>");
            document.write("Hoş geldiniz! " + kullanıcıAdi);
        }
    }
</script>
</head>
<body>

```

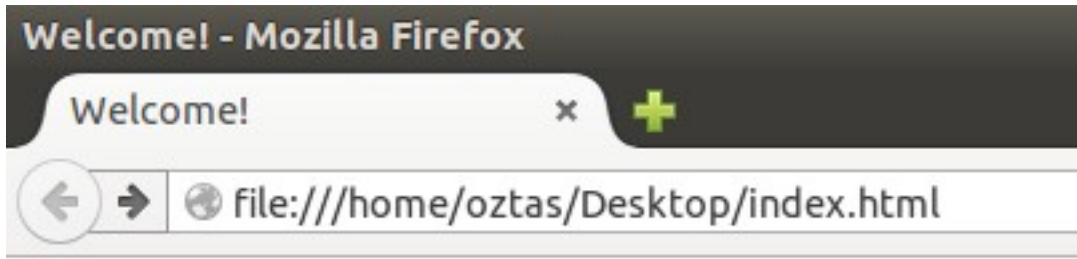
```

<form>
<table>
<tr>
<td colspan="2"><b>ÜYE KAYIT</b></td>
</tr>
<tr>
<td colspan="2"><hr></td>
</tr>
<tr>
<td><label>Adınız:</label></td>
<td><input type="text" name="ad" id="ad"/></td>
</tr>
<tr>
<td><label>Soyadınız:</label></td>
<td><input type="text" name="soyad" id="soyad"/></td>
</tr>
<tr>
<td><label>Yaşadığınız Yer:</label></td>
<td><input type="text" name="yer" id="yer"/></td>
</tr>
<tr>
<td><label>Mesleğiniz:</label></td>
<td><input type="text" name="meslek" id="meslek"/></td>
</tr>
<tr>
<td><label>e-Mail:</label></td>
<td><input type="text" name="mail" id="mail"/></td>
</tr>
<tr>
<td><label>Kullanıcı Adı:</label></td>
<td><input type="text" name="kadi" id="kadi"/></td>
</tr>
<tr>
<td><label>Şifre:</label></td>
<td><input type="password" name="sifre1" id="sifre1"/></td>
</tr>
<tr>
<td><label>Şifre (Tekrar):</label></td>
<td><input type="password" name="sifre2" id="sifre2"/></td>
</tr>
<tr>
<td colspan="2"><hr></td>
</tr>
<tr>
<td><input type="button" value="Kaydı Tamamla" onclick="kontrolEt()"/></td>
<td><input type="reset" value="İptal Et"/></td>
</tr>
</table>
</form>
</body>
</html>

```

Üye kayıt sayfamızı tasarladık ve önceden belirlediğimiz gibi form için gerekli kontrolleri yaptık. Yukarıdaki sayfamız kusursunuz çalışıyor. Fakat bu sayfamız sadece bir prototip. Yani web sayfaları böyle kullanılmaz. Form'un method'una göre post veya get edilmesi gerekiyor, e-mail alanını daha kesin bir şekilde kontrol edilmeli ve ayrıca kullanıcı adı değerinde unique (eşsiz) bir değer olup olmadığı kontrol edilmeli. Form'un post veya get edilmesini bu bölümün sonlarına doğru göreceğiz.

Yukardaki formumuzun ekran çıktısını alalım. Bakalım tasarımı güzel olmuş mu?



ÜYE KAYIT

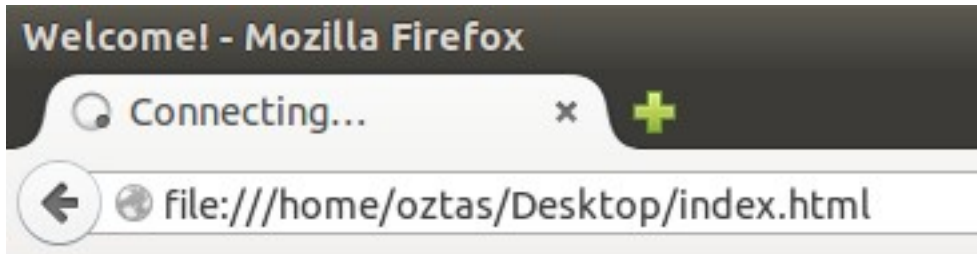
| | |
|------------------|---|
| Adınız: | <input type="text" value="Emre Can"/> |
| Soyadınız: | <input type="text" value="ÖZTAŞ"/> |
| Yaşadığınız Yer: | <input type="text" value="ANKARA"/> |
| Mesleğiniz: | <input type="text" value="Bilgisayar Mühendisi"/> |
| e-Mail: | <input type="text" value="emrecanoztas@outlook.com"/> |
| Kullanıcı Adı: | <input type="text" value="jsawyer"/> |
| Şifre: | <input type="password" value="....."/> |
| Şifre (Tekrar): | <input type="password" value="....."/> |

Kaydı Tamamla

İptal Et

Sayfamızın tasarımı gayet güzel duruyor. CSS ile biçimlendirme yapabiliydik fakat bu kitabın konusu olmadığı için şimdilik böyle kalması daha doğru. Tasarlamış olduğumuz formu doldurdum. İsterseniz sizde kaynak kodları indirip sayfamızın doğru çalışıp, çalışmadığını kontrol edebilirsiniz. Hatta kendiniz yeni eklemeler de yapıp kullanabilirsiniz.

Üyelik formumuzu doldurduktan sonra “Kaydı Tamamla” butonuna tıklayalım. Doldurduğum formda herhangi bir hata yapmadım. Dolayısıyla herhangi bir hata mesajı ile karşılaşmayacağım. Son olarak aşağıdaki gibi bir ekran çıktımız olacaktır.



Kayıt Başarılı!
Hoş geldiniz! jsawyer

Buraya kadar olan bölümde hep; text'lerden değerleri okuduk. Zaten diğer input girişleri içinde mantık aynıdır. Hadd-i zatinde hepsi birer input giriş niteliği taşır ve .value anahtar kelimesi kullanılır. Gömülü HTML elementlerinin değerlerini nasıl alabiliriz, şimdi bunun üzerinde biraz konuşalım.

Konunun başında da belirttiğimiz gibi gömülü HTML elementleri elde etmek için .innerHTML anahtar kelimesini kullanmalıyız. Yani .value yerine .innerHTML yazmamız gerekiyor. Ama yine örnek kullanımını verelim.

```
document.getElementById("elementId").innerHTML;
```

Yukarıdaki kullanımın doğru olmadığını, bunu bir değişkene bağlayıp kullanmamızın daha doğru olacağını belirtmiştik. Yani aşağıdaki gibi bir kullanım daha evladır.

```
var deger = document.getElementById("elementId").innerHTML;
```

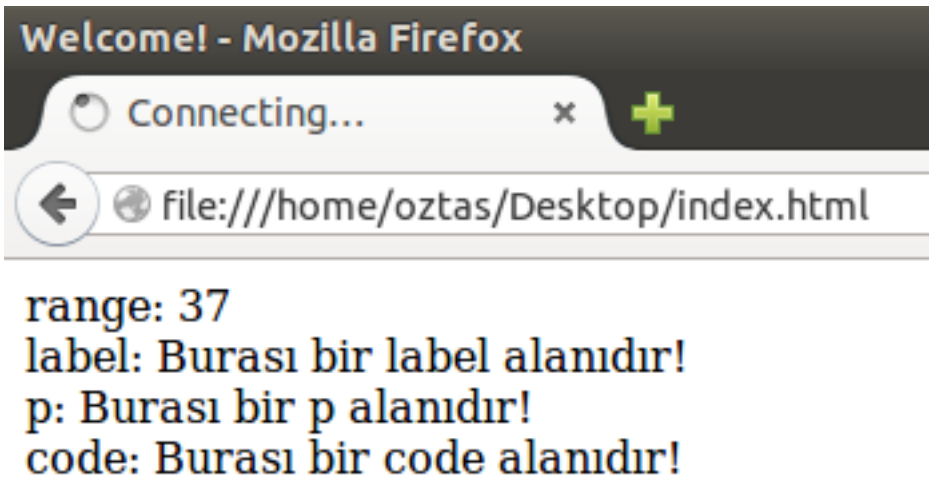
Yukarıdaki vermiş olduğumuz kullanıma dair, gömülü HTML elementleriyle ilgili basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function getir(){
        // degerleri alalım.
        // input alanların degerleri .value ile alınır.
        var rangeDeger = document.getElementById("range").value;
        // gömülü html elementlerin degerleri .innerHTML ile alınır.
        var labelDeger = document.getElementById("label").innerHTML;
        var pDeger = document.getElementById("p").innerHTML;
        var codeDeger = document.getElementById("code").innerHTML;
        // aldığımız degerleri yazdıralım.
        document.write("range: " + rangeDeger + "<BR>");
        document.write("label: " + labelDeger + "<BR>");
        document.write("p: " + pDeger + "<BR>");
        document.write("code: " + codeDeger);
    }
</script>
</head>
<body>
<input type="range" name="range" min="0" max="50" id="range"/><BR>
<label id="label">Burası bir label alanıdır!</label>
<p id="p">Burası bir p alanıdır!</p>
<code id="code">Burası bir code alanıdır!</code><BR>
<input type="button" value="Değerleri Yakala!" onclick="getir()"/>
</body>
</html>
```

Yukarıdaki örneğimizin ilk olarak nasıl görüldüğüne bakalım.



Yukarıdaki ekran çıktısında görüldüğü gibi “Değerleri Yakala!” butonuna tıklayalım ve ekrandaki belirtmiş olduğumuz elementlerin değerlerini yazdıralım. Bir sonraki ekran çıktımız aşağıdaki gibi olacaktır.



Yukarıdaki ekran alıntısında olduğu gibi belirlediğimiz gömülü HTML elementlerin değerlerini aldık ve ekrana yazdırdık. Son örnekle birlikte; getElementById() fonksiyonu ile ilgili herhangi bir sıkıntımız kalmadığına inanıyorum.

getElementById() metodu, web sayfalarında kullanıcıyla iletişim kurmak için en çok kullanılan yoldur. Diğer değineceğimiz yollar farklı alanlarda kullanılır fakat biz hepsine değinmeyeceğiz.

13.1.2 getElementsByTagName()

getElementsByTagName(), document nesnesinin bir fonksiyonudur. Bu fonksiyon ile elementlerin etiket isimlerine göre değerlerine erişilebilir. Burada dikkat edilmesi gereken nokta; bir web sayfasında aynı elementten birden fazla kullanılabilir. Dolayısıyla alınan elementler bir dizi değişkende depolanır. getElementsByTagName() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
document.getElementsByTagName("elementTagName");
```

Alınan değerin havada kalmaması için bu elde ettiğimiz değeri bir değişkene bağlayalım.

```
var deger = document.getElementsByTagName("elementTagName");
```

getElementsByTagName() fonksiyonu ile basit bir örnek yapalım ve mantığını anlamaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

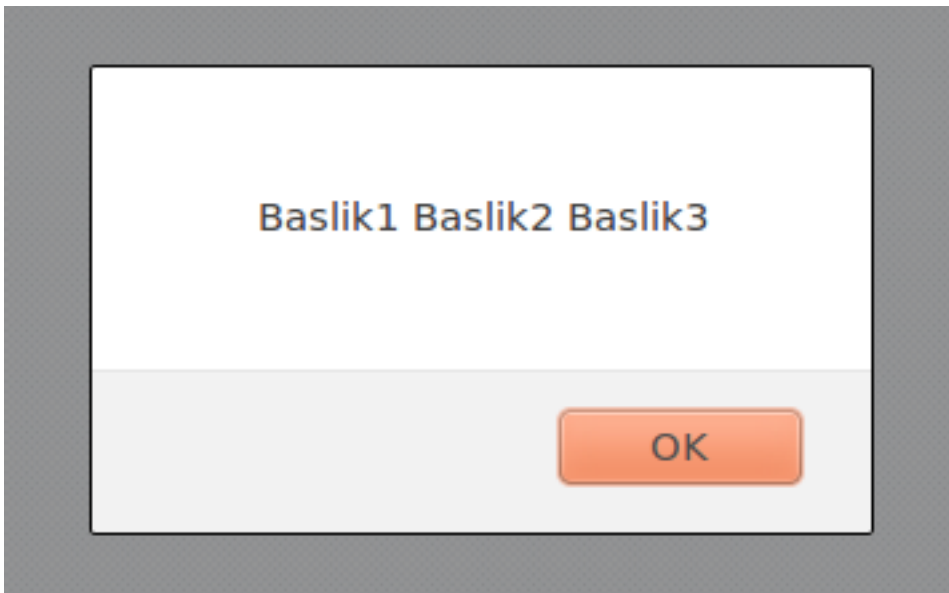
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function getir(){
        var deger = document.getElementsByTagName("title");
        document.write(deger[0].text);
    }
</script>
</head>
<body>
<input type="button" value="Click" onclick="getir()" />
</body>
</html>
```

Örneğimizi biraz inceleyelim. getElementByTagName() fonksiyonu ile <title> etiketinin değerini aldık ve bunu bir değişkene attık. Değişkenin yazdırılmasına dikkat edelim. Konumuzun başında da belirttiğimiz gibi getElementByTagName() fonksiyonu ile alınan değerler bir dizi değişken olmaktadır. Dolayısıyla değişkenimiz bir dizi olduğu için; deger[0].text şeklinde yazdırmalıyız.

Yukarıdaki örneğimizi çalıştırıp deneyebilirsiniz. getElementByTagName() fonksiyonunun çalışma stili üzerine biraz daha konuşalım. Yukarıdaki örneğimize yeni <title> etiketleri ekleyelim. Ve <title> etiketlerindeki değerleri yazmaya çalışalım. Yeni örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Baslik1</title>
<title>Baslik2</title>
<title>Baslik3</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function getir(){
        // tum title degerleri aliniyor...
        var deger = document.getElementsByTagName("title");
        alert(deger[0].text + " " +
            deger[1].text + " " +
            deger[2].text);
    }
</script>
</head>
<body>
<input type="button" value="Click" onclick="getir()" />
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Ekran çıktımızı incelediğimizde bütün <title> etiketlerinin değerlerini aldığımızı görüyoruz. Sizde benzer yapılar kurarak; çeşitli yapılar oluşturabilirsiniz.

getElementsByTagName() fonksiyonu ile bir örnek daha yapalım, anlama katsayımız artsın. Bu örneğimizde <p> etiketlerini getElementByName() fonksiyonu ile almaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function getir(){
        var x = document.getElementsByTagName("p");
        document.write(x[2].innerHTML);
    }
</script>
</head>
<body>
<p>W1</p><p>W2</p><p>W3</p><p>W4</p>
<input type="button" value="Click!" onclick="getir()"/>
</body>
</html>
```

Yukarıdaki örneğimizde; .text yerine innerHTML ile <p> etiketlerinin değerlerini yazdırdık. Çünkü <p> etiketi, HTML sayfasının varsayılan etiketleri arasında bulunmamaktadır. getElementById(), getElementsByTagName() dışında getElementByName() ve getElementByClassName() gibi çeşitli stiller vardır. Fakat burada benim değinmek istediğim şey; temel bir JavaScript için getElementById() fonksiyonunun kullanımını çok iyi bilinmelidir. getElementsByTagName() fonksiyonunun kullanımını da örnek olması açısından paylaştık. Diğer kullanım stilleri farklı amaçla kullanılmaktadır ve “Advanced JavaScript” konusuna dahil olmaktadır. Dolayısıyla bu kitabın ilgi alanı içerisinde bulunmamaktadır.

13.2 Elementlerin Değerlerini Değiştirmek

Elementlere nasıl erişebileceğimizi bir önceki bölümde gördük. Bölümün sonlarına doğru da

getElementById() fonksiyonunun kullanımının çok iyi bilinmesi gerektiğinden bahsettik. Bu bölümde HTML elementlerine erişip değerlerini değiştirmeyi göreceğiz. Konularımız ağırlıklı olarak getElementById() fonksiyonu üzerine olacak.

HTML elementlerine ulaşmak ile değerlerini değiştirmek arasında pek bir fark yoktur. Aynı stilde kullanımları vardır. Yine kullanıcının doğrudan değiştirebildiği HTML elementlerinin değerlerini .value ve gömülü HTML elementlerinin değerlerini .innerHTML anahtar kelimesiyle değiştireceğiz. Bu dediklerimizin standart gösterimlerini yazalım.

```
document.getElementById(elementId).value = atanacakDeger;  
// veya  
document.getElementById(elementId).innerHTML = atanacakDeger;
```

Elementlerin değerlerini değiştirmek üzere bir örnek yazalım. Örneğin sayfamızda iki tane text ve bir tane buton olsun. Birinci text alanına yazdığımız değeri ikinci text alanına ekleyelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome!</title>  
<meta charset="utf-8">  
<script type="text/JavaScript">  
    var d1;  
    function ekle(){  
        d1 = document.getElementById("txt1").value;  
        document.getElementById("txt2").value = d1;  
    }  
</script>  
</head>  
<body>  
<label>input1:</label><input type="text" name="txt1" id="txt1"/><BR>  
<label>input2:</label><input type="text" name="txt2" id="txt2"/><BR>  
<input type="button" value="Ekle" onclick="ekle()"/>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.

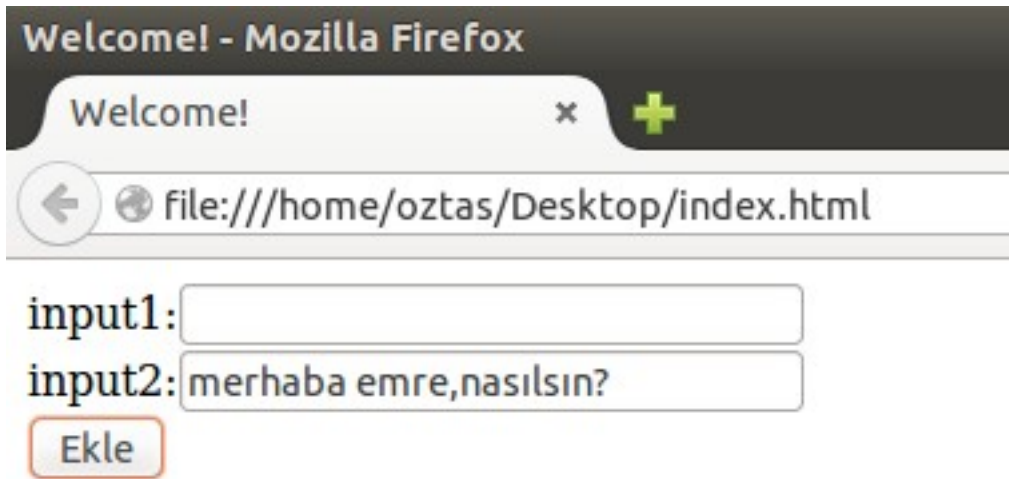


Yukarıdaki ekran alıntısında da görüldüğü gibi; birinci text alanına yazdığımız değer, “Ekle” butonuna tıklayınca ikinci text alanına eklendi. Birinci text alanına yeni bir değer yazdığımız zaman bu seferde yeni değer ikinci text alanına eklenecektir.

Bu yazdığımız üzerinden konumuza devam edelim. Örneğin birinci text alanına yazdığımız değerler ikinci text alanında silinmeden art arda yazılsın. Bu iş içinde yazacağımız kodlar aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var d1;
    var d2;
    function ekle(){
        d1 = document.getElementById("txt1").value;
        d2 = document.getElementById("txt2").value;
        document.getElementById("txt2").value = d2 + d1;
        /* birinci text alanına yazdığımız
           değerleri, yeni değerler almak için
           silelim.
        */
        document.getElementById("txt1").value = "";
    }
</script>
</head>
<body>
<label>input1:</label><input type="text" name="txt1" id="txt1"/><BR>
<label>input2:</label><input type="text" name="txt2" id="txt2"/><BR>
<input type="button" value="Ekle" onclick="ekle()"/>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktımızda görüldüğü gibi birinci text alanına yazdığımız değerler ikinci text alanında art arda yazılabiliyor.

Yukarıdaki örneğimiz üzerinden başka örnekler de yazalım. Örneğin sayfamıza bir “Reset” butonu ekleyelim. Bu Reset butonuna tıklayınca her iki text alanındaki değerler de silinsin. Ayrıca HTML5 ile input alanına gelen readonly yordamını da ikinci text alanına ekleyelimki kimse ikinci text alanına doğrudan müdahale edemesin. Bu iş için yazacağımız örneğimiz aşağıdaki gibi olacaktır.

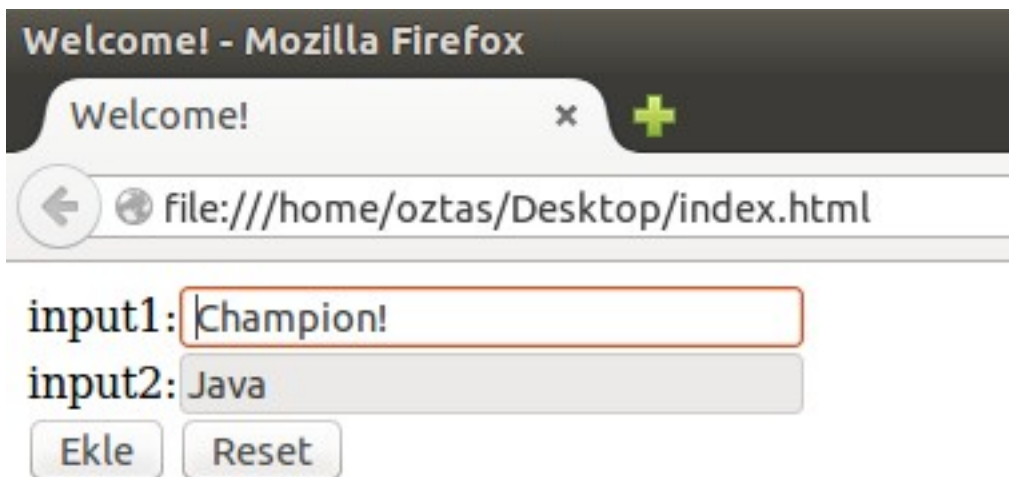
```
<!DOCTYPE html>
<html>
```

```

<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var d1;
    var d2;
    function ekle(){
        d1 = document.getElementById("txt1").value;
        d2 = document.getElementById("txt2").value;
        document.getElementById("txt2").value = d2 + d1;
        /* birinci text alanina yazdigimiz
           degerleri, yeni degerler almak icin
           silelim.
        */
        document.getElementById("txt1").value = "";
    }
    function sil(){
        document.getElementById("txt1").value = "";
        document.getElementById("txt2").value = "";
    }
</script>
</head>
<body>
<label>input1:</label><input type="text" name="txt1" id="txt1"/><BR>
<!--
HTML5 ile gelen readonly anahtar kelimesinin kullanımına dikkat edelim -->
<label>input2:</label><input type="text" name="txt2" id="txt2"
readonly/><BR>
<input type="button" value="Ekle" onclick="ekle()"/>
<input type="button" value="Reset" onclick="sil()"/>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Örneğimizi yazdık, ekran çıktısını aldık. Siz de buna benzer yapılar kurabilirsiniz.

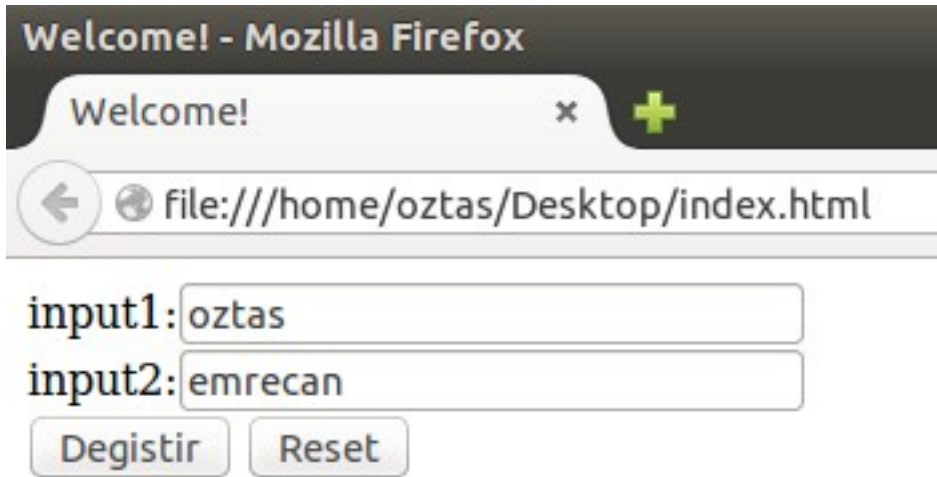
Son bir örnek yapalım. Bu örneğimizde; yukarıdaki HTML yapısını biraz değiştirerek kullanalım. Birinci text alanındaki değeri ikinci text alanıyla, ikinci text alandaki değeri de birinci text alanındaki değerle değiştirelim örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var d1;
    var d2;
    var temp;
    function degistir(){
        d1 = document.getElementById("txt1").value;
        d2 = document.getElementById("txt2").value;
        temp = d1;
        document.getElementById("txt1").value = d2;
        document.getElementById("txt2").value = temp;
    }
    function sil(){
        document.getElementById("txt1").value = "";
        document.getElementById("txt2").value = "";
    }
</script>
</head>
<body>
<label>input1:</label><input type="text" name="txt1" id="txt1"/><BR>
<label>input2:</label><input type="text" name="txt2" id="txt2"/><BR>
<input type="button" value="Degistir" onclick="degistir()"/>
<input type="button" value="Reset" onclick="sil()"/>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü üzere; “Değiş tir” butonuna tıkladığımız anda iki text alanındaki değerler yer değiştirmektedir.

Buraya kadar bölümde hep kullanıcının doğrudan müdahale edebildiği input alanlar üzerine örneklerimizi yaptık. Buradan sonraki bölümde gömülü HTML elementlerinin değerlerinin değiştirilmesi üzerinde duralım. Başta da söylediğimiz gibi gömülü HTML elementlerinin içeriklerini .innerHTML anahtar kelimesiyle değiştiriyorduk.

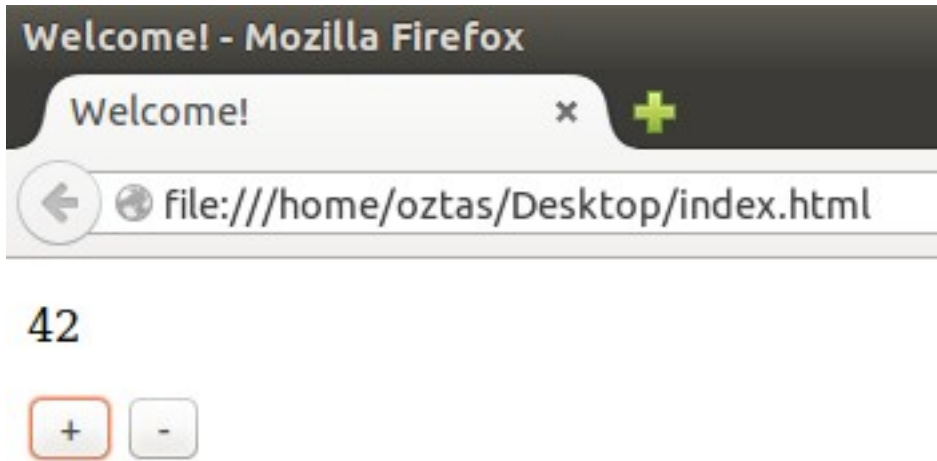
Gömülü HTML elementlerinin içeriklerini değiştirme ile ilgili bir örnek yapalım. Örneğin sayfamızda iki tane buton ve bir tane <p> gömülü HTML etiketi olsun. Bu butonlardan bir tanesi belirlediğimiz değeri arttırken diğeri değeri azaltsın. Yazacağımız örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var deger = 0;
    function arttir(){
        deger++;
        document.getElementById("p").innerHTML = deger;
    }
    function azalt(){
        deger--;
        document.getElementById("p").innerHTML = deger;
    }
</script>
</head>
<body>
<p id="p">0</p>
<input type="button" value="+" onclick="arttir()"/>
<input type="button" value="-" onclick="azalt()" />
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktımızı incelediğimizde; + butonuna bastığımız zaman <p> etiketindeki değer artmakta ve – butonuna bastığımız zaman <p> etiketindeki değer azalmaktadır.

Böylelikle gömülü HTML etiketlerindeki değerleri değiştirme mantığını anladık. Lakin bu yapımız üzerinden biraz daha çalışalım. Yukarıdaki örneğimize; bir <p> daha ekleyelim ve en son hangi hangi işlemi yaptığımızı bize gösterebilir. Bir de “Reset” butonu ekleyerek alanlarımızı varsayılan değerlerine çekelim. Bu iş için yazacağımız örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var deger = 0;
    function arttir(){
        deger++;

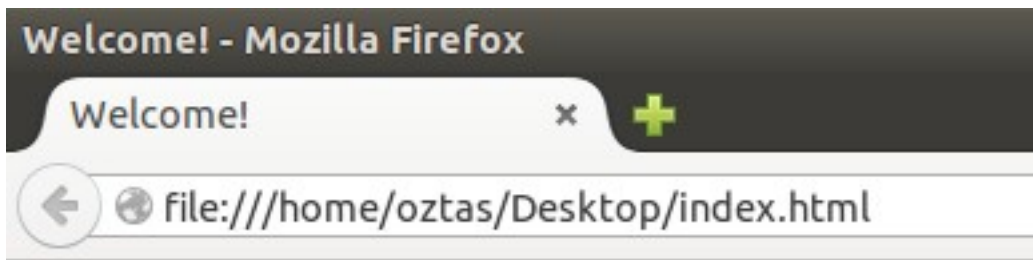
```

```

        document.getElementById("p1").innerHTML = "Arttırıldı!";
        document.getElementById("p").innerHTML = deger;
    }
    function azalt(){
        deger--;
        document.getElementById("p1").innerHTML = "Azaltıldı!";
        document.getElementById("p").innerHTML = deger;
    }
    function sil(){
        document.getElementById("p").innerHTML = 0;
        document.getElementById("p1").innerHTML = "Resetlendi!";
    }
}
</script>
</head>
<body>
<p id="p">0</p>
<p id="p1">Henüz işlem Yapılmadı!</p>
<input type="button" value="+" onclick="arttir()" />
<input type="button" value="-" onclick="azalt()" />
<input type="button" value="Reset" onclick="sil()" />
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



3

Azaltıldı!



Yukarıdaki ekran alıntısını incelediğimizde; hangi işlemi yaparsak yapalım bunlar bize p1 id'li <p> etiketimizde bildiriliyor.

.innerHTML özelliği ile gömülü HTML elementlerinin değiştirilmesi ile ilgili son iki örneğimizden birincisini yazalım. Bu örneğimizde; sayfamızda bir text ve bir <p> etiketi olsun. text etiketine yazdığımız her karakter eş zamanlı olarak <p> etiketine yazılsın. Bu iş için onclick yerine onkeyup olayını kullanmalıyız. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var total = "";
    function yazdir(){

```

```

        total = document.getElementById("txt").value;
        document.getElementById("p").innerHTML = total;
    }
</script>
</head>
<body>
<input type="text" name="txt" placeholder="Birşey yazın!" id="txt"
onkeyup="yazdir()" />
<p id="p"></p>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi text alanına ne yazarsak; <p> etiketine doğrudan yazılmaktadır.

.innerHTML mantığını anlamak için daha doğrusu üye kayıt sayfasının mantığını anlamak için son bir örnek yapalım. Daha önce kullanıcıdan değerleri alıp kontrol edip bunları document.write() ile ekrana yazdırmıştık. Şimdi bu örneğimizi gömülü HTML elementlerini kullanarak, ekrandan herhangi bir nesne kayboldan yazdıralım. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function kontrolEt(){
        var uyeAd = document.getElementById("ad").value;
        var uyeSoyad = document.getElementById("soyad").value;
        var uyeYer = document.getElementById("yer").value;
        var meslek = document.getElementById("meslek").value;
        var mail = document.getElementById("mail").value;
        var kullanıcıAdi = document.getElementById("kadi").value;
        var sifre1 = document.getElementById("sifre1").value;
        var sifre2 = document.getElementById("sifre2").value;
        if(uyeAd == "" || uyeSoyad == "" ||
            uyeYer == "" || meslek == "" ||
            mail == "" || kullanıcıAdi == "" ||
            sifre1 == "" || sifre2 == "")
            document.getElementById("hata").innerHTML = "Lütfen boş alan
bırakmayınız!"
        else if(mail.indexOf("@") < 0)
            document.getElementById("hata").innerHTML = "Lütfen geçerli

```

```

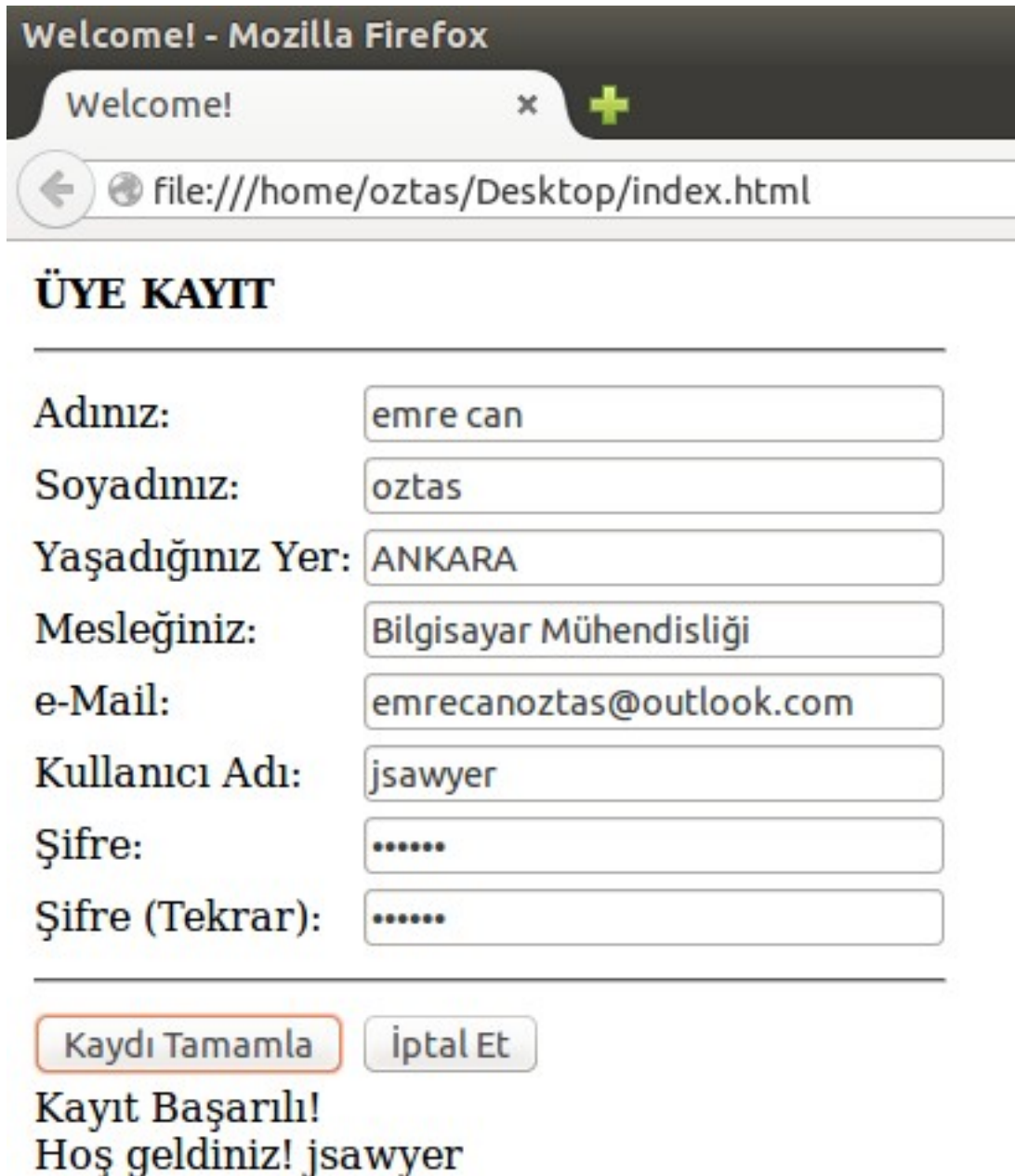
        bir mail adresi giriniz!";
        else if(sifre1.length < 6)
            document.getElementById("hata").innerHTML = "Şifre alanı en
az 6 karakter olmalıdır!";
        else if(sifre1 != sifre2)
            document.getElementById("hata").innerHTML = "Şifre alanları
uyuşmamaktadır!";
        else{
            /*
                burada kullanıcının bilgileri basari ile alingi icin,
                kullanıcının db'ye kayıt edilip, sayfanin yonlendirilmesi
                gerekiyor.
            */
            document.getElementById("hata").innerHTML = "Kayıt Başarılı!" +
"<BR>" + "Hoş geldiniz! " + kullanıcıAdi;
        }
    }
</script>
</head>
<body>
<form>
<table>
<tr>
<td colspan="2"><b>ÜYE KAYIT</b></td>
</tr>
<tr>
<td colspan="2"><hr></td>
</tr>
<tr>
<td><label>Adınız:</label></td>
<td><input type="text" name="ad" id="ad"/></td>
</tr>
<tr>
<td><label>Soyadınız:</label></td>
<td><input type="text" name="soyad" id="soyad"/></td>
</tr>
<tr>
<td><label>Yaşadığınız Yer:</label></td>
<td><input type="text" name="yer" id="yer"/></td>
</tr>
<tr>
<td><label>Mesleğiniz:</label></td>
<td><input type="text" name="meslek" id="meslek"/></td>
</tr>
<tr>
<td><label>e-Mail:</label></td>
<td><input type="text" name="mail" id="mail"/></td>
</tr>
<tr>
<td><label>Kullanıcı Adı:</label></td>
<td><input type="text" name="kadi" id="kadi"/></td>
</tr>
<tr>
<td><label>Şifre:</label></td>
<td><input type="password" name="sifre1" id="sifre1"/></td>
</tr>
<tr>
<td><label>Şifre (Tekrar):</label></td>
<td><input type="password" name="sifre2" id="sifre2"/></td>
</tr>

```



```
<tr>
<td colspan="2"><hr></td>
</tr>
<tr>
<td><input type="button" value="Kaydı Tamamla" onclick="kontrolEt()"/></td>
<td><input type="reset" value="İptal Et"/></td>
</tr>
<tr>
<td colspan="2">
<!-- innerHTML anahtar kelimesi herhangi bir etikette kullanılabilir.
      Biz bu örneğimizde label etiketini kullanalım.
-->
<label id="hata"></label>
</td>
</tr>
</table>
</form>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



The screenshot shows a Mozilla Firefox browser window with a single tab titled 'Welcome!'. The address bar displays the file path: file:///home/oztas/Desktop/index.html. The main content area features a registration form titled 'ÜYE KAYIT'. The form contains several input fields with the following labels and values: 'Adınız:' (emre can), 'Soyadınız:' (oztas), 'Yaşadığınız Yer:' (ANKARA), 'Mesleğiniz:' (Bilgisayar Mühendisliği), 'e-Mail:' (emrecanoztas@outlook.com), 'Kullanıcı Adı:' (jsawyer), 'Şifre:' (masked with dots), and 'Şifre (Tekrar):' (masked with dots). Below the form are two buttons: 'Kaydı Tamamla' (highlighted with a red border) and 'İptal Et'. At the bottom, a message reads 'Kayıt Başarılı! Hoş geldiniz! jsawyer'.

Yukarıdaki örneğimizin kaynak kodlarını indirip inceleyebilir, değiştirebilir ve kullanabilirsiniz. Daha önce de belirttiğimiz gibi sayfamız şuan için prototip seviyesindedir. Lakin belirlediğimiz işlevlerimizi yerine getirmektedir

Son örneğimizle birlikte anladık ki; bir web sayfasındaki herhangi bir gömülü HTML etiketini de değiştirebilir, amacımıza göre kullanabiliriz.

13.3 Elementleri CSS İle Şekillendirme

JavaScript ile sadece elementlerin değerlerine erişebilme veya bu elementlerin değerlerini değiştirme gerçekleştirilmez. Bu elementler, CSS ile yeniden şekillendirilebilir. Örneğin hata mesajlarını belirtmek için kırmızı rengi çok sık kullanılır. Veya onay için yeşil rengin kullanıldığı gibi.

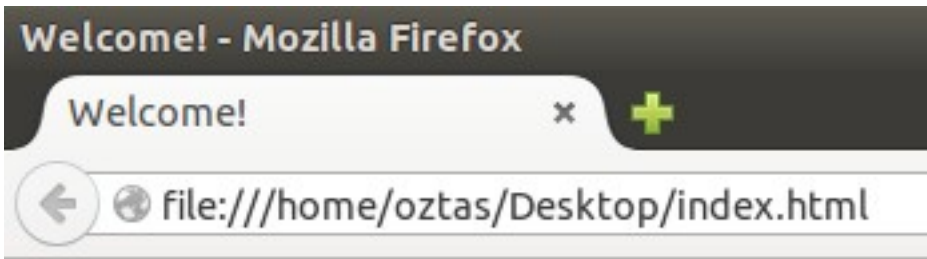
Herhangi bir HTML elementini, CSS ile şekillendirmek için aşağıdaki standart kullanım uygulanır.

```
document.getElementById(elementId).style.cssBilesini = deger;
```

CSS bileşenleri olduğu gibi JavaScript'te kullanılamaz. Bu bileşenlerin JavaScript karşılıkları vardır. Bu karşılıkları vermeden önce bir örnek yapalım ve CSS ile elementleri şekillendirmeye değinelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function red(){
        document.getElementById("p").style.color = "red";
    }
    function green(){
        document.getElementById("p").style.color = "green";
    }
    function yellow(){
        document.getElementById("p").style.color = "yellow";
    }
</script>
</head>
<body>
<p id="p">Yazı rengi</p>
<table>
<tr>
<td><input type="button" value="Red" onclick="red()"/></td>
<td><input type="button" value="Green" onclick="green()"/></td>
<td><input type="button" value="Yellow" onclick="yellow()"/></td>
</tr>
</table>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yazı rengi

Red

Green

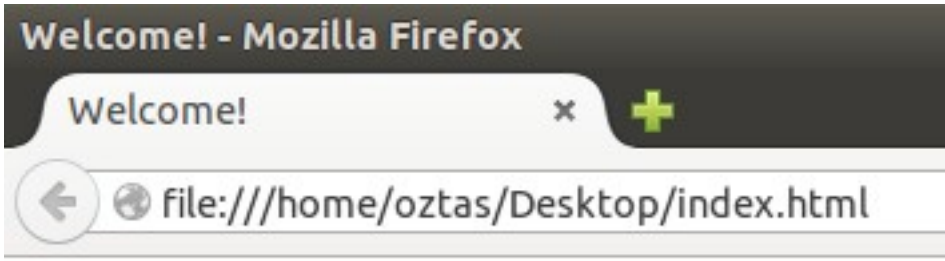
Yellow

Yukarıdaki ekran alıntısında da görüldüğü gibi üzerinde adı yazan herhangi bir butona tıkladığımız zaman `<p>` elementindeki yazının rengi değişmektedir.

JavaScript ile CSS kavramını daha iyi anlamak için bir örnek daha yapalım. Sayfamızda yine bir `<p>` elementi ve iki tane buton olsun. Bu butonlardan bir tanesi yazının font büyüklüğü arttırsın, diğeri de azaltsın. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    var s1 = 0;
    var s2 = 0;
    function buyult(){
        s1++;
        document.getElementById("p").style.fontSize = parseInt(s1) + "px";
    }
    function kucult(){
        s1--;
        document.getElementById("p").style.fontSize = parseInt(s1) + "px";
    }
</script>
</head>
<body>
<p id="p">Yazı büyüklüğü</p>
<table>
<tr>
<td><input type="button" value="+" onclick="buyult()"/></td>
<td><input type="button" value="-" onclick="kucult()"/></td>
</tr>
</table>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yazı büyüklüğü



Yukarıdaki ekran alıntısında da görüldüğü gibi; kendimiz yazının font büyüklüğünü ayarlayabiliyoruz.

JavaScript ile CSS işlemlerinde; .value ve .innerHTML anahtarlarının kullanımı gerekmez. Vermiş olduğumuz yapı ile hem kullanıcının doğrudan müdahale edebildiği hemde gömülü HTML elementlerinin CSS değerleri değiştirilebilir.

En başta söylediğimiz gibi JavaScript ortamında CSS bileşenleri farklı bir formatta yazılır. Şimdi bunları tablo şeklinde yazalım.

Bazı CSS bileşenlerinin JavaScript ortamındaki yazımları:

| CSS Bileşenleri | JavaScript Karşılığı |
|---------------------|----------------------|
| background | background |
| background-color | backgroundColor |
| background-image | backgroundImage |
| border-bottom | borderBottom |
| border-bottom-color | borderBottomColor |
| border-bottom-style | borderBottomStyle |
| border-bottom-width | borderBottomWidth |
| border-left-color | borderLeftColor |
| border-left-style | borderLeftStyle |
| border-left-width | borderLeftWidth |
| border-right-color | borderRightColor |
| border-right-style | borderRightStyle |
| border-right-width | borderRightWidth |
| border-top-color | borderTopColor |
| border-top-style | borderTopStyle |
| border-top-width | borderTopWidth |
| color | color |
| cursor | cursor |
| filter | filter |

| | |
|-------------------------------|---------------------------|
| font-family | fontFamily |
| font-size | fontSize |
| font-variant | fontVariant |
| font-weight | fontWeight |
| height | height |
| left | left |
| letter-spacing | letterSpacing |
| line-height | lineHeight |
| list-style | listStyle |
| list-style-image | listStyleImage |
| list-style-position | listStylePosition |
| list-style-type | listStyleType |
| margin-bottom | marginBottom |
| margin-left | marginLeft |
| margin-right | marginRight |
| margin-top | marginTop |
| padding-bottom | paddingBottom |
| padding-left | paddingLeft |
| padding-right | paddingRight |
| padding-top | paddingTop |
| text-align | textAlign |
| text-decoration | textDecoration |
| text-decoration: line-through | textDecorationLineThrough |
| text-decoration: none | textDecorationNone |
| text-decoration: overline | textDecorationOverline |
| text-decoration: underline | textDecorationUnderline |
| text-transform | textTransform |
| top | top |
| visibility | visibility |
| width | width |

13.4 Extras

Bu bölümde ekstra olarak document nesnesinin diğer fonksiyonları üzerinde duracağız. Bu fonksiyonlar yardımıyla web sayfasının çeşitli bilgileri okunabilmekte ve gereken bilgiler kolayca öğrenilebilmektedir.

document nesnesinin diğer fonksiyonları neler şimdi bunlara değinelim.

13.4.1 domain

domain fonksiyonu ile web sayfasının internet domain adresi öğrenilebilmektedir. Domain

fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
document.domain;  
// degiskene baglamayi unutmayalım.  
var dom = document.domain;
```

Şuna JavaScript'i kendi bilgisayarımızda kodluyoruz ve herhangi bir domain alanına sahip değiliz doğal olarak yukardaki fonksiyonunun döneceği değer “boş” olacaktır. Fakat bilgisayarınızda herhangi bir web server kurulu ise bu fonksiyonu orada deneyebilirsiniz. Örneğin; şuan benim bilgisayarımda hem php server hem de apache web server kurulu. Her iki server'da da denediğim zaman dönen değer: “local” olmaktadır.

13.4.2 lastModified

lastModified fonksiyonu, web sayfamızın en son ne zaman düzenlendiğini göstermektedir. lastModified fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
document.lastModified;  
// degiskene baglamayi unutmayalım.  
var dlm = document.lastModified;
```

LastModified fonksiyonunu kullanarak daha önce hazırlamış olduğunuz herhangi bir web sayfasını en son ne zaman düzenlemiş olduğunuzu görebilirsiniz. Oldukça kullanışlı bir fonksiyon.

13.4.3 location

location fonksiyonu, web sayfamızın bulunduğu URL'i vermektedir. Location fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
document.location;  
// degiskene baglamayi unutmayalım.  
var lcal = document.location;
```

Bu kitap boyunca; sizi bilmem ama ben hazırlamış olduğumuz web sayfamızı Desktop (Masaüstü)'a kaydettim. Location fonksiyonu ile bu kaydettiğimiz sayfamızın adresini alabiliriz. Eğer web sayfamız herhangi bir web alanına sahip olsaydı o zaman da o web alanının URL'sini getirecekti.

Web sayfamızın URL'si aşağıdaki gibi olacaktır.

```
file:///home/oztas/Desktop/index.html
```

13.4.4 title

title fonksiyonu bildiğimiz <title></title> etiketleri arasındaki ifadeyi verir. Title fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
document.title;  
// degiskene baglamayi unutmayalım.  
var baslik = document.title;
```

Web sayfalarımıza hep “Welcome!” başlığını vermiştik. Yukarıdaki fonksiyonu yazıp çalıştırdığımız zaman da bu “Welcome!” yazısını verecektir.

13.4.5 writeln()

`writeln()` fonksiyonu, `write()` fonksiyonu ile aynı görevdedir. Hatırlarsanız; `write()` fonksiyonu, değerleri art arda yazdırmaktaydı. `writeln()` fonksiyonu ise belirtilen değeri yazdıktan sonra bir boşluk “ ” bırakmaktadır. `writeln()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
document.writeln(yazdirilacak ifade);
```

BÖLÜM 14:

Browser

Bu bölümde Web Browser (Web Tarayıcı)'lar üzerinde duracağız. Bildiğiniz gibi şuan piyasada pek çok tarayıcı bulunmakta. Örnek verecek olursak; Internet Explorer, Windows Edge (Windows 10 ile birlikte; Internet Explorer'ın yerini alan yeni tarayıcı), Yandex, Chrome, Chromium (Chrome'un GNU / Linux uyarlaması), Firefox ve Opera. Bunlar bilinen güncel tarayıcılar. Bunların dışında GNU / Linux ve Mac ortamında kullanılan farklı tarayıcıları da saymaya kalkarsak; sayfalar yetmez. O yüzden işi tadında bırakalım ve konumuza dönelim.

Piyasada bir çok tarayıcı var ve her tarayıcı bir değil. Yani her tarayıcının kendine has özellikleri var. Tabi hangi tarayıcıyı kullanmakta, kullanıcının tercihi kalmış. Bu bölümde; tarayıcıların özelliklerini öğreneceğiz. Bunun dışında sayfa yönlendirilmesi çeşitli yararlı konular üzerinde de durmaya çalışacağız.

Konularımızı üç nesne üzerinden göreceğiz. Bunlar: navigator, window ve screen nesnesidir.

14.1 navigator

navigator, bir nesnedir. Bu nesnenin özellikleri ve fonksiyonları ile bir tarayıcının çeşitli özelliklerini öğrenebilmekte ve bu özelliklere göre çeşitli işlemler gerçekleştirebilmekteyiz. navigator nesnesinin özellikleri nelermiş şimdi buna bakalım.



navigator nesnesinin özelliklerini ve fonksiyonlarını web sayfamızda kullanarak, sayfamız ziyaretçilerinin tarayıcıları hakkında bilgi alabilmekteyiz. navigator nesnesiyle birlikte çeşitli tip ve milletten insanlara göre hizmetler verilebilir. Örneğin web sayfasının kullanım dili gibi...

14.1.1 appName

appName özelliği ile bir tarayıcının Application Code Name (Uygulama Kod Adı)'ni öğrenebiliriz. appName özelliğinin standart kullanımı aşağıdaki gibidir.

```
navigator.appName;  
// bir degiskene baglamayi unutmayalım.  
var browser = navigator.appName;
```

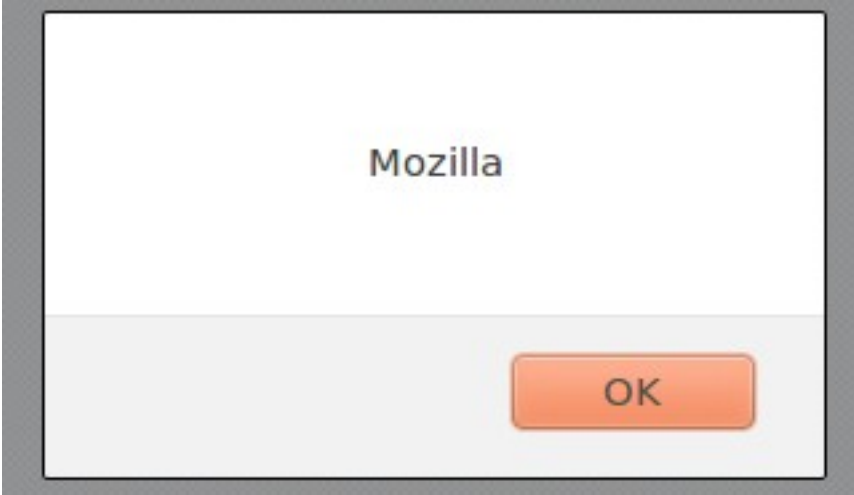
appName ile basit bir örnek yapalım. Bakalım tarayıcımızın kod adı neymiş öğrenelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome!</title>  
<script type="text/JavaScript">  
    function browser(){  
        alert(navigator.appName);
```



```
}
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi benim kullandığım tarayıcı: Firefox Mozilla. Eğer başka tarayıcılarda denerseniz, denediğiniz o tarayıcının kod adı dönecektir.

Her tarayıcının kod ismi aynı değildir. Örneğin: Microsoft Internet Explorer: Internet Explorer veya Google Chrome: Chrome'dır.

14.1.2 appVersion

appVersion özelliği ile bir tarayıcının Application Version (Uygulama Versiyonu)'ni öğrenebiliriz. appVersion özelliğinin standart kullanımı aşağıdaki gibidir.

```
navigator.appVersion;
// bir degiskene baglamayi unutmayalım.
var browserApp = navigator.appVersion;
```

appVersion özelliği ile basit bir örnek yapalım. Tarayıcımızın version numarası neymiş öğrenelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<script type="text/JavaScript">
    function browser(){
        alert(navigator.appVersion);
    }
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi Firefox Mozilla 5.0 (X11) sürümünü kullanmaktayım. Bu sürüm Ubuntu 14.04 LTS sürümünde varsayılan olarak gelmektedir.

14.1.3 language

language özelliği, tarayıcının kullanım dilini verir. language özelliğinin standart kullanımı aşağıdaki gibidir.

```
navigator.language;  
// bir degiskene baglamayi unutmayalım.  
var browserLang = navigator.language;
```

language özelliği ile basit bir örnek yapalım. Tarayıcımızın kullanım dilini öğrenelim. Örneğimiz aşağıdaki gibi olacaktır.



Yukarıdaki ekran alıntısında da görüldüğü gibi tarayıcımın dili ingilizce ayrıca Amerikan ingilizcesi, dikkatiniz çekerim (Havaya bak!).

language özelliği kullanılarak, web sayfanızın ziyaretçilerinin tarayıcı dilini öğrenebilir ve eğer o dile destek veriyorsanız sayfanızı yönlendirebilirsiniz. Sayfa yönlendirme işlemlerini window bölümünde detaylı olarak inceleyeceğiz.

14.1.4 plarform[]

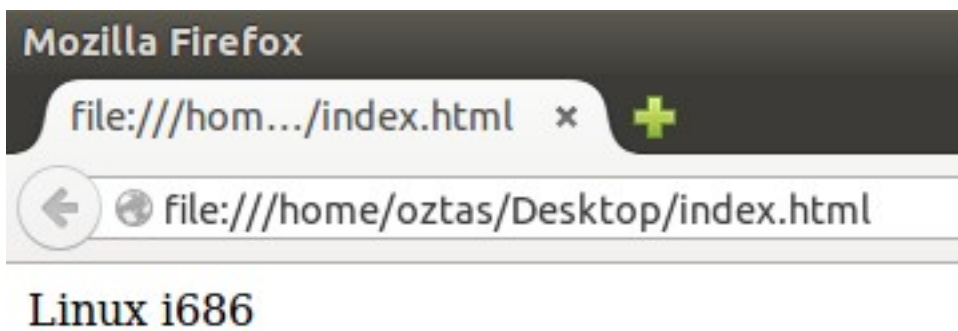
plarform[] özelliği ile tarayıcının çalıştığı işletim sisteminin adını öğrenebiliriz. plarform[] özelliğinden dönen değer bir dizi olacaktır. Bunu da unutmayalım. plarform[] özelliğinin standart kullanımı aşağıdaki gibidir.

```
navigator.platform[];
```

plarform[] özelliği ile basit bir örnek yapalım ve sistemimiz neymiş öğrenelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<script type="text/JavaScript">
    function browser(){
        // platform ozelliginin uzunlugunu alalim.
        var sayac = navigator.platform.length;
        // ekrana yazdiralim
        for(var i = 0; i < sayac; i++)
            document.write(navigator.platform[i]);
    }
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki özelliğimize ait olan ekran çıktısını alalım.



Ekran çıktımızda da görüldüğü üzere; şuan benim kullandığım sistem: GNU / Linux ve işlemcim i686 yani 32 bit.

14.1.5 userAgent

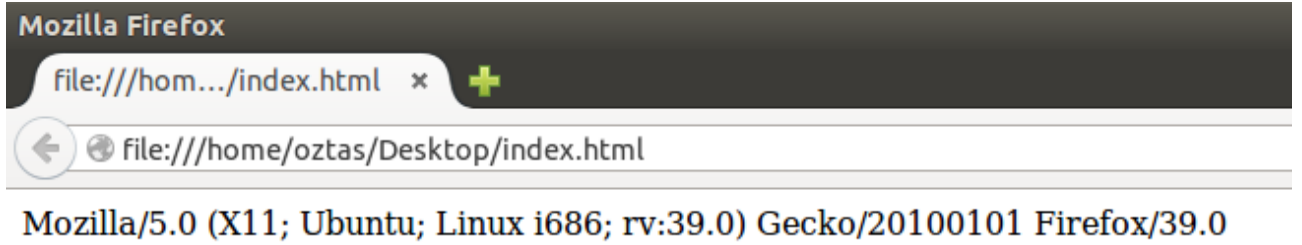
Tarayıcı, herhangi bir server'a istek gönderdiği zaman özelliklerini bu server'a gönderir ya da amiyane bir tabirle ben buyum diye kendini tanıtır. userAgent özelliği ise işte tarayıcının server'a gönderdiği bilgileri verir. userAgent özelliğinin standart kullanımı aşağıdaki gibidir.

```
navigator.userAgent;
// bir degiskene baglamayi unutmayalim.
var brUser = navigator.userAgent;
```

userAgent özelliği ile basit bir örnek yapalım ve tarayıcımızın özellikleri neler bunları öğrenelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<script type="text/JavaScript">
    function browser(){
        var userAgent = navigator.userAgent;
        document.write(userAgent);
    }
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Ekran alıntısında da görüldüğü gibi hem tarayıcımız hem de sistemimiz hakkında tüm bilgiyi userAgent özelliği ile alabiliriz.

Şuan sistemimde ikinci bir tarayıcı olarak Chrome'un GNU / Linux için uyarlaması olan Chromium da kurulu. Kullanmayı pek tercih etmesem de yinede bazı durumlarda tasarlamış olduğum web sayfalarının görünümlerini kontrol edebiliyorum. userAgent özelliğini bir de Chromium'da deneyelim. Bakalım nasıl bir sonuç verecek. Chromium, ekran çıktısı aşağıdaki gibidir.



Ekran alıntısında da görüldüğü gibi Chromium'da kendine özel imzasıyla server'a kendini belirtiyor.

14.2 window

window, bir nesnedir. Bu nesnenin, özellikleri ve fonksiyonları kullanılarak; farklı bir pencere açılabilir. Ya da tarayıcıya ait olan özellikler alınabilir. Şimdi window nesnesinin bazı kullanışlı özelliklerini inceleyelim.

14.2.1 open()

open() özelliği ile yeni bir pencere açılabilir. Örneğin; bazı web sitelerine bağlandığımızda canımızı sıkran reklam sayfaları açılır. İşte bu reklam sayfaları JavaScript ortamında open() fonksiyonu ile kontrol edilebilir. open() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
window.open(acilacak sayfa);
```

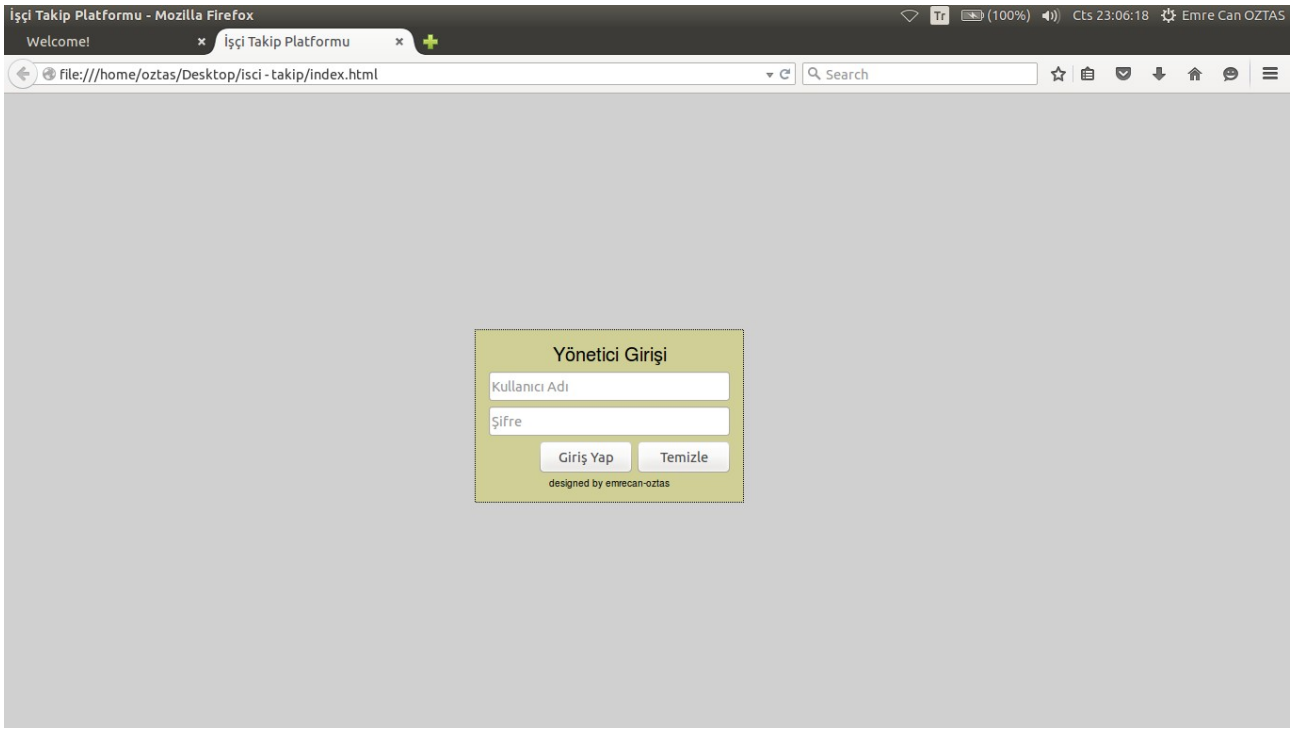
open() fonksiyonu ile basit bir örnek yapalım. Sayfamız açıldığı anda başka bir sayfa açsın. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<script type="text/JavaScript">
    function browser(){
        window.open("http://www.google.com");
    }
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki örneğimizi çalıştırdığımızda; yeni bir pencere açılacak ve www.google.com adresi gösterilecektir. Burada size daha farklı birşey göstermek istiyorum. open() fonksiyonunu kullanarak bilgisayarımızdaki herhangi bir web sayfasını da açabiliriz. Aşağıdaki örneğimiz bununla ilgili.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<script type="text/JavaScript">
    function browser(){
        window.open("file:///home/oztas/Desktop/isci%20-
%20takip/index.html");
    }
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Yukarıdaki ekran alıntısını incelediğimizde ekstra olarak `open()` fonksiyonunda parametre olarak belirttiğimiz adresteki web sayfası açıldı.

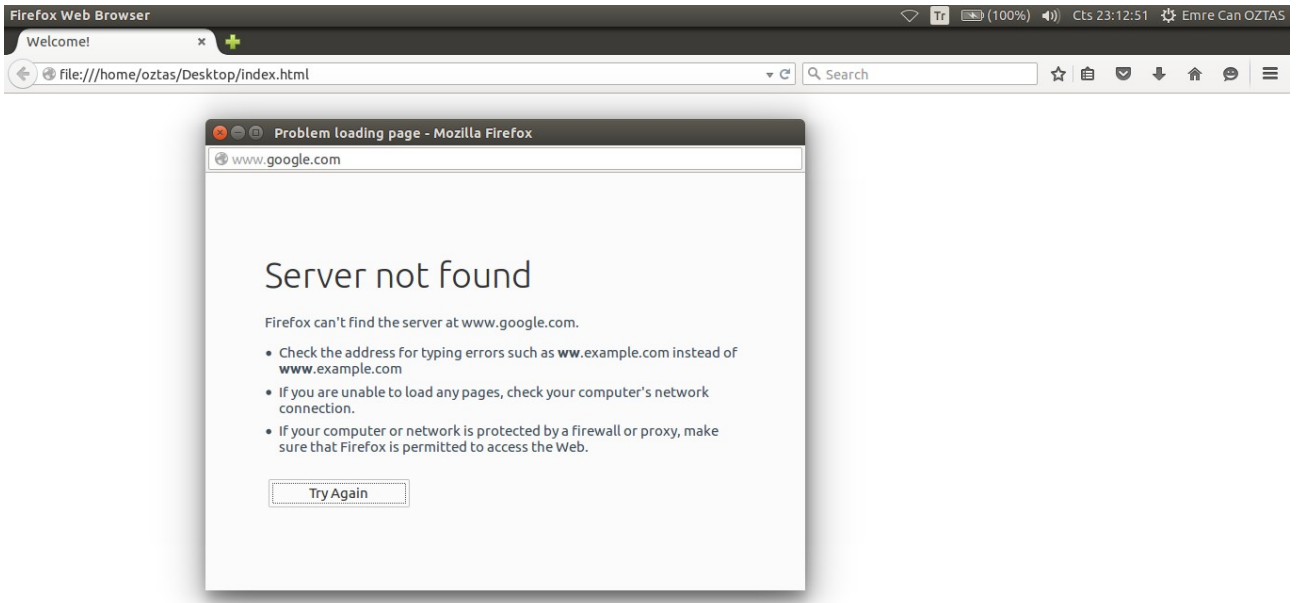
`open()` fonksiyonu sadece tek parametre alabileceği gibi farklı parametreler de alabilir. Aldığı bu parametrelerle daha kullanışlı bir yapıya sahip olur.

```
window.open(acilacak sayfa, pencere adi, pencere ozellikleri);
```

Yukarıdaki birden fazla parametrelili `open()` fonksiyonunu görmektesiniz. Açılacak olan sayfa dışında bu açılacak olan sayfaya bir isim verilebilir ve bu sayfanın özellikleri belirlenebilir. Yukarıdaki birden fazla parametrelili `open()` fonksiyonunu kullanarak bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<script type="text/JavaScript">
    function browser(){
        window.open("http://www.google.com", "Hello!", "menubar=false,
toolbar=false, scrollbars=false, width=400, height=300, resizable=yes");
    }
</script>
</head>
<body onload="browser()">
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Şuan ağ bağlantım kapalı olduğu için belirttiğim www.google.com adresine bağlanamadı. Fakat burada size bahsetmek istediğim şey başka. Yukarıdaki ekran alıntısını incelediğimizde tarayıcıdan ayrı bir pencere açıldı ve belirttiğim sayfa açıldı. Bunu neye göre belirledik? Bunu open() fonksiyonuna gönderdiğimiz üçüncü parametreye yani “pencere özelliklerine” göre belirledik.

open() fonksiyonu ile açılacak olan pencerenin özelliklerini belirtmek için kullandığımız parametreleri tanıyalım.

| Pencere Özellikleri | Açıklama |
|---------------------|--|
| menubar | Tarayıcıda; menülerin bulunduğu çubuktur. No veya false değeri alırsa bu çubuk görünmez, yes veya true değerini alırsa görünür. |
| toolbar | Tarayıcıda; geri, ileri v.s gibi araçların bulunduğu çubuktur. No veya false değeri alırsa bu çubuk görünmez, yes veya true değerini alırsa görünür. |
| location | Kısaca URL adres satırıdır. No veya false değeri alırsa bu satır görünmez, yes veya true değerini alırsa görünür. |
| status | Durum çubuğudur. No veya false değeri alırsa bu çubuk görünmez, yes veya true değerini alırsa görünür. |
| scrollbars | Kaydırma çubuklarıdır (Yatay / Dikey). No veya false değeri alırsa bu çubuk görünmez, yes veya true değerini alırsa görünür. |
| resizable | Açılan pencerenin boyutlarının değiştirilmesidir. No veya false değeri alırsa pencerenin boyutları değiştirilemez, yes veya true değerini alırsa değiştirilebilir. |
| width | Pencerenin genişliğidir. Herhangi bir değer verilerek pencerenin genişliği değiştirilebilir. |
| height | Pencerenin boyudur. Herhangi bir değer verilerek pencerenin boyu değiştirilebilir. |
| left | Pencerenin, sol taraftan uzaklığını belirtir. Herhangi bir değer verilerek pencerenin soldan kaç piksel içerde olacağı değiştirilebilir. |
| top | Pencerenin, yukarı taraftan uzaklığını belirtir. Herhangi bir değer verilerek pencerenin yukarıdan kaç piksel içerde olacağı değiştirilebilir. |

14.2.2 close()

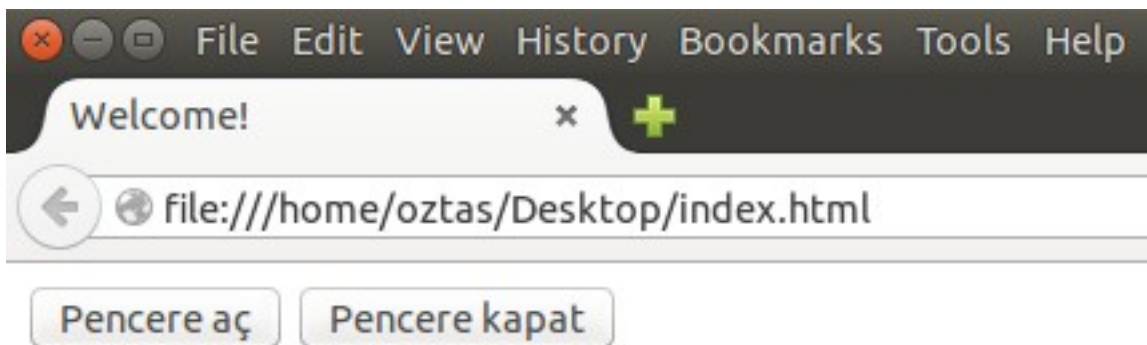
open() ile belirttiğimiz herhangi bir web sitesini açabiliriz. close() ile de açılan bu pencereyi kapatabiliriz. close() fonksiyonunun kullanımı biraz farklıdır. Aşağıdaki örnek kullanıma bakalım.

```
// window.open() islemini bir degiskene baglamaliyiz.  
var pencere = window.open("http://www.google.com");  
// actigimiz pencereyi kapatalim.  
pencere.close()
```

close() fonksiyonunun kullanımını daha iyi anlamak için basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome!</title>  
<meta charset="utf-8">  
<script type="text/JavaScript">  
    var adres = "http://www.google.com";  
    var pencere;  
    function pencereAc(){  
        pencere = window.open(adres);  
    }  
    function pencereKapat(){  
        pencere.close();  
    }  
</script>  
</head>  
<body>  
<input type="button" value="Pencere aç" onclick="pencereAc()"/>  
<input type="button" value="Pencere kapat" onclick="pencereKapat()"/>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



“Pencere aç” butonuna tıkladığımız zaman belirlediğimiz URL açılacaktır. “Pencere kapat” butonuna tıkladığımız zamanda açılan URL kapatılacaktır.

14.2.3 print()

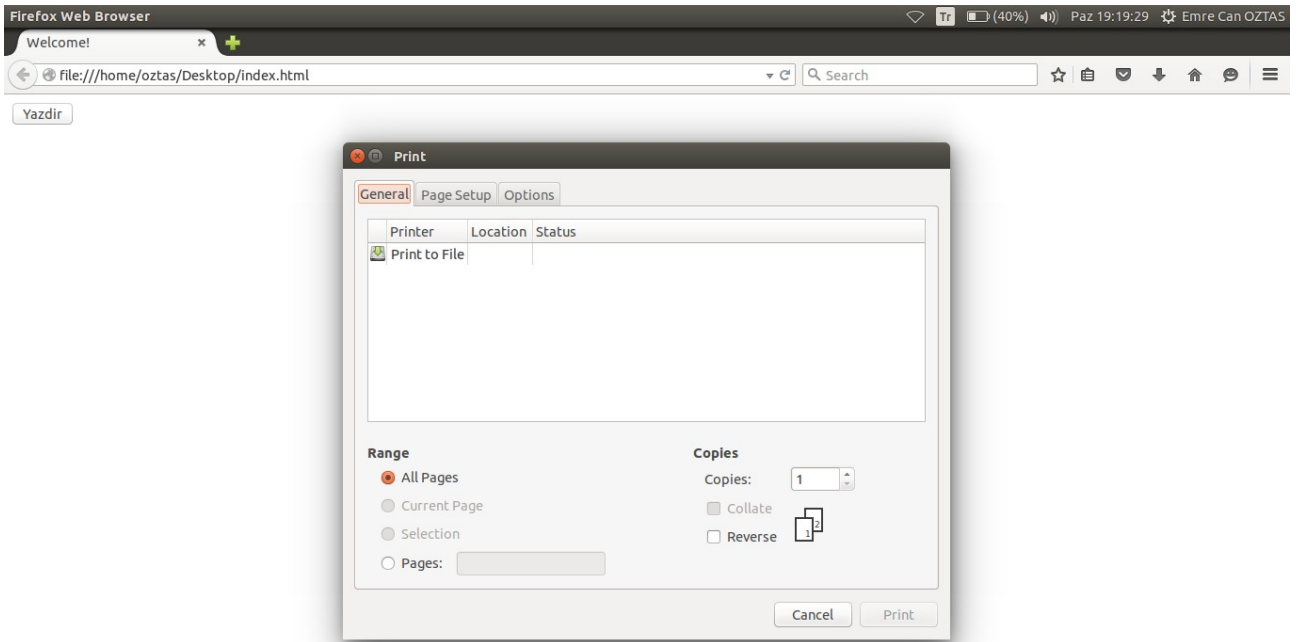
print(), window nesnesine ait bir fonksiyon olup sayfanın yazdırılmasını sağlar. print() fonksiyonunun kullanımı aşağıdaki gibidir.

```
window.print();
```

print() fonksiyonunu kullanarak bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function yazdir(){
        window.print();
    }
</script>
</head>
<body>
<input type="button" value="Yazdır" onclick="yazdir()" />
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Sayfamıza eklediğimiz “Yazdır” butonuna tıkladığımız zaman; print() penceresi açılmış oldu. Şuan benim bilgisayarım da herhangi bir tanımlı bir yazıcı olmadığı için, gelen pencerede tanımlı bir yazıcı yok.

14.2.4 location

Hiç Mad Max filmini izlediniz mi bilmiyorum. İzlemediyseniz izleyin. Toplam 3 serisi var. iste Mad Max'in ilk filminde; bir sahnede polisler bir arabanın peşine düşüyorlar ve arabayı kullanan kanun kaçağını yakalamaya çalışıyorlardı. İlginçtir kanun kaçağı kendisine “Gece Sürücüsü” diyordu. Şimdi bu isimde bir sitemiz olduğunu düşünelim (Bir site için oldukça kötü bir isim). Böyle bir site var mı bilmiyorum. Varsa da bizi affetsinler, onlarla bir alakamız veya sataşmak gibi bir kaygımız yok. Bu

sitemizi başka bir URL'ye yönlendirmek istiyoruz diyelim. location özelliği tamda bu noktada imdadımıza yetişir. Toparlarsak; location özelliği ile sayfa yönlendirilmesi yapılır. location özelliğinin standart kullanımı aşağıdaki gibidir.

```
window.location = "http://www.google.com";
```

Eğer yüksek hit alan bir siteniz varsa ve bu siteyi yönlendirmek istiyorsanız dikkatli olmanızı tavsiye ederim. Çünkü google bir süre sonra yönlendirilen bu siteleri banlayabiliyor.

location özelliği ile basit örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
<script type="text/JavaScript">
    function yonlendir(){
        window.location = "http://www.google.com";
    }
</script>
</head>
<body>
<input type="button" value="Yönlendir" onclick="yonlendir()" />
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Sayfamızdaki “Yönlendir” butonuna tıkladığımız zaman sayfamız; betirlen adrese yönlendirilecektir. Örneğimizde butona bağlı olarak kullandık. Eğer hakikaten sayfanızı yönlendirmek istiyorsanız; unload olayı ile birlikte kullanmalısınız.

Location'ın kendine ait özellikleri vardır. Bu özellikleri kullanarak sayfa hakkında çeşitli bilgiler edinilebilir. Bu özellikler aşağıdaki gibidir.

| Location Özellikleri | Açıklama |
|----------------------|---|
| location.host | Web sayfasının barındırıldığı alanın özelliklerini verir. Bu komutla herhangi bir host alanında bulunmayan sayfa boş değer alacaktır. Örneğin Apache Tomcat üzerinde denediğimizde: localhost:8080 çıktısını verecektir.

Kullanımı:
window.location.host; |
| location.hostname | Web sayfasının barındırıldığı alanın adını verir. Bu komutla herhangi bir host alanında bulunmayan sayfa boş değer alacaktır. Herhangi bir local web server'da denediğimizde: localhost çıktısını verecektir.
Kullanımı:
window.location.host; |
| location.href | Web sayfasının barındırıldığı alanın adresini verir. Bu komutla herhangi bir host alanında bulunmayan sayfanın, bulunduğu local dizinin adresi döner.

Kullanımı:
window.location.href; |
| location.pathname | Web sayfası, barındırıldığı alanda, hangi dizinde yer alıyorsa o dizinin adresini verir.

Kullanımı:
window.location.pathname; |
| location.port | Web sayfası, hangi port üzerinde hizmet veriyorsa; o port numarasını döndürür. Herhangi bir host alanında barındırılmayan web sayfasının herhangi bir port değeri dönmez. Örneğin Apache Tomcat üzerinde denediğimizde: 8080 değeri dönmektedir.

Kullanımı:
window.location.port; |
| location.protocol | Web sayfasının hangi protokol üzerinde çalışıyorsa; o protokol'un adını verir. Örneğin herhangi bir server üzerinde çalıştırılmayan bir web sayfası file değerini döndürecek. Ancak Apache Tomcat üzerinde; sayfayı çalıştırdığımızda http çıktısını almaktayız.

Kullanımı:
window.location.protocol; |

14.3 screen

screen nesnesi ile web sayfasını ziyaret eden kullanıcı tarayıcısının; ekran çözünürlüğü, ekran boyutu ve renk derinliği gibi özellikleri öğrenilir. cscreen nesnesini sahip olduğu sınırlı özelliklerini sırasıyla incelemeye çalışalım.

14.3.1 width

Tarayıcının açık olan pencere genişliğini veren özelliktir. width özelliğinin kullanımı aşağıdaki gibidir.

```
screen.width;
// bir degiskene baglamayi unutmayalim.
```

```
var scW = screen.width;
```

width özelliği ile basit örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    document.write(screen.width);
</script>
</body>
</html>
```

14.3.2 height

Tarayıcının açık olan pencere boyunu veren özelliktir. height özelliğinin kullanımı aşağıdaki gibidir.

```
screen.height;
// bir degiskene baglamayi unutmayalım.
var scH = screen.height;
```

height özelliği ile basit örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    document.write(screen.height);
</script>
</body>
</html>
```

14.3.3 colorDepth

Tarayıcının açık olan pencerenin renk derinliğini veren özelliktir. colorDepth özelliğinin kullanımı aşağıdaki gibidir.

```
screen.colorDepth;
// bir degiskene baglamayi unutmayalım.
var scD = screen.colorDepth;
```

colorDepth özelliği ile basit örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
```

```
<script type="text/JavaScript">
    document.write(screen.colorDepth);
</script>
</body>
</html>
```

14.3.4 availWidth

Tarayıcı penceresinin, kullanılabilir ekran çözünürlüğünün genişliğini verir. `availWidth` özelliğinin kullanımı aşağıdaki gibidir.

```
screen.availWidth;
// bir degiskene baglamayi unutmayalim.
var scaW = screen.availWidth;
```

`availWidth` özelliği ile basit örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    document.write(screen.availWidth);
</script>
</body>
</html>
```

14.3.5 availHeight

Tarayıcı penceresinin, kullanılabilir ekran çözünürlüğünün boyunu verir. `availHeight` özelliğinin kullanımı aşağıdaki gibidir.

```
screen.availHeight;
// bir degiskene baglamayi unutmayalim.
var scaH = screen.availHeight;
```

`availHeight` özelliği ile basit örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    document.write(screen.availHeight);
</script>
</body>
</html>
```

BÖLÜM 15:

String

JavaScript ortamında; karakter bazlı tanımlanan herhangi bir değişken `String` bir yapıda olur, bunu daha önceki konularımızda görmüştük. Ama yine de hatırlamak için küçük bir örnek yapalım. Değişkenlerin veri tiplerini `typeof` komutuyla elde edebiliyorduk.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var n1 = "Turkey";
    var n2 = 'T';
    document.write(typeof n1);
    document.write(typeof n2);
</script>
</body>
</html>
```

Yukarıdaki kodlarımızın ekran çıktılarını aldığımız zaman `n1` ve `n2` değişkenlerinin `String` yapıda olduğunu görebiliriz.

JavaScript ortamında `String` bir değişken tanımlamak için iki farklı yol mevcuttur. Aşağıdaki yollardan herhangi birini kullanarak `String` tipinde bir değişken tanımlaması yapabiliriz. Bu yollardan ilkinin bu zamana kadar kullandık.

```
/* var stringA;
   seklinde bir tanımlama undefined'tir.
   yani herhangi bir tipi yoktur. */
var stringA = "";
var stringB = new String();
```

İkinci yol olarak tanımlanan `String` tipi `Object` türündedir. Ayrıca aşağıdaki gibi tanımlanan değişkene ilk değer ataması yapılabilir.

```
var stringDegisken = new String("Merhaba!");
```

Buradan hareketle; varmak istediğimiz nokta `String` tipinin değişik fonksiyonları vardır. Bu hazır fonksiyonlar yardımıyla değişik işlemler yaptırabiliriz. Yani ekstra fonksiyon yazmaya gerek kalmaz. Şimdi `String` tipindeki bir değişkenin sahip olduğu bazı fonksiyonları görelim. Bu fonksiyonlar ile değişik örnekler yapalım ve `String` yapısını anlamaya çalışalım.

15.1 charAt()

`charAt()` fonksiyonu belirtilen indisteki karakteri elde etmek için kullanılır. Standart kullanımı

aşağıdaki gibidir.

```
stringDegiskenAdi.charAt(index);
```

Yukarıdaki ifadeyi herhangi bir değişkene bağlı olarak kullanmaz isek ya da ekrana yazdırmaz isek charAt() fonksiyonu ile ulaşılan karakter havada kalacaktır. Yani uçup gider, bu karaktere ulaşamayız. O yüzden herhangi bir değişken ile kullanmamız daha doğru olacaktır.

```
degiskenAdi = stringDegiskenAdi.charAt(index);
```

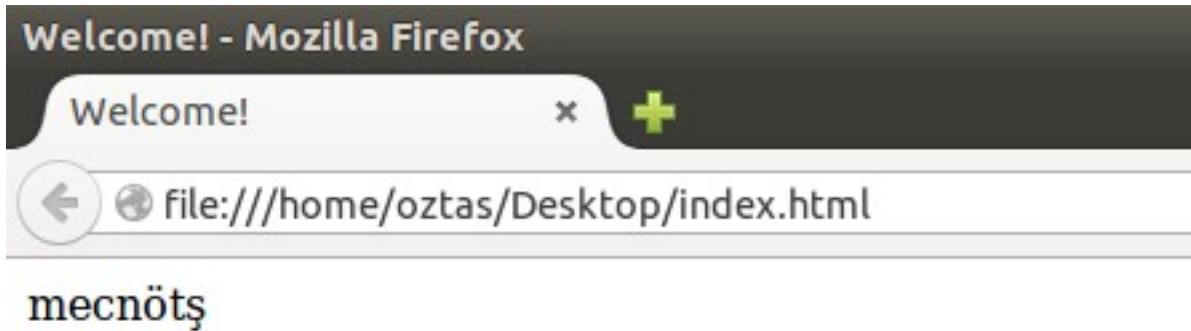
charAt() fonksiyonu ile bir örnek yapalım. Örneğin kullanıcıdan aldığımız bir ifadenin tek sayı olan index numarasındaki değerleri ekrana yazdıralım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var ad = prompt("Adınız: ");
    for(var i = 0; i< ad.length; i++){
        if(i % 2 != 0) document.write(ad.charAt(i));
    }
</script>
</body>
</html>
```



String tipindeki bir değişkenin uzunluğunu length fonksiyonu ile ulaşabiliyorduk, hatırlayalım.

Yukarıdaki örneğimize ait ekran çıktısını alalım. prompt ile gelen soruya herhangi bir ifadeyi yazabilirsiniz. Ben adımlı yazacağım.



Yukarıdaki örnek ekran çıktımızı inceleyelim. Ben “emre can ö z t a ş” olarak giriş yapmıştım. charAt() fonksiyonu ile tek sayı indisteki ekrana yazdırdık.



String tipindeki ifadeler bir Array (Dizi) yapısındadır. Array'lerde ilk indis 0 ile başlar, hatırlayalım. String ifadelerde de durum aynıdır.

15.2 charCodeAt()

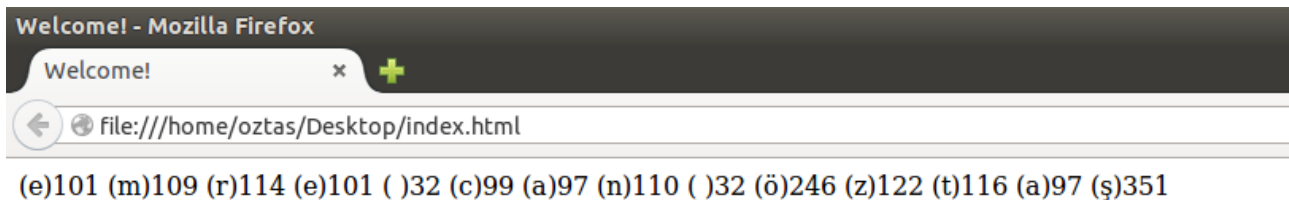
charAt() fonksiyonu belirtilen indisteki karakteri veriyordu. charCodeAt() fonksiyonu ise belirtilen indisteki değerin Unicode karşılığını vermektedir. Unicode, bildiğiniz gibi çeşitli dilleri içeren karakterler topluluğudur. 65536 adet unicode karakter vardır. Detaylı bilgi için www.unicode.org adresine bakabilirsiniz. charCodeAt() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
stringDegiskenAdi.charCodeAt(index);
```

charAt() bölümünde yaptığımız örneğimizi revize edip charCodeAt() fonksiyonu için uyarlayalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var ad = prompt("Adınız: ");
    for(var i = 0; i < ad.length; i++)
        document.write(" (" + ad.charAt(i) + ") " + ad.charCodeAt(i));
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait ekran çıktımızı alalım. Ben yine prompt ile gelen soruya “emre can öztaş” olarak cevap vereceğim.



Yukarıdaki ekran çıktısında görüldüğü gibi çeşitli karakterlerin unicode karşılığını charCodeAt() fonksiyonu ile elde ettik.

15.3 concat()

concat() fonksiyonu belirtilen iki veya daha fazla String ifadeyi birleştirir. concat aynı zamanda concatenation kelimesinin kısaltılmasıdır. concatenation kelimesi hadd-i zatinde bağlamak, birleştirmek anlamına gelmektedir. Buraya kadar olan bölümde String ifadeleri birleştirme işlemini + operatörüyle gerçekleştirmiştik. concat() fonksiyonun standart kullanımı aşağıdaki gibidir.

```
ifadeN.concat(ifade1, ifade2, ...);
```

concat() fonksiyonu ile küçük bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
```

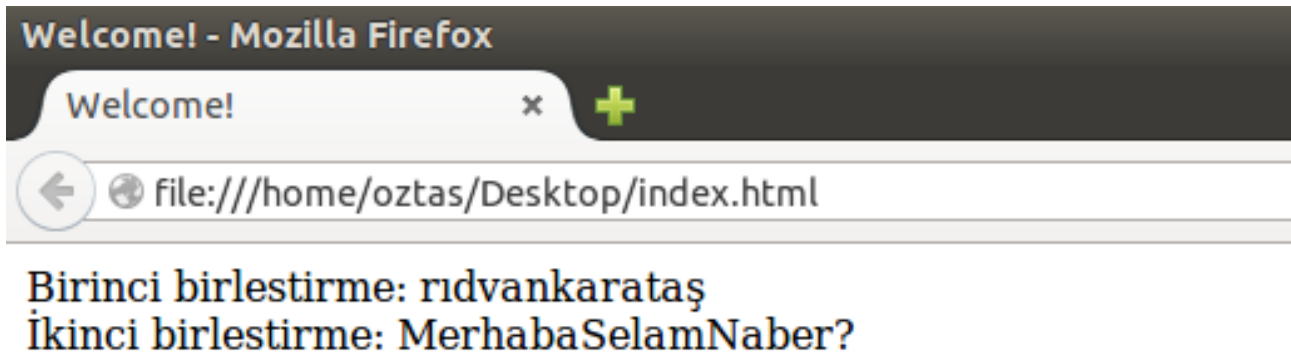


```

</head>
<body>
<script type="text/JavaScript">
    // 2 ifadenin birlestirilmesi
    var n1 = "rıdvan";
    var n2 = "karataş";
    n1 = n1.concat(n2)
    document.write("Birinci birlestirme: " + n1 + "<BR>");
    // 3 ifadesinin birlestirilmesi
    var n4 = "Merhaba";
    var n5 = "Selam";
    var n6 = "Naber?";
    n4 = n4.concat(n5, n6);
    document.write("İkinci birlestirme: " + n4 + "<BR>");
    // daha fazla ifade de birlestirilebilir
</script>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi ifadelerimizi concat() fonksiyonu ile kolaylaca birleştirdik.

15.4 indexOf()

indexOf() fonksiyonu verilen karakteri veya bir kelimeyi, belirtilen indis numarasında başlayarak, ilgili String ifade içerisinde aramaya işlemi yapar. Eğer aranan karakter veya kelime bulunamazsa -1 değerini döndürür. Aranan karakter veya kelime bulunduğunda o karakterin veya kelimenin ilk indis numarasını döndürür. Burada dikkat edilecek olan nokta; indexOf() ifadesi, aradığı karakteri veya kelimeyi bulduğu an o ilk karakterin indisini verir. Örneğin emre kelimesinde e karakterini arattığımızda ilk e karakterinin indis numarasını verir. Diğer e harfinin indis numarasını vermez. IndexOf() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
kelime.indexOf("aranacakKarakter");
```

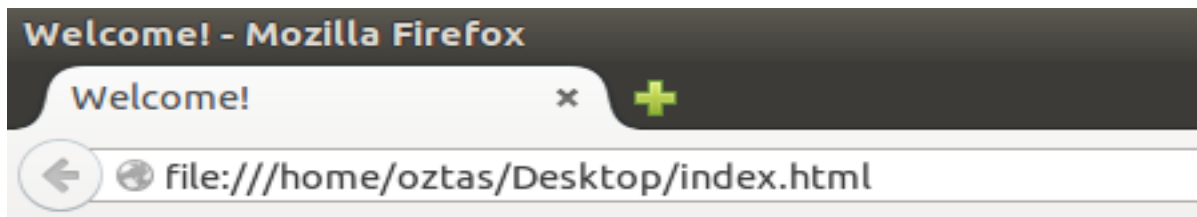
indexOf() fonksiyonu yukarıdaki gibi kullanıldığı zaman belirtilen karakter veya kelimeyi ilk karakterden başlayarak arama işlemini gerçekleştirir. Bunun dışında indexof() fonksiyonuna doğrudan başlangıç indisi atanarak, hangi karakterden itibaren arama yaptırılmak isteniyorsa gerçekleştirilebilir. Bunun için indexOf() fonksiyonunun aşağıdaki gösterimdeki kullanımını uygulamak gerekir.

```
kelime.indexOf("aranacakKarakter", indexBaslangicNo);
```

indexOf() fonksiyonu ile basit bir örnek yapalım. Kullanıcıdan bir kelime, aranacak karakter veya kelime ve başlangıç indisini alalım. Kullanıcının aranmasını istediği karakter veya kelime var mı bakalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var sayac = 0;
    while(sayac < 3){
        document.write(sayac + "." + "<BR>");
        var kelime = prompt("Kelime giriniz: ");
        var kkel = prompt("Aranan Karakter / Kelime: ");
        var indexNo = prompt("Başlangıç index: ");
        var durum = kelime.indexOf(kkel, indexNo);
        document.write("Kelime: " + kelime + "<BR>");
        document.write("Aranan Karakter / Kelime: " + kkel + "<BR>");
        document.write("Başlangıç index: " + indexNo + "<BR>");
        document.write("Arama Durumu: " + durum + "<BR>");
        sayac++;
    }
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



```
0.
Kelime: galatasaray
Aranan Karakter / Kelime: e
Başlangıç index: 0
Arama Durumu: -1
1.
Kelime: galatasaray
Aranan Karakter / Kelime: a
Başlangıç index: 4
Arama Durumu: 5
2.
Kelime: galatasaray
Aranan Karakter / Kelime: ata
Başlangıç index: 0
Arama Durumu: 3
```

Ekran alıntısında da görüldüğü gibi Şampiyon Galatasaray. Tabiki şaka. `indexOf()` fonksiyonunu daha iyi özetlemek için kullanıcıdan 3 giriş aldık. İlk durumda aranan karakter, kelimenin içerisinde olmadığı için -1 döndürdü. İkinci durumda ise aranan karakter, kelimenin içerisinde var arama indisini 4 olarak belirledik. Üçüncü durumda ise kelimenin içerisinde başka bir kelimeyi aradık ve arama indisini 0'dan başlattık.

15.5 `lastIndexOf()`

`indexOf()` fonksiyonu, aranan karakter veya kelimeyi bulduğu anda bulunan bu karakter veya kelimenin index adresini veriyordu. Eğer aranan kelimeye veya karakter, aranan kelimenin içerisinde birden fazla ile ilki dışında diğerlerine bakmıyordu. `lastIndexOf()` fonksiyonu ise aranan kelime veya karakterin en sonda olanının indisini verir. Yani emre ifadesinde e harfini arattığımızda `indexOf()` fonksiyonu 0, `lastIndexOf()` fonksiyonu ise 3 değerini döndürecektir. `lastIndexOf()` fonksiyonun standart kullanımı aşağıdaki gibidir.

```
kelime.lastIndexOf("aranacakKarakter");
```

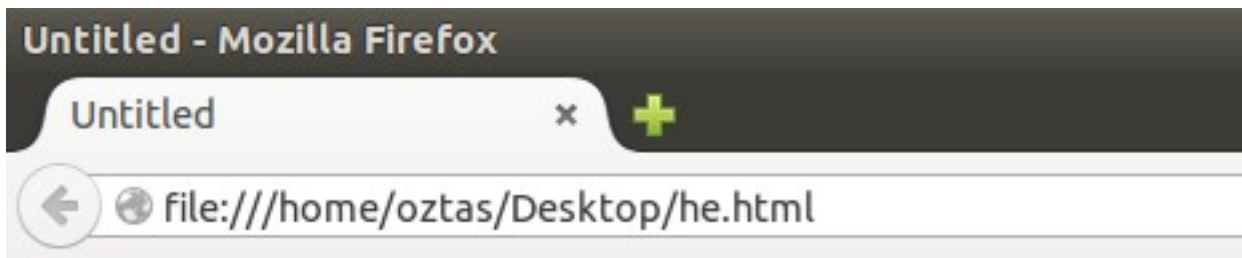
Bunun dışında `indexOf()` fonksiyonunda olduğu gibi belirtilen indisten başlayarakta arama işlemi gerçekleştirilebilir. Bu işlemi uygulamak için aşağıda gösterimi verilen yolu uygulamak gerekir.

```
kelime.lastIndexOf("aranacakKarakter", indexBaslangicNo);
```

`lastIndexOf()` fonksiyonu ile basit bir örnek yapalım. Bu örneğimizde `indexOf()` fonksiyonunu da kullalım ve iki fonksiyon arasındaki farkı anlamaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Untitled</title>
</head>
<body>
<script type="text/JavaScript">
    var kelime = "Galatasaray";
    var karakter = "a";
    var iof = kelime.indexOf(karakter);
    var liof = kelime.lastIndexOf(karakter);
    document.write("indexOf(): " + iof);
    document.write("<BR>");
    document.write("lastIndexOf(): " + liof);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



indexOf(): 1
lastIndexOf(): 9

Yukarıdaki ekran alıntısında da görüldüğü gibi; aranan karakter veya kelime, indexOf() fonksiyonu ile baştan, lastIndexOf() ile fonksiyonu ise sondan ilk indisi veririr. Tabiki aranan kelime veya karakter bulunamadığı durumda -1 döndürülmesi her iki fonksiyon içinde geçerlidir.

15.6 slice()

slice(), ilgili karakter dizisinin belli elemanlarını almak için kullanılan bir String fonksiyonudur. slice() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
kelime.slice(baslangic, bitis);
```

slice() fonksiyonu iki parametre alır. Bu parametrelerden ilki başlangıç değeridir, yani hangi karakter indisinden başlayarak alınacağıdır. Diğer parametre ise bitiş değeridir. Yani alınacak olan indisler hangi indis değerinde son bulacaktır.

String tipine ait diğer fonksiyonlarda olduğu gibi slice() fonksiyonu ile elde edilen değerler ya ekrana yazdırılmalı ya da herhangi bir değişkene bağlanmalıdır. Çünkü slice() fonksiyonu parametrelerle belirlenen değerleri herhangi bir değişkene bağlamaz isek veya doğrudan ekrana yazdırmaz isek; alınan bu değerler uçar gider, yani bir kalıcılığı olmaz. O yüzden slice() fonksiyonunu aşağıdaki şekildeki gibi kullanmamız daha doğru olacaktır.

```
degiskenAdi = kelime.slice(baslangic, bitis);
```

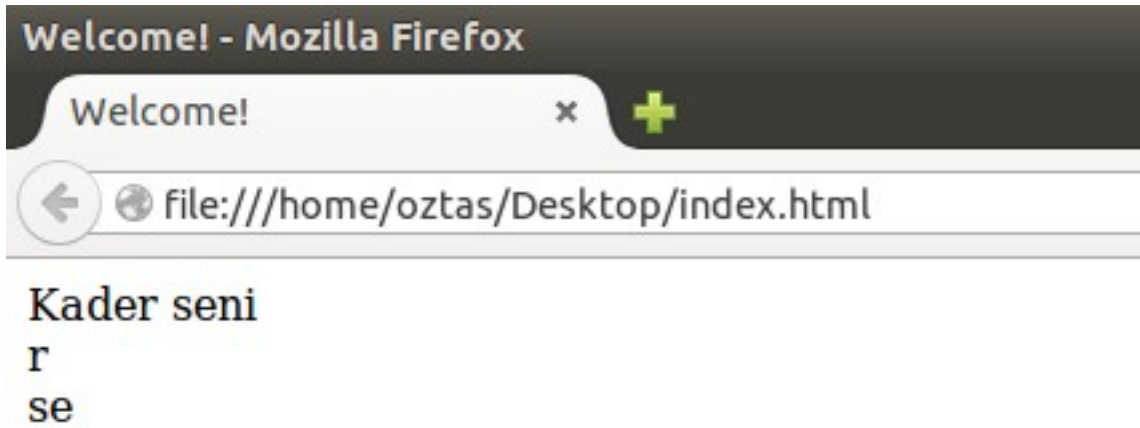
slice() fonksiyonu ile basit bir örnek yapalım ve bu fonksiyonu yakından tanımaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var kelime = "Kader seni güldürmüyorsa, espriyi anlayamadın demektir! - Anton Çehov";
    // ilk slice işlemi
    var s1 = kelime.slice(0, 10);
    // ikinci slice işlemi
    var s2 = kelime.slice(4, 5);
    //ucuncu slice işlemi
    var s3 = kelime.slice(6, 8);
    document.write(s1 + "<BR>");
```

```
document.write(s2 + "<BR>");  
document.write(s3 + "<BR>");  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimizde basit olarak slice() fonksiyonu ile bir karakter dizisinden rastgele karakter kümeleri kestik ve bunları değişkenlere atadık.

Örneğimize ait olan ekran çıktısını alalım.



slice() fonksiyonu ile herhangi bir karakter dizisinin, karakterlerini sondan başlayarak kesmek istersek, ikinci parametreyi negatif bir sayı vermemiz gerekir. Baştaki parametre soldan başlangıç indisiydi, hatırlarsanız. Bu yüzden ikinci parametrede sağdan başlangıç indisidir. Özetlemek gerekirse; “Turkey” kelimesinde “k” karakterini elde etmemiz gerekirse o zaman slice(3, -2) şeklinde fonksiyonumuzu düzenlememiz gerekir.

15.7 split()

split() fonksiyonu, ilgili karakter dizisi içerisinde, belirtilen separator (ayırıcı) ve belirtilen sayı kadar kelime veya karakter dizisi kesme işlemini gerçekleştirir. Tam özetleyemedim galiba ama örneklerimize geçtiğimizde ne demek istediğimi daha iyi anlayacağınızı sanıyorum. split() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
kelime.split(ayirici, max_ayrim_sayisi);
```

split() fonksiyonu, yukarıdaki standart kullanımında olduğu gibi iki parametre almaktadır. Bu parametrelerden birincisinde belirtilen karakter dizisinin neye göre ayrılacağıdır. Bu ayırma işlemi; , . / - + “ ” v.s gibi işaretlere göre olabilir. Bu tamamen sizin seçiminize kalmış. Buradaki tek şart tabiki ayırıcının karakter dizisi içerisinde geçmesidir. İkinci parametre ise ayırıcı ile parçalanmış bu karakter setlerinin kaç tane olmasının belirtilmesidir.

split() fonksiyonu ile basit bir örnek yapalım ve anlama katsayımızı arttırmaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

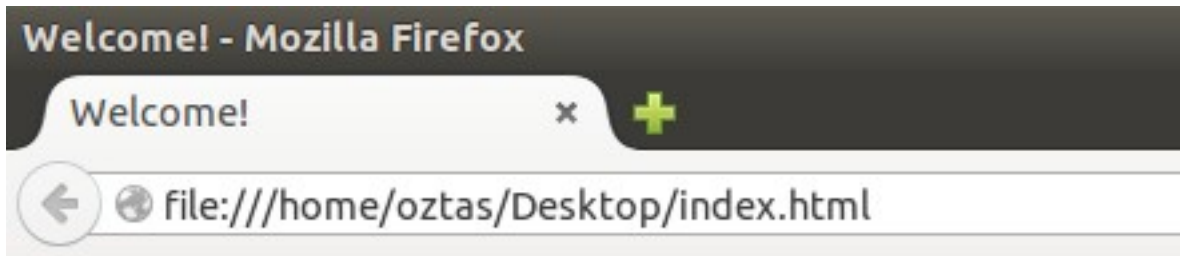
```
<!DOCTYPE html>  
<head>  
<meta charset="utf-8">  
<title>Welcome!</title>  
</head>
```

```

<body>
<script type="text/JavaScript">
    var s1 = "1.2.3.4.5.6.7.8.8.9.10";
    var s2 = "Muslera Sabri Chedjou Semih Telles Olcan Melo Selçuk Yasin Sneider Burak";
    var s3 = "Selam + Salam + Salem + Kalem + Kalam + Kelam";
    var sp1 = s1.split(".", 6);
    var sp2 = s2.split(" ", 1);
    var sp3 = s3.split("+", 2);
    document.write(sp1 + "<BR>");
    document.write(sp2 + "<BR>");
    document.write(sp3);
</script>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



1,2,3,4,5,6
Muslera
Selam , Salam

Yukarıdaki ekran alıntısında görüldüğü gibi verilen üç farklı karakter dizisini, bu karakter dizisi içerisinde geçen çeşitli ayırıcılara göre ayırdık. Bu ayırma işleminde kaç tane karakter kümesi oluşmasını istiyor isek bunun sayısını da belirttik.

15.8 substr()

`substr()`, ilgili karakter dizisinden belli sayıda karakter almak için kullanılan bir fonksiyondur. `substr()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
kelime.substr(baslangic, karakter_sayisi);
```

`substr()` fonksiyonu iki parametre almaktadır. Birinci parametre başlangıç indisi, ikinci parametre ise bu karakter dizisinden alınacak karakter sayısını belirtir. `substr()` fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

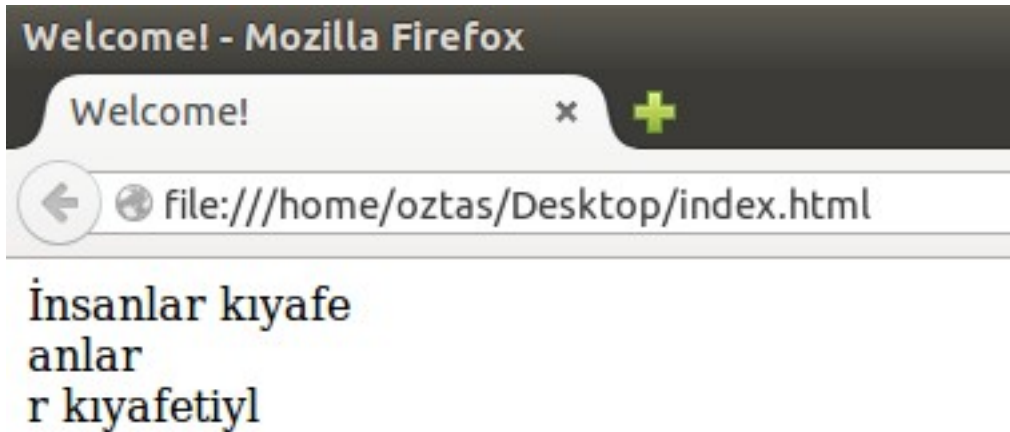
```

<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var kelime = "İnsanlar kıyafetiyle karşılanır, davranışıyla uğurlanır!  
Hz. Mevlana";

```

```
var s1 = kelime.substr(0, 15);  
var s2 = kelime.substr(3, 5);  
var s3 = kelime.substr(7, 12);  
document.write(s1 + "<BR>");  
document.write(s2 + "<BR>");  
document.write(s3);  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi; başlangıç indisini ve ne kadarlık bir karakter alacağımızı belirttiğimiz substr() fonksiyonu bu belirttiğimiz değerler arasındaki karakterleri bize verdi.

substr() fonksiyonu ile ilgili karakter dizinden, sondan başlayarakta belirli karakterleri alabiliriz. Bunun için ilk parametreyi negatif (-) sayı olarak belirtmemiz gerekir.

```
<!DOCTYPE html>  
<head>  
<meta charset="utf-8">  
<title>Welcome!</title>  
</head>  
<body>  
<script type="text/JavaScript">  
    var kelime = "İnsanlar kıyafetiyle karşılaşılır, davranışıyla uğurlanır!  
    Hz. Mevlana";  
    var s4 = kelime.substr(-11, 11);  
    document.write(s4);  
</script>  
</body>  
</html>
```

Yukarıdaki kodlarımızın ekran çıktısı da: Hz . Mevlana olacaktır.

15.9 toLowerCase()

toLowerCase() fonksiyonu, belirtilen karakter dizisini küçük harfe çevirir. Bu belirtilen karakter dizisi içerisinde büyük harf, küçük harf veya tamamen büyük ve tamamen küçük harfler olabilir. Bu durumlar herhangi birisinin olması farketmeksizin tüm karakter dizisi küçük harfe çevrilecektir. toLowerCase() fonksiyonunun standart kullanımı aşağıdaki gibidir.

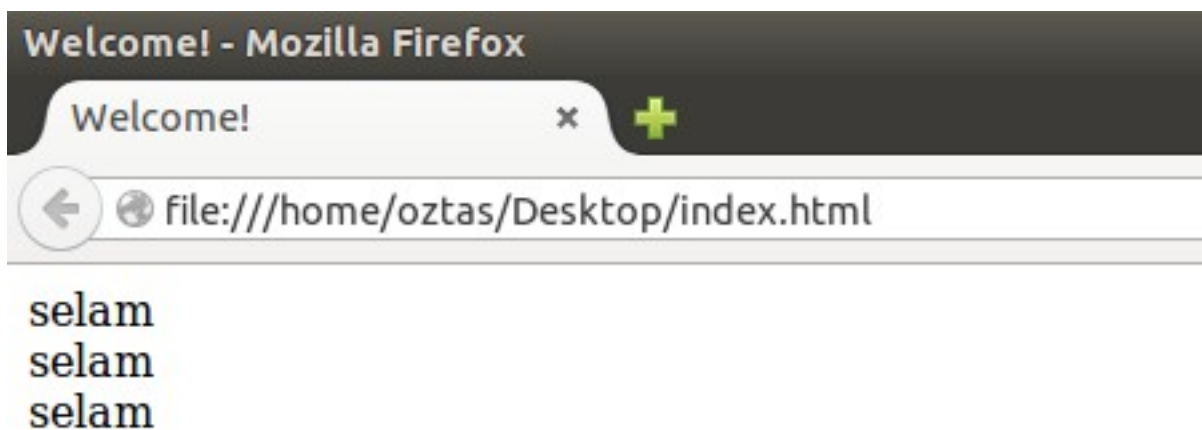
```
kelime.toLowerCase();
```

Her zaman dediğimiz gibi; bu fonksiyonların sonuçları ya doğrudan ekrana yazdırmalıyız ya da herhangi bir değişkene bağlamalıyız. Bunu unutmayalım.

toLowerCase() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = "Selam";
    var s2 = "seLAM";
    var s3 = "selam";
    document.write(s1.toLowerCase() + "<BR>");
    document.write(s2.toLowerCase() + "<BR>");
    document.write(s3.toLowerCase());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi toLowerCase() fonksiyonu belirtilen karakter dizisini tümüyle küçük harfe çevirmektedir.

15.10 toUpperCase()

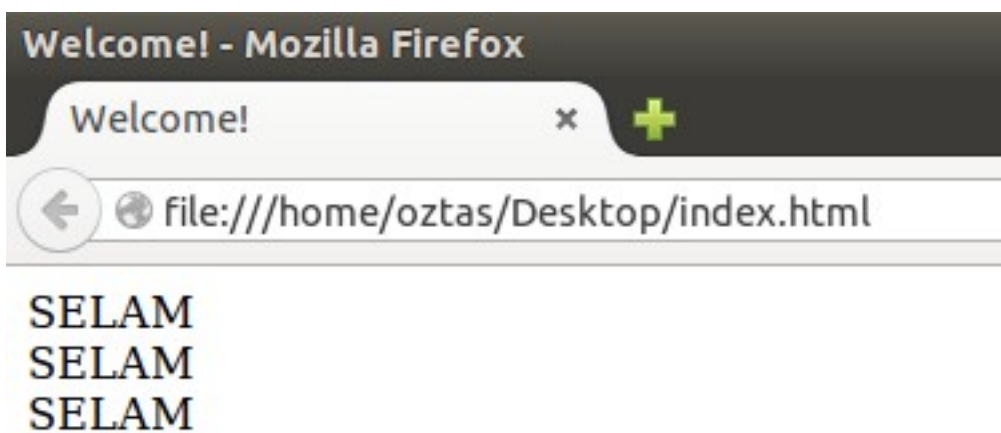
toUpperCase() fonksiyonu, belirtilen karakter dizisini büyük harfe çevirir. Bu belirtilen karakter dizisi içerisinde büyük harf, küçük harf veya tamamen büyük ve tamamen küçük harfler olabilir. Bu durumlardan herhangi birisinin olması farketmeksizin tüm karakter dizisi büyük harfe çevrilecektir. toUpperCase() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
kelime.toUpperCase();
```

ToUpperCase() fonksiyonu ile basit bir örnek yapalım. ToLowerCase() bölümündeki örneğimizi revize edelim. Örneğimiz aşağıdaki gibi olacaktır.


```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = "Selam";
    var s2 = "seLAM";
    var s3 = "selam";
    document.write(s1.toUpperCase() + "<BR>");
    document.write(s2.toUpperCase() + "<BR>");
    document.write(s3.toUpperCase());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi toUpperCase() fonksiyonu belirtilen karakter dizisini tümüyle büyük harfe çevirmektedir.

BÖLÜM 16:

Math

Programcılıkta, bazen bazı durumlarda matematiksel işlemler kullanmanız gerekir. Örneğin herhangi bir sayının üssünü hesaplama veya karekökünü alma gibi v.s. Bu gibi durumlarda kısa betikler yazarak bu sorunları çözebiliriz fakat fazladan kod yazmış oluruz. Bu da iyi bir programcının yapmaması gereken şeydir. En az kodla en fazla işi yapmak temel gayemiz olmalı. Çünkü kod satır sayısı ne kadar artarsa bilinki o bir o kadar amatörsünüzdür. Neyse konumuza dönelim.

JavaScript ortamında belli matematiksel işlemleri gerçekleştirmek için Math kütüphanesi kullanılır. Java, bilenler bilirler. Aynı mantık Java'da da vardır. Math sınıfı tamamen static bir yapıya sahiptir. Yani her yerden erişime açıktır. JavaScript'te de durum böyledir. Math kütüphanesinin sahip olduğu belli fonksiyonlar vardır. Bu fonksiyonları sırasıyla görelim.



Math kütüphanesinin sahip olduğu fonksiyonlara geçmeden önce bir uyarı da bulunalım. Hatırlarsanız bu tip fonksiyonlarda geri dönüş değerleri olduğu için, bu fonksiyonları herhangi bir değişkene bağlı olarak kullanmalı veya doğrudan ekrana yazdırmalıyız. Aksi halde dönüş değerleri havada kalır, ulaşamayız.

16.1 e

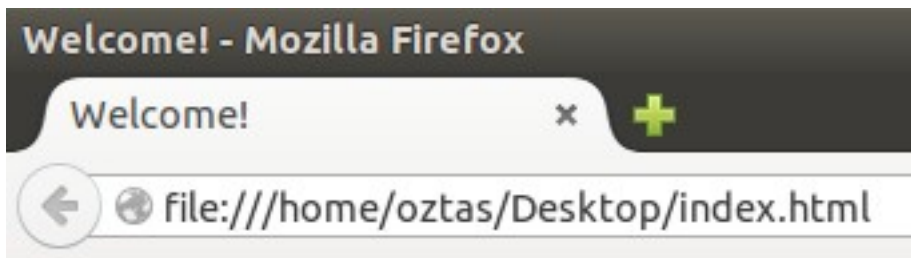
Logaritmadaki e harfini bilirsiniz. İşte burda ki e o'dur. Yani doğal logaritma (\ln). e standart olarak 2.7182 sayısına denktir. Math.E fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.E;
```

Math.E fonksiyonu ile basit bir örnek yapalım. Örneğin doğal logaritmanın tam olarak değerini öğrenelim.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var e = Math.E;
    document.write("e: " + e);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



e: 2.718281828459045

16.2 LN2

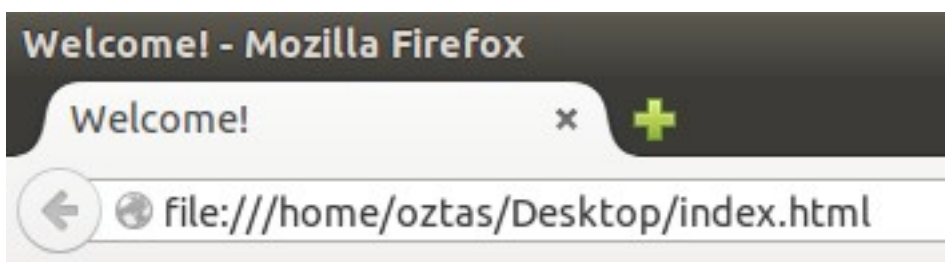
`Math.LN2` fonksiyonu, 2 sayısının e tabanındaki logaritmasını verir. Matematiksel olarak göstermek istersek; $\ln 2$. `Math.LN2` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.LN2;
```

`Math.LN2` fonksiyonu ile basit bir örnek yapalım. $\ln 2$ 'nin tam değerini hesaplayalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var ln2 = Math.LN2;
    document.write("ln2: " + ln2);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



ln2: 0.6931471805599453

16.3 LN10

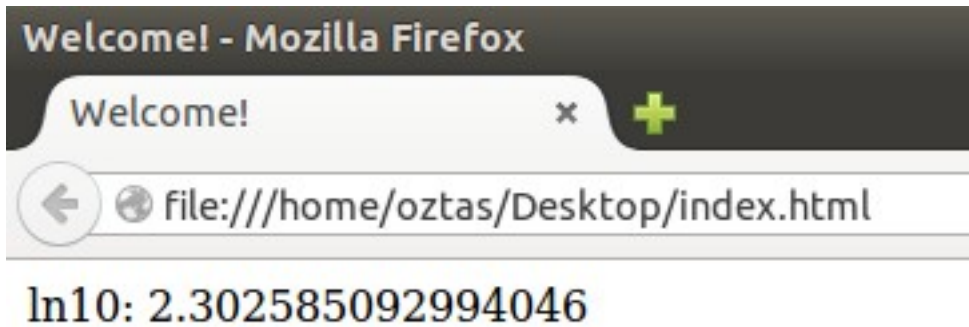
`Math.LN10` fonksiyonu, 10 sayısının e tabanındaki logaritmasını verir. Matematiksel olarak göstermek istersek; $\ln 10$. `Math.LN10` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.LN10;
```

`Math.LN10` fonksiyonu ile basit bir örnek yapalım. $\ln 10$ 'nun tam değerini hesaplayalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var ln10 = Math.LN10;
    document.write("ln10: " + ln10);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



16.4 LOG2E

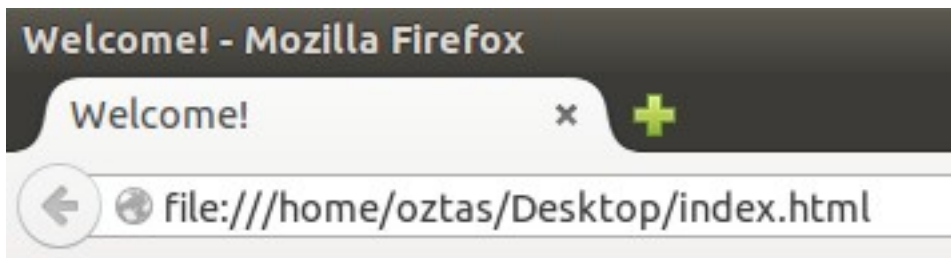
`Math.LOG2E` fonksiyonu, e 'nin logaritma 2 tabanındaki değerini verir. Matematiksel olarak göstermek istersek; $\log_2 e$. `Math.LOG2E` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.LOG2E;
```

`Math.LOG2E` fonksiyonu ile basit bir örnek yapalım. `Math.LOG2E`'nin tam değerini hesaplayalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var log2E = Math.LOG2E;
    document.write("log2E: " + log2E);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



log2E: 1.4426950408889634

16.5 LOG10E

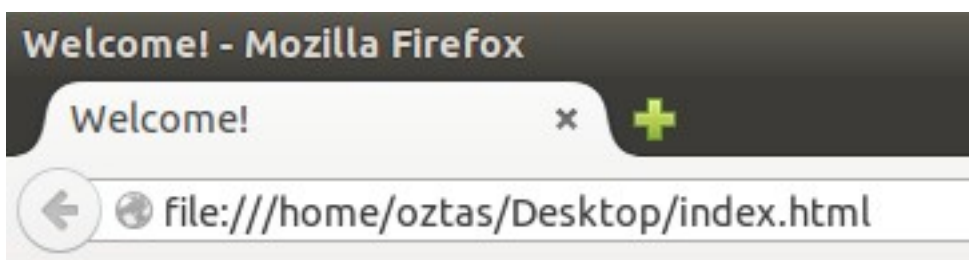
Math.LOG10E fonksiyonu, e'nin logaritma 10 tabanındaki değerini verir. Matematiksel olarak göstermek istersek; \log_e . Math.LOG10E fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.LOG10E;
```

Math.LOG10E fonksiyonu ile basit bir örnek yapalım. Math.LOG10E'nin tam değerini hesaplayalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var log10E = Math.LOG10E;
    document.write("log10E: " + log10E);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



log10E: 0.4342944819032518

16.6 PI

Math.PI fonksiyonu, pi sayısının değerini vermektedir. Standart olarak pi sayısı 3.14 sayısı olarak bilinir. Ya da tam değerini hesaplamak için $22 / 7$ işlemi yapılır. Math.PI fonksiyonu ile pi sayısının değerini kolayca elde edebilmekteyiz. Math.PI fonksiyonunun standart kullanımı aşağıdaki gibidir.

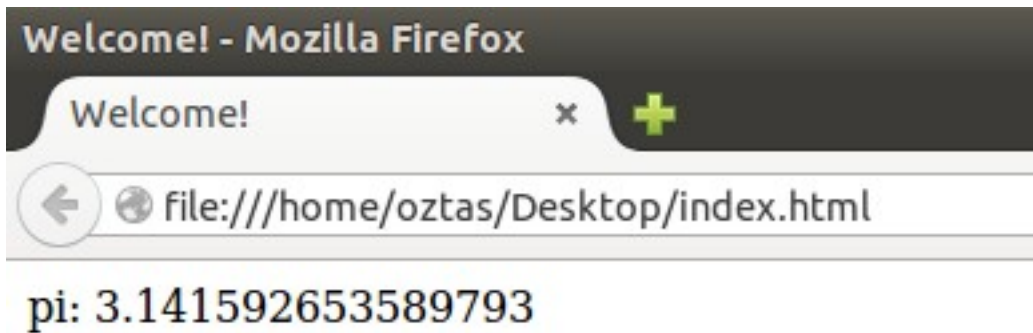
```
Math.PI;
```

Math.PI fonksiyonu ile pi sayısının tam değerini alalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var pi = Math.PI;
    document.write("pi: " + pi);
</script>
</body>
</html>
```

En başta da belirttiğimiz gibi eğer pi'nin değerini Math.PI ile hesaplamak aklınıza gelmeyecek bile olsa 22 / 7 işlemini uygulayarak pi'nin değerine ulaşabilirsiniz.

Yukarıdaki kodlarımızın ekran çıktısını alalım.



16.7 abs()

abs() fonksiyonu, herhangi belirtilen bir sayının mutlak değerini alır. abs kısaltması da zaten absolute yani mutlak ifadesinden gelmektedir. abs() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.abs(deger);
```

abs() fonksiyonu, parametre olarak sadece bir değer almaktadır. Mutlak değeri alınacak olan sayı hangisiyle abs(deger) şeklinde belirtilmelidir.

abs() fonksiyonu ile basit bir örnek yapalım. Örneğin çeşitli sayılar gönderelim ve sonuçlarına bakalım. Örneğimiz aşağıdaki gibi olacaktır.

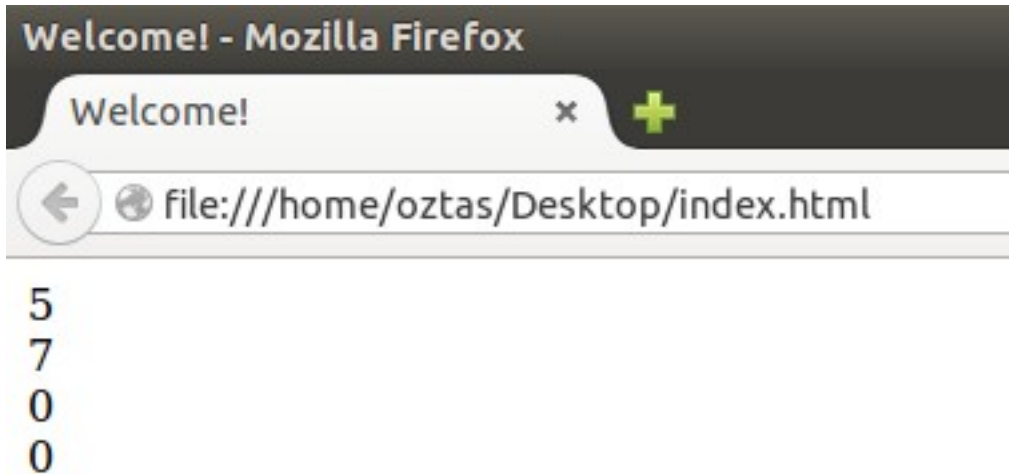
```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = Math.abs(5);
    var s2 = Math.abs(-7);
    var s3 = Math.abs(0);
```

```

var s4 = Math.abs(null);
document.write(s1 + "<BR>");
document.write(s2 + "<BR>");
document.write(s3 + "<BR>");
document.write(s4);
</script>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktısında gördüğümüz gibi negatif değerler abs() fonksiyonu ile pozitif hale gelmiştir. Burada dikkatinizi çekmek istediğim nokta null değerlerin 0 olduğudur. Herhangi bir null değer abs() fonksiyonu ile 0 değerini almaktadır.

İlla abs() fonksiyonu kullanarak negatif değerleri pozitif yapmamıza gerek yok. Bu bölümün en başında da belirttiğim gibi kendimiz betikler yazarakta sayılarımızı pozitif yapabiliriz. Aşağıdaki kod parçasını kullanarak.

```

<script type="text/JavaScript">
    var sayi = -35;
    if(sayi < 0) {
        sayi = sayi * (-1);
    }
</script>

```

16.8 acos()

acos() fonksiyonu, belirtilen sayının arccos'unu hesaplar ve dönen değeri verir. acos() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.acos(deger);
```

acos() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>

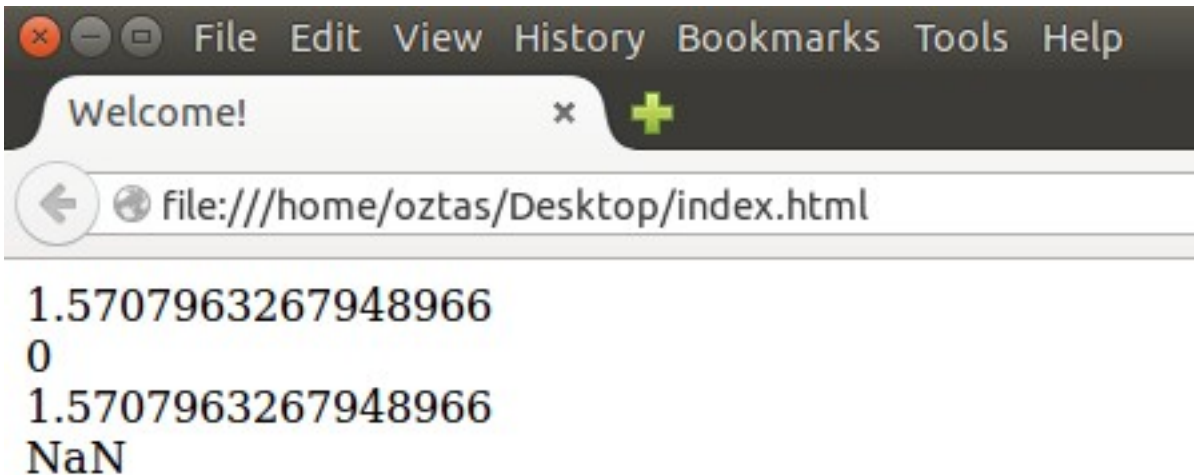
```

```

</head>
<body>
<script type="text/JavaScript">
    var s1 = 0;
    var s2 = 1;
    var s3 = null;
    var s4 = 60;
    document.write(Math.acos(s1) + "<BR>");
    document.write(Math.acos(s2) + "<BR>");
    document.write(Math.acos(s3) + "<BR>");
    document.write(Math.acos(s4) + "<BR>");
</script>
</body>
</html>

```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



16.9 asin()

`asin()` fonksiyonu, belirtilen sayının arcsin'ini hesaplar ve dönen değeri verir. `asin()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.asin(deger);
```

`acos()` fonksiyonunda yazmış olduğumuz örneğimizi `asin()` fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = 0;
    var s2 = 1;
    var s3 = null;
    var s4 = 60;
    document.write(Math.asin(s1) + "<BR>");
    document.write(Math.asin(s2) + "<BR>");
    document.write(Math.asin(s3) + "<BR>");

```



```
document.write(Math.asin(s4) + "<BR>");  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



16.10 atan()

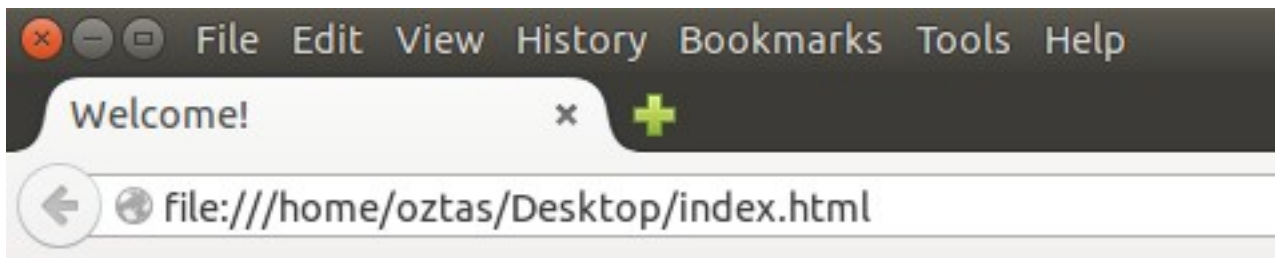
atan() fonksiyonu, belirtilen sayının arctan'ını hesaplar ve dönen değeri verir. atan() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.atan(deger);
```

asin() fonksiyonunda vermiş olduğumuz örneğimizi atan() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<head>  
<meta charset="utf-8">  
<title>Welcome!</title>  
</head>  
<body>  
<script type="text/JavaScript">  
    var s1 = 0;  
    var s2 = 1;  
    var s3 = null;  
    var s4 = 60;  
    document.write(Math.atan(s1) + "<BR>");  
    document.write(Math.atan(s2) + "<BR>");  
    document.write(Math.atan(s3) + "<BR>");  
    document.write(Math.atan(s4) + "<BR>");  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



```
0
0.7853981633974483
0
1.554131203080956
```

16.11 ceil()

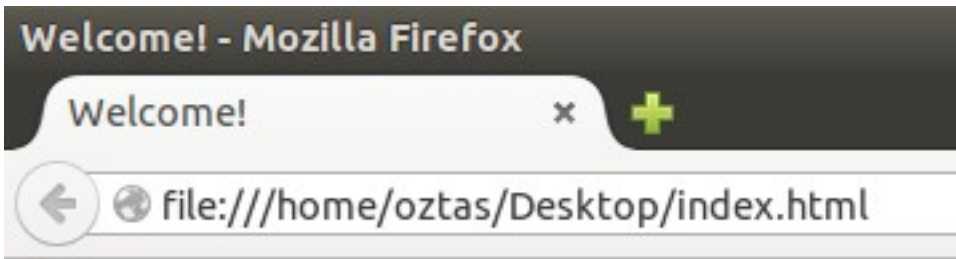
`ceil()` fonksiyonu, belirtilen kayan noktalı sayıyı, yukarıya, yani kendisine en yakın olan tam sayıya yuvarlar. Bu yuvarlama işleminde dikkat edilen tek şey sayının kayan noktalı olmasıdır. Sayı kayan noktalıysa başka birşeye bakılmaz. `ceil()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.ceil(kayanNoktalıDeger);
```

`ceil()` fonksiyonu basit bir örnek yapalım ve yakından tanımaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = 22 / 7;
    var s2 = 52.0;
    var s3 = 17.0001;
    var s4 = 29.999;
    var s5 = null;
    document.write(Math.ceil(s1) + "<BR>");
    document.write(Math.ceil(s2) + "<BR>");
    document.write(Math.ceil(s3) + "<BR>");
    document.write(Math.ceil(s4) + "<BR>");
    document.write(Math.ceil(s5) + "<BR>");
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



4
52
18
30
0

Yukarıdaki örneğimizde görüldüğü gibi; `ceil()` fonksiyonu kendisine gönderilen kayan noktalı sayıları, kendinden bir sonraki tamsayıya çevirmektedir. Null olan değerleri de 0 yapmaktadır. Yukarıdaki örneğimizde negatif (-) herhangi bir sayıyı eklemedik. Soralım o zaman peki `ceil()` fonksiyonu kendisine gönderilen negatif kayan noktalı bir sayıya nasıl davranır? `ceil` fonksiyonu kendisine gönderilen negatif bir tamsayının kesirli kısmını atar ve negatif sayının tam kısmını verir. Yukarıdaki örneğimizde 17.0001 sayıyı -17.0001 olsaydı, `ceil()` fonksiyonu bu sayıyı 17 olarak verecekti.

16.12 `cos()`

`cos()` fonksiyonu, belirlen herhangi bir sayının kosinüs değerini verir. `Cos()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

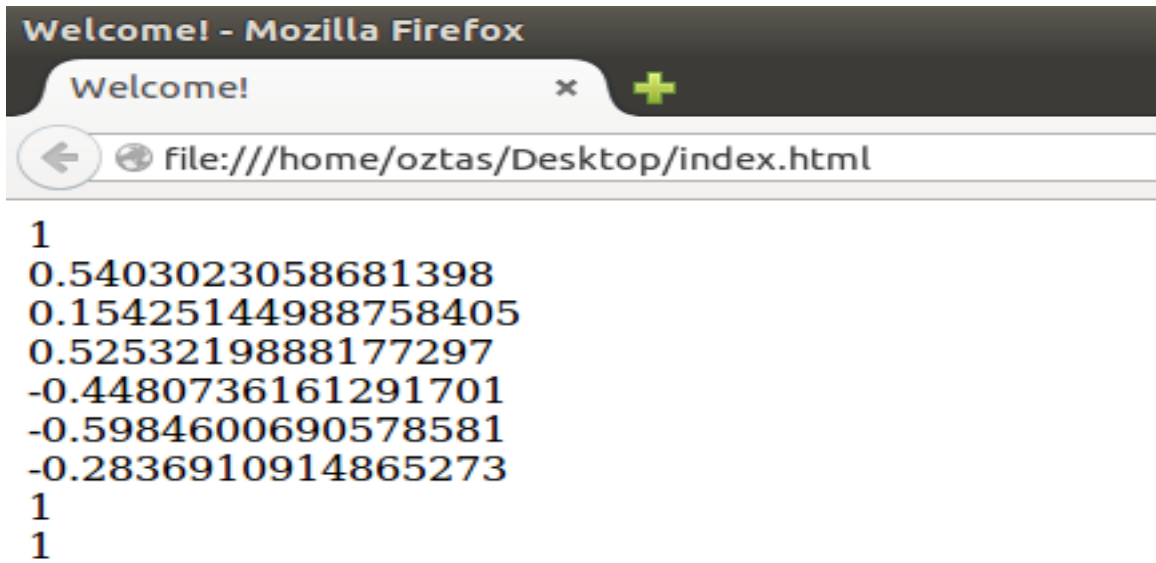
```
Math.cos(deger);
```

`cos()` fonksiyonuyla çok basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = 0;
    var s2 = 1;
    var s3 = 30;
    var s4 = 45;
    var s5 = 90;
    var s6 = 180;
    var s7 = 360;
    var s8 = null;
    document.write(Math.cos(s1) + "<BR>");
    document.write(Math.cos(s2) + "<BR>");
    document.write(Math.cos(s3) + "<BR>");
    document.write(Math.cos(s4) + "<BR>");
    document.write(Math.cos(s5) + "<BR>");
    document.write(Math.cos(s6) + "<BR>");
    document.write(Math.cos(s7) + "<BR>");
    document.write(Math.cos(s8) + "<BR>");
    document.write(Math.cos(s8));
```

```
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktısında görüldüğü gibi `cos()` fonksiyonu kendisine parametre olarak gönderdiğimiz değerleri hesapladı. Bu değerler tamamen çözümlenmiş değerler. En azında bilgisayar size kalkıpta $\cos 30 = \frac{1}{2}$ demez yani.

16.13 floor()

`floor()` fonksiyonu, kendisine parametre olarak gönderilen kayan noktalı sayıyı aşağı yani kendisine en yakın ve kendisinden küçük olan tamsayıya çevirir. Burada `floor()` fonksiyonun tek dikkat ettiği şey; kendisine parametre olarak gönderilen sayı kayan noktalı bir sayı mı değil mi. `floor()` fonksiyonu başka bir şeye bakmaz. `floor()` fonksiyonun standart kullanımı aşağıdaki gibidir.

```
Math.floor(kayanNoktalıSayı);
```

Her zaman olduğu gibi `floor()` fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = -3.02;
    var s2 = 17.06;
    var s3 = 4.0;
    var s4 = 5;
    var s5 = null;
    document.write(Math.floor(s1) + "<BR>");
    document.write(Math.floor(s2) + "<BR>");
    document.write(Math.floor(s3) + "<BR>");
    document.write(Math.floor(s4) + "<BR>");
    document.write(Math.floor(s5));
```

```
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi floor() fonksiyonu kendisine parametre olarak gönderilen kayan noktalı sayıları, kendisine en yakın ve kendisinden küçük olan sayıya çevirir. Burada dikkat ederseniz null olarak gönderilen parametre 0 oldu. Aynı şey ceil() fonksiyonunda da olmuştu. Yani her halukarda null parametreler 0 oluyor.

16.14 log()

log () fonksiyonu, kendisine parametre olarak gönderilen değerin e tabanında logaritmasını alır. Yani matematiksel olarak göstermek istersek; $\ln \text{deger}$. log() fonksiyonunun standart kullanımı aşağıdaki gibidir.

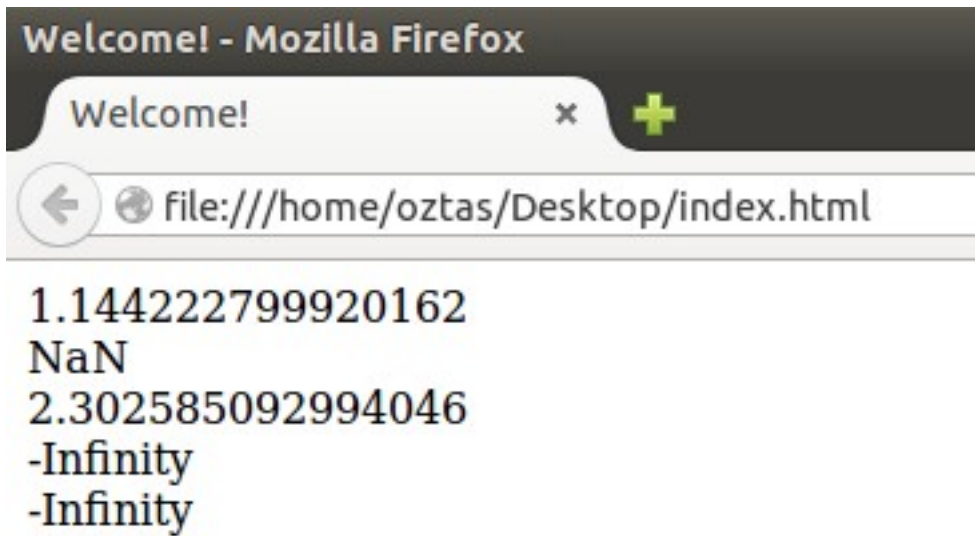
```
Math.log(deger);
```

log() fonksiyonu basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = 3.14;
    var s2 = -5;
    var s3 = 10;
    var s4 = null;
    var s5 = 0;
    document.write(Math.log(s1) + "<br>");
    document.write(Math.log(s2) + "<br>");
    document.write(Math.log(s3) + "<br>");
    document.write(Math.log(s4) + "<br>");
    document.write(Math.log(s5) + "<br>");
</script>
```

```
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Ekran çıktımızı inceleyelim. `log()` fonksiyonuna göre; kendisine parametre olarak gönderilen değer negatif ise daima NaN değerini döndürür. Null ve 0 değeri de infinity yani sonsuz olacaktır.

16.15 max()

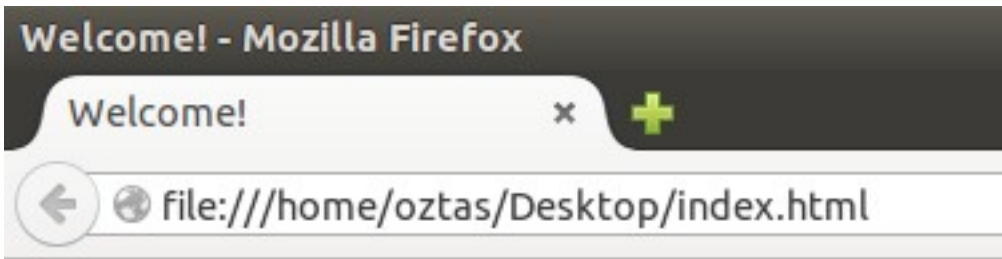
`max()` fonksiyonu, kendisine gönderilen parametreler arasından en büyük olanı seçer. Parametre uzunluğu bakımından herhangi bir sıkıntı yoktur. İstenilen sayıda parametre gönderilebilir. `Max()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.max(parametre1, parametre2, parametre3, ... ,parametreN);
```

`max()` fonksiyonu basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = Math.max(5, 3, -2, 15, -100);
    var s2 = Math.max(-4, -6, -12, -1.35, -2.75);
    var s3 = Math.max();
    var s4 = Math.max(4, 9);
    document.write(s1 + "<BR>");
    document.write(s2 + "<BR>");
    document.write(s3 + "<BR>");
    document.write(s4);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



15
-1.35
-Infinity
9

Yukarıdaki ekran çıktısını incelediğimizde `max()` fonksiyonuna parametre olarak gönderdiğimiz sayılar arasında en büyük olan yine `max()` fonksiyonu bize geri döndü. Burada dikkat etmemiz gereken nokta; `max()` fonksiyonuna herhangi bir parametre göndermediğimiz dönen değer infinity yani sonsuzdur.

16.16 min()

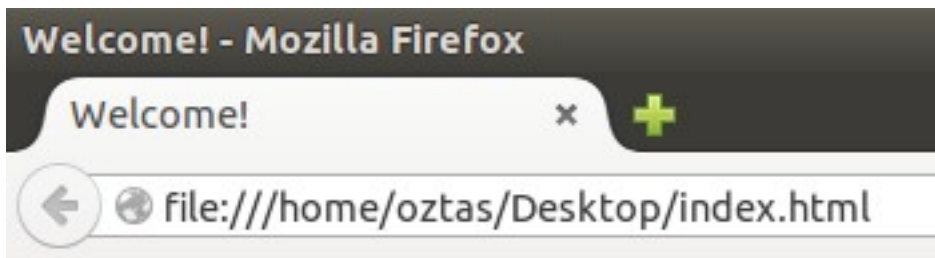
`min()` fonksiyonu, `max()` fonksiyonunun yaptığı işin tam tersini yapmaktadır. Yani kendisine gönderilen parametreler arasından en küçük olanı seçer. Parametre uzunluğu bakımından herhangi bir sınıtı yoktur. İstenilen sayıda parametre gönderilebilir. `min()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.min(parametre1, parametre2, parametre3, ... ,parametreN);
```

`max()` fonksiyonunda uyguladığımız örneğimizi `min()` fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = Math.min(5, 3, -2, 15, -100);
    var s2 = Math.min(-4, -6, -12, -1.35, -2.75);
    var s3 = Math.min();
    var s4 = Math.min(4, 9);
    document.write(s1 + "<BR>");
    document.write(s2 + "<BR>");
    document.write(s3 + "<BR>");
    document.write(4);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



-100
-12
Infinity
4

Yukarıdaki ekran çıktısını incelediğimizde; max() fonksiyonunda olduğu gibi min() fonksiyonuna parametre gönderilmediği zaman dönen değer infinity yani sonsuz olacaktır.

16.17 pow()

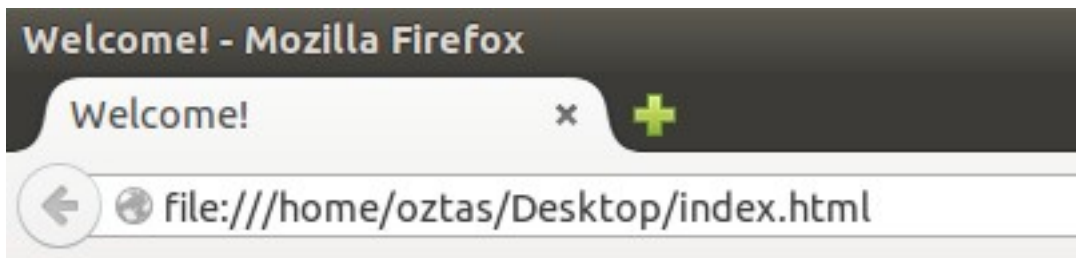
pow() fonksiyonu, üs alma fonksiyonudur. İki parametre alır. Bunlardan ilki taban değeri diğeri ise üs değeridir. pow() fonksiyonunun standart gösterimi aşağıdaki gibidir.

```
Math.pow(taban, us);
```

pow() fonksiyonu ile çok basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = Math.pow(3, 5);
    var s2 = Math.pow(-2, 4);
    var s3 = Math.pow(null, null);
    var s4 = Math.pow(10, 0);
    var s5 = Math.pow(0, null);
    document.write(s1 + "<BR>");
    document.write(s2 + "<BR>");
    document.write(s3 + "<BR>");
    document.write(s4 + "<BR>");
    document.write(s5 + "<BR>");
</script>
</body>
</html>
```

Yukarıdaki fonksiyonumuza ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktımızda, null değerlerin aldığı değerlere dikkat edelim.

16.18 random()

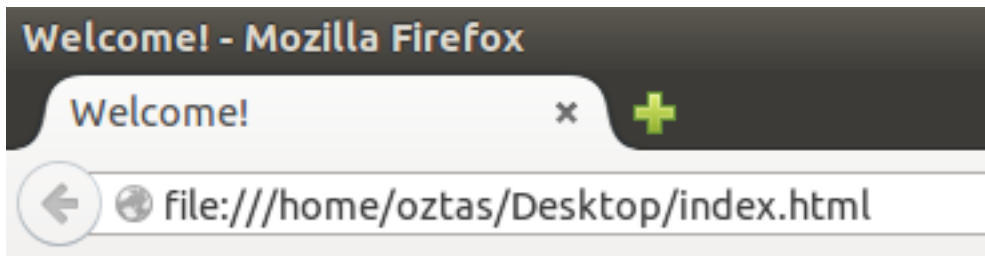
`random()` fonksiyonu, 0 ile 1 arasında bir sayı üretmek için kullanılır. `random()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.random();
```

`random()` fonksiyonu ile basit bir örnek yapalım. 10 adet sayı üretelim ve sonuçlarını ekrana yazdıralım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    for(var i = 1; i <= 10; i++){
        var rnd = Math.random();
        document.write(i + ". " + rnd + "<BR>");
    }
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



1. 0.7758451260500612
2. 0.07646360878217884
3. 0.08883637273308631
4. 0.9448815100648779
5. 0.6073654267573225
6. 0.9815834571506967
7. 0.337655880292663
8. 0.43125024043240323
9. 0.9901960202740571
10. 0.360356910594136

16.19 round()

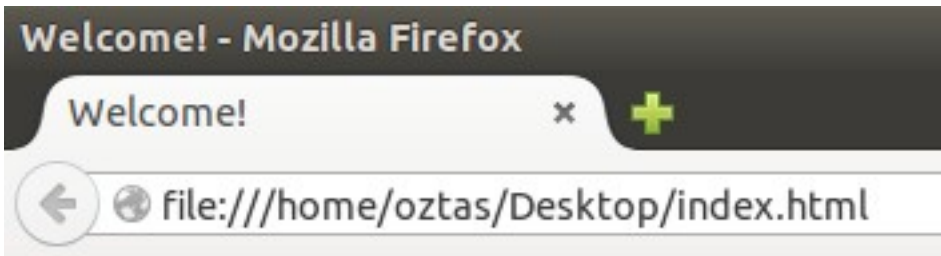
`round()` fonksiyonu, kendisine parametre olarak gönderilen kayan noktalı bir değeri; `ceil()` ve `floor()` fonksiyonlarından farklı bir yolla yuvarlama işlemi yapar. Eğer sayının virgülden sonraki değeri 0.5'ten küçükse: aşağı, büyükse: yukarı yuvarlama işlemi yapar. Örneğin herhangi bir öğrencinin hesaplanan notları son işlem olarak `round` fonksiyonuna tabi tutulur, Türkiye'de. `Round()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.round(kayanNoktalıSayı);
```

`round()` fonksiyonu ile basit örnek yapalım. Kullanıcıdan aldığımız vize ve final notlarını hesaplayıp son işlem olarak `round()` fonksiyonuna tabi turalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    for(var i = 1; i < 3; i++){
        var vize = parseInt(prompt("Vize: "));
        var fnal = parseInt(prompt("Final"));
        var sonuc = (vize * 0.6) + (fnal * 0.4);
        var yeniSonuc = Math.round(sonuc);
        document.write(i + "<BR>");
        document.write("Sonuç: " + sonuc + "<BR>");
        document.write("Yeni Sonuç: " + yeniSonuc + "<BR>");
    }
</script>
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



```
1
Sonuç: 64.2
Yeni Sonuç: 64
2
Sonuç: 53.6
Yeni Sonuç: 54
```

16.20 sin()

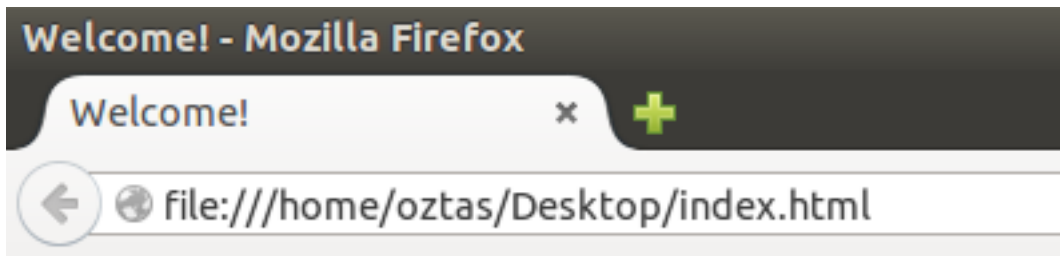
`sin()` fonksiyonu, temel olarak kendisine gönderilen değerin sinüs değerini hesaplar. Yani matematiksel olarak göstermek istersek: `s indeger`. `Sin()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.sin(deger);
```

`cos()` fonksiyonunda verdiğimiz örneğimizi `sin()` fonksiyonu için revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = 0;
    var s2 = 1;
    var s3 = 30;
    var s4 = 45;
    var s5 = 90;
    var s6 = 180;
    var s7 = 360;
    var s8 = null;
    document.write(Math.sin(s1) + "<BR>");
    document.write(Math.sin(s2) + "<BR>");
    document.write(Math.sin(s3) + "<BR>");
    document.write(Math.sin(s4) + "<BR>");
    document.write(Math.sin(s5) + "<BR>");
    document.write(Math.sin(s6) + "<BR>");
    document.write(Math.sin(s7) + "<BR>");
    document.write(Math.sin(s8) + "<BR>");
    document.write(Math.sin(s8));
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



0
0.8414709848078965
-0.9880316240928618
0.8509035245341184
0.8939966636005579
-0.8011526357338304
0.9589157234143065
0
0

16.21 sqrt()

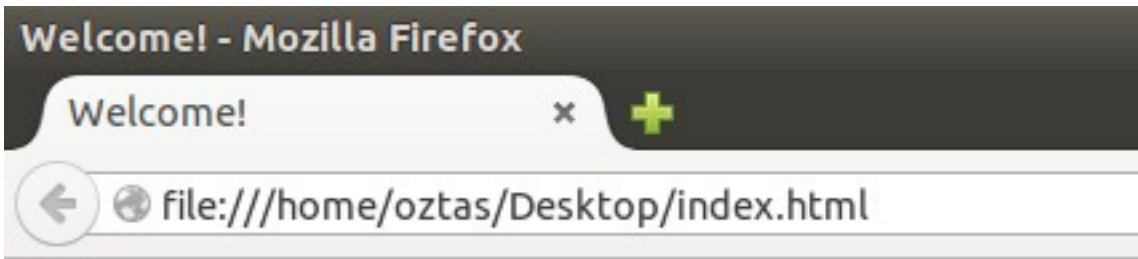
`sqrt()` fonksiyonu, temel olarak; aldığı parametrenin karekökünü almaktadır. `sqrt()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
Math.sqrt(deger);
```

`sqrt()` fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = Math.sqrt(4);
    var s2 = Math.sqrt(25);
    var s3 = Math.sqrt(16);
    var s4 = Math.sqrt(7);
    document.write(s1 + "<BR>");
    document.write(s2 + "<BR>");
    document.write(s3 + "<BR>");
    document.write(s4);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi çeşitli sayıların karekökünü alma işlemini gerçekleştirdik.

`sqrt()` fonksiyonu sadece bir tane parametre alır ve aldığı bu parametrenin karekökünü hesaplar. Peki örneğin üçüncü derecen beş sayısının kökünü almak istersek? Ya da başka herhangi bir formdaki köklü bir sayının kökünü almak istersek? O zaman `sqrt()` fonksiyonu çaresiz kalır. Böyle durumlarda daha farklı bir yaklaşım vardır. Temel matematiğiniz var ise; üçüncü dereceden kök beş ifadesinin, $5^{1/3}$ olduğunu bilirsiniz. Geriye sadece bu sayının `pow()` fonksiyonu ile üs hesaplaması kalır. Aşağıdaki örneğimizi inceleyelim.

```
<script type="text/JavaScript">
    // 3'ncü dereceden kök 5 hesaplaması
    var taban = 5;
    var us = 1/3;
    sonuc = Math.pow(taban, us);
    // 5'nci dereceden kök 25 hesaplaması
    var taban = 25;
    var us = 1/5;
    sonuc = Math.pow(taban, us);
    // 5'nci dereceden kök 9 hesaplaması
    var taban = 9;
    var us = 1/5;
    sonuc = Math.pow(taban, us);
    /* aynı yapıyı kullanarak çeşitli
       formdaki köklü sayılar
       kolayca hesaplanabilir.
    */
</script>
```

Yukarıdaki örneğimizde bulunan yapı ile istediğiniz köklü sayının kök alma işlemi kolayca yapılabilir. Tekrar bahsedelim. `sqrt()` fonksiyonu sadece belirtilen sayının karekökünü hesaplar. Bu bilgi aklınızın bir köşesinde bulunsun.

16.22 tan()

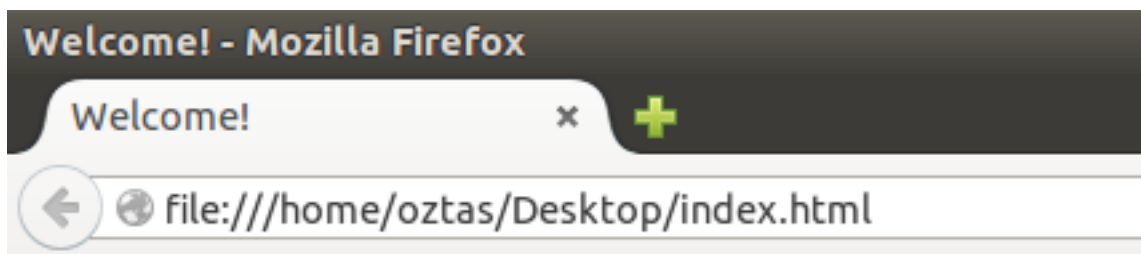
`tan()` fonksiyonu, kendisine parametre olarak gönderilen sayının tanjantını hesaplar. Trigonometriden de hatırlarsınız; karşı kenar / komşu kenar hesabı yapmaktadır. `tan()` fonksiyonun standart kullanımı aşağıdaki gibidir.

```
Math.tan(deger);
```

sin() fonksiyonundaki örneğimizi tan() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Welcome!</title>
</head>
<body>
<script type="text/JavaScript">
    var s1 = 0
    var s2 = 1;
    var s3 = 30;
    var s4 = 45;
    var s5 = 90;
    var s6 = 180;
    var s7 = 360;
    var s8 = null;
    document.write(Math.tan(s1) + "<BR>");
    document.write(Math.tan(s2) + "<BR>");
    document.write(Math.tan(s3) + "<BR>");
    document.write(Math.tan(s4) + "<BR>");
    document.write(Math.tan(s5) + "<BR>");
    document.write(Math.tan(s6) + "<BR>");
    document.write(Math.tan(s7) + "<BR>");
    document.write(Math.tan(s8) + "<BR>");
    document.write(Math.tan(s8));
</script>
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısı da aşağıdaki gibi olacaktır.



```
0
1.5574077246549023
-6.405331196646276
1.6197751905438615
-1.995200412208242
1.3386902103511544
-3.380140413960958
0
0
```

BÖLÜM 17:

Date

Date kütüphanesi ile tarih ve saat işlemlerimizi gerçekleştirebilmekteyiz. Bazı durumlarda tarih ve saat işlemleri yapmamız gerekir. Örneğin sayfamız yüklendiğinde ekranda tarih ve saati gösterebiliriz veya web için geliştirilmiş herhangi bir programlama diline gerek kalmadan kullanıcının herhangi bir formu doldurma ve gönderme zamanını göstermek isteyebiliriz. Bu gibi durumlarda JavaScript'in sahip olduğu Date kütüphanesini kullanmaya ihtiyaç duyarız.

Date bir kütüphanedir dedik. Dolayısıyla Date kütüphanesinin kendine ait fonksiyonları vardır. Biz de Date kütüphanesinin fonksiyonlarını kullanmak için; Date kütüphanesinden bir nesne oluşturmamız ve işlemlerimizi bu nesne üzerinden gerçekleştirmeliyiz.

Aşağıdaki form ile Date kütüphanesinden bir nesne oluşturabiliriz.

```
var tarihSaat = new Date();
```

Date kütüphanesinden nesne oluşturduktan sonra, bu nesne ile tam tarih ve saat değerlerini alalım. Bu tam tarih değeri içinde saniye, dakika, saat, gün, ay ve yıl değerleri vardır. Daha sonraki göreceğimiz fonksiyonlarda bu değerleri ayrı ayrı olarak alabiliriz. Yazacağımız kod aşağıdaki gibi olacaktır.

```
<script type="text/JavaScript">
    // nesne olusturalim.
    var tarihSaat = new Date();
    // tarihSaat nesnemizde artık tam tarih degeri bulunuyor.
    // ekrana yazdiralim.
    document.write("Tarih / Saat: " + tarihSaat);
</script>
```

Date kütüphanesini 2 ana başlık altında detaylı olarak incelemeye çalışacağız. Bu başlıklar: Local ve Goba. Daha sonra Extras başlığı altında tarih ve saat formatıyla ilgili değineceğimiz belli bir takım fonksiyonlar olacak. Birinci ana başlığımız ile başlayalım.

17.1 Local: Tarih ve Saat

Bu ana başlık altında Date ile tarih ve saat işlemlerimizi gerçekleştireceğiz. Ayrıca bu başlık altında göreceğimiz Date kütüphanesinin fonksiyonlarının ürettiği tarih ve saat değerleri tamamen kullanıcının bilgisayarından alınan değerlerdir. Yani genel olarak saat 17.00 iken kullanıcının bilgisayarındaki saat 17.02 ise saat için dönecek olan değer: 17.02'dir. Şimdi Date kütüphanesinin, kullanıcının bilgisayarı kaynaklı tarih ve saat işlemlerini gerçekleştirelim.

17.1.1 getDate()

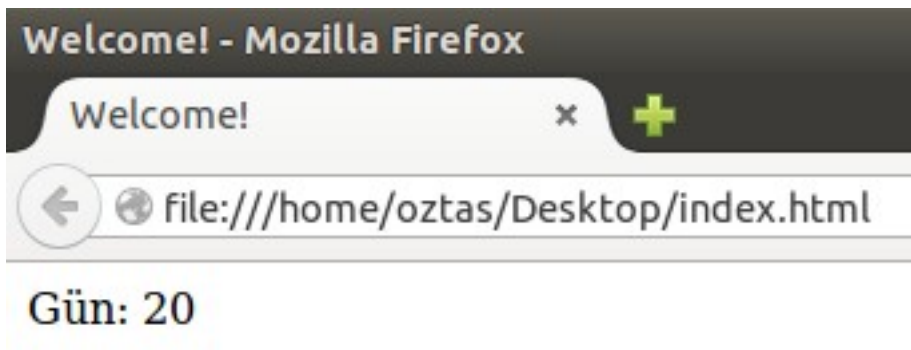
getDate() fonksiyonu, içinde bulunulan günün değerini verir. Yani bugün ayın kaçı? Sorusuna cevap veren fonksiyondur diyebiliriz. getDate() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getDate();
```

getDate() fonksiyonu ile basit bir örnek yapalım. Şuan içinde bulunduğumuz gün değerini alalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write("Gün: " + tarihSaat.getDate());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi bugün ayın 20'si. Yani herhangi bir sıkıntı yok.

Şayet, Time & Date Settings segmesinden bilgisayarımın tarih olarak gün değerini değiştirirsem, getDate() fonksiyonu ile dönen gün değeri de değişecektir. Buna dikkat edelim.

17.1.2 getDay()

getDay() fonksiyonu, şuan içinde bulunduğumuz günün haftalık bazda değerini verir. Yani daha açıklayıcı olmasını istersek; bugün haftanın hangi günü ise bu değeri bize sayısal olarak verir. getDay() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getDay();
```

getDay() fonksiyonu ile alakalı bir özellik var. getDay() fonksiyonunda, günler;

0 – Pazar 1 – Pazartesi 2 – Salı 3 – Çarşamba 4 – Perşembe 5 – Cuma 6 – Cumartesi

olarak ilerlemektedir. Yani haftanın günlerini bir dizi gibi düşünersek; 0 nolu indis pazar gününe denk düşmektedir. Buna dikkat edelim.

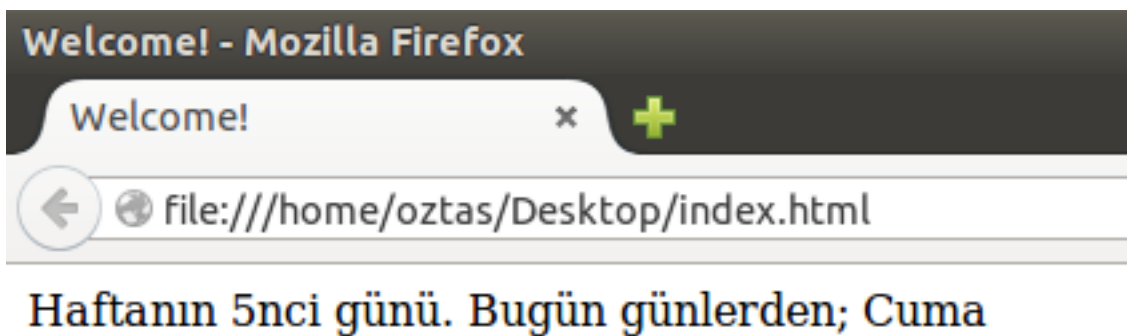
getDay() fonksiyonu ile basit bir örnek yapalım. Bu örneğimizde; içinde bulunduğumuz günün haftanın hangi günü olduğunu bulalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
```



```
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var gun = tarihSaat.getDay();
    document.write("Haftanın " + gun + "nci günü. ");
    document.write("Bugün günlerden; ");
    switch(gun){
    case 0: document.write("Pazar");
    break;
    case 1: document.write("Pazartesi");
    break;
    case 2: document.write("Salı");
    break;
    case 3: document.write("Çarşamba");
    break;
    case 4: document.write("Perşembe");
    break;
    case 5: document.write("Cuma");
    break;
    case 6: document.write("Cumartesi");
    break;
    }
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi `getDay()` fonksiyonu ile dönen değer her zaman sayısaldır. Biz örneğimizde `switch..case` yapısı ile bu dönen değeri ekrana günün adı şeklinde yazdırdık. Aynı zamanda bu değeri sayısal olarak doğrudan da ekrana yazdırabiliriz.

17.1.3 `getMonth()`

`getMonth()` fonksiyonu, içinde bulunduğumuz ay'ı sayısal olarak döndürür. Yani içinde bulunduğumuz ay hangisi sorusuna yanıt veren fonksiyondur diyebiliriz. `getMonth()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getMonth();
```

`getMonth()` fonksiyonunda ay sıralamaları günlük hayattaki gibi gitmez. Onun yerine aşağıdaki form uygulanır.

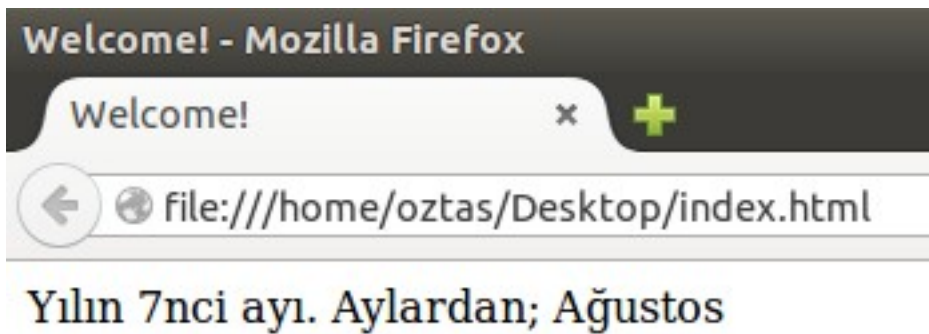
| | |
|-------------|-------------|
| 0 – Ocak | 6 – Temmuz |
| 1 – Şubat | 7 – Ağustos |
| 2 – Mart | 8 – Eylül |
| 3 – Nisan | 9 – Ekim |
| 4 – Mayıs | 10 – Kasım |
| 5 – Haziran | 11 – Aralık |

şeklinde gitmektedir. Ayları bir dizi şeklinde düşünersek; bu dizinin 0 nolu indisi ocak ayına ait olacaktır.

getMonth() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var ay = tarihSaat.getMonth();
    document.write("Yılın " + ay + "nci ayı. ");
    document.write("Aylardan; ");
    switch(ay){
        case 0: document.write("Ocak"); break;
        case 1: document.write("Şubat"); break;
        case 2: document.write("Mart"); break;
        case 3: document.write("Nisan"); break;
        case 4: document.write("Mayıs"); break;
        case 5: document.write("Haziran"); break;
        case 6: document.write("Temmuz"); break;
        case 7: document.write("Ağustos"); break;
        case 8: document.write("Eylül"); break;
        case 9: document.write("Ekim"); break;
        case 10: document.write("Kasım"); break;
        case 11: document.write("Aralık"); break;
    }
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktısını incelediğimizde aldığımız sonuçlar doğru lakin günlük hayatta ay değeri bakımından örtüşmüyor. getMonth() fonksiyonu ile dönen ay değerine +1 ekleyerek bu sorunu çözebiliriz.

17.1.4 getFullYear()

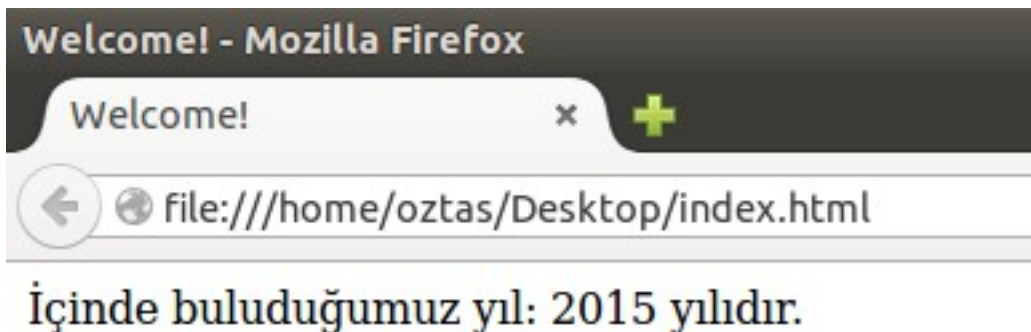
getFullYear() fonksiyonu, yıl değerini döndürür. Şuan içinde bulunduğumuz yıl 2015. getFullYear() fonksiyonu da bu değeri bize verecektir. getFullYear() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getFullYear();
```

getFullYear() fonksiyonu ile alakalı çok küçük bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var yil = tarihSaat.getFullYear();
    document.write("İçinde bulunduğumuz yıl: " + yil + " yılıdır.");
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Ekran alıntısında da görüldüğü üzere içinde bulunduğumuz yıl doğru bir şekilde getFullYear() fonksiyonu ile verildi. Çünkü bilgisayarımın güncel tarihi doğru. Yanlış olsaydı bu değer de yanlış çıkacaktı.

17.1.5 getMilliseconds()

getMilliseconds() fonksiyonu, şuanki aktif zamanın milisaniyesini verir. Bu milisaniye değeri; 0 – 999 arasında bir sayı olacaktır. getMilliseconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

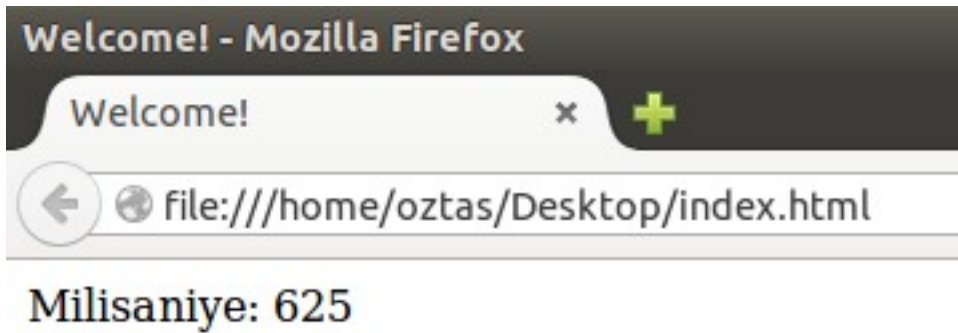
```
tarihSaat.getMilliseconds();
```

getMilliseconds() fonksiyonu ile aktif zamanın milisaniyesini alalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
```

```
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var miliSaniye = tarihSaat.getMilliseconds();
    document.write("Milisaniye: " + miliSaniye);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Tarayıcımızı her Refresh (Yenile: F5) ettiğimizde bu değer değişecektir.

17.1.6 getSeconds()

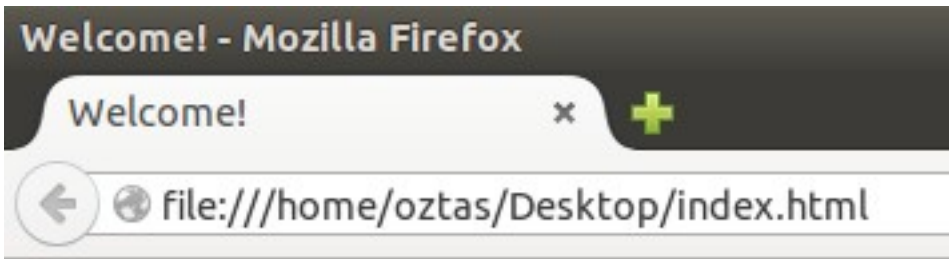
getSeconds() fonksiyonu, şuanki aktif zamanın saniyesini verir. Bu saniye değeri; 0 – 59 arasında bir sayı olacaktır. getSeconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getSeconds();
```

getSeconds() fonksiyonu ile basit bir örnek yapalım. Bu örneğimizde aktif zamanın saniyesini bulalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var saniye = tarihSaat.getSeconds();
    document.write("Saniye: " + saniye);
</script>
</body>
</html>
```

Yukarıdaki örneğimizin ekran çıktısını alalım.



Saniye: 59

Şansa bakın ki tam 59'ncu saniyede ekran alıntısı yapmayı başarmışım. getMilliseconds() fonksiyonunda da hatırlattığımız gibi; Refresh (Yenile: F5) ettiğimizde bu saniye değeri değişecektir.

17.1.7 getMinutes()

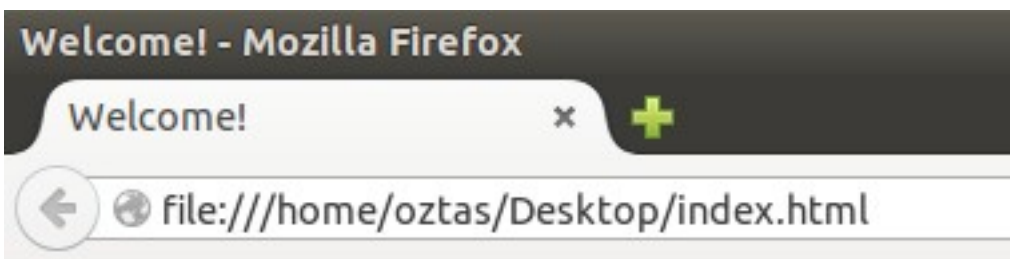
getMinutes() fonksiyonu, şuanki aktif zamanın dakikasını verir. Bu dakika değeri; 0 – 59 arasında bir sayı olacaktır. getMinutes() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getMinutes();
```

Aşağıdaki örneğimiz ile aktif zamanın dakikasını öğrenelim.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var dakika = tarihSaat.getMinutes();
    document.write("Dakika: " + dakika);
</script>
</body>
</html>
```

Örneğimizin ekran çıktısını alalım.



Dakika: 57

17.1.8 getHours()

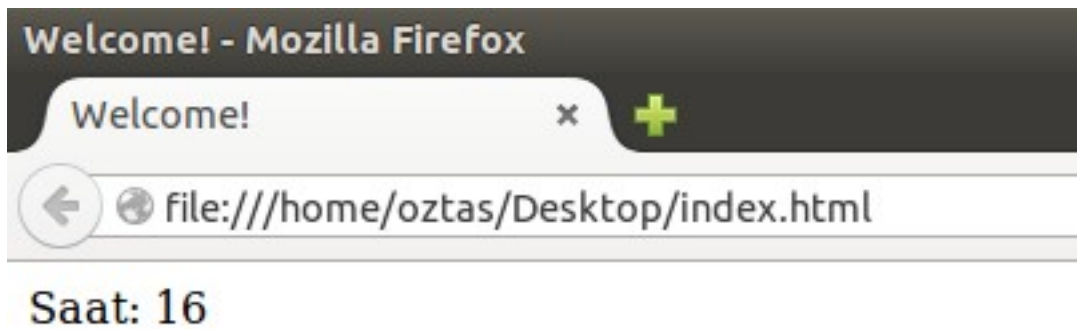
getHours() fonksiyonu, şuanki aktif zamanın saatini verir. Bu saat değeri; 0 – 23 arasında bir sayı olacaktır. getHours() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getHours();
```

getHours() fonksiyonu ile şuan içinde bulunduğumuz aktif zamanın saatini alalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var saat = tarihSaat.getHours();
    document.write("Saat: " + saat);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



17.1.9 getTime()

Programlama dillerinde genellikle başlangıç zamanı genellikle; 1 Ocak 1970 00:00:00 olarak alınır. Yani 1 Ocak 1970'e kadar destek verirler. getTime() fonksiyonunun görevi de 1 Ocak 1970'ten günümüze kadar geçen zamanı milisaniye cinsinden olarak döndürür. getTime() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getTime();
```

getTime() fonksiyonu ile sadece; 1 Ocak 1970'ten günümüze kadar geçen zaman milisaniye cinsinden dönmez. Date kütüphanesinden nesne oluştururken, Date kütüphanesine göndereceğimiz tarih parametreleri ile bu belirlediğimiz zamana kadar olan zamanı da milisaniye cinsinden alabiliriz. Aşağıdaki örnek kullanımda olduğu gibi.

```
// tarih parametresini duzgun yazmaya dikkat edelim.
var tarihSaat = new Date("October 1, 2000 21:44:57");
tarihSaat.getTime();
```

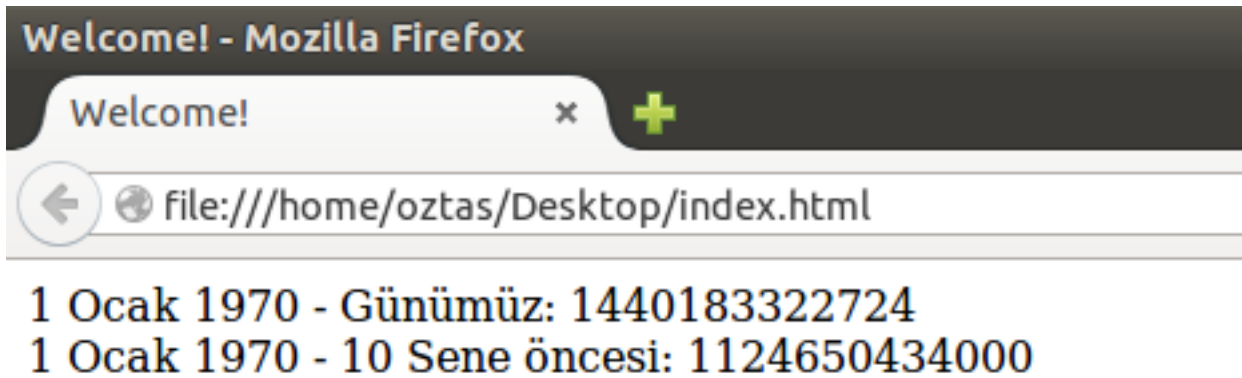
Yukarıdaki örnek kullanımı incelediğimizde; zamanın farklı bir biçimde gönderildiğini görüyoruz. Yani daha açıklayıcı göstermek istersek;
Ay Gun, Yıl Saat:Dakika:Saniye

Tabiki ay ismini de ingilizce olarak yazmak zorundayız. Buna da dikkat edelim.

Yukarıdaki getTime() fonksiyonun her iki formunu da uygulayarak basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var gecenZaman = tarihSaat.getTime();
    document.write("1 Ocak 1970 - Günümüz: " + gecenZaman);
    document.write("<BR>");
    var tarihSaat = new Date("August 21, 2005 21:53:54");
    var gecenZaman = tarihSaat.getTime();
    document.write("1 Ocak 1970 - 10 Sene öncesi: " + gecenZaman);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktısını incelediğimizde hem günümüze kadar olan zaman hem de belirlediğimiz zamanın milisaniye cinsinden dönen değerlerini görmekteyiz. Sayfamızı her Refresh ettiğimiz de günümüze kadar olan hala devam etmekte ve ucu açık olduğu için değişecektir. Diğer belirlediğimiz zaman bittiği için değişmeyecektir.

17.1.10 getTimezoneOffset()

Bildiğiniz gibi zamanımız; İngiltere Greenwich'ten geçen meridyen referans olarak kabul edilerek hesaplanmıştır. Türkiye UTC + 2 zaman diliminde fakat saatlerin bir saat ileri alınmasıyla UTC + 3 zaman dilimine geçmektedir. getTimezoneOffset() fonksiyonu da Greenwich'ten kaç saat ileri ve geride olduğumuzu gösteren bir fonksiyondur. Dönen değer dakika cinsinden olur. getTimezoneOffset() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getTimezoneOffset();
```

getTimezoneOffset() fonksiyonunu kullanarak GMT (Greenwich Mean Time)'dan dakika olarak ne kadar ileride olduğumuzu öğrenelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var geczenZaman = tarihSaat.getTimezoneOffset();
    document.write(geczenZaman);
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran alıntısında da görüldüğü gibi şuan Greenwich'ten 3 saat (180 Dk.) ilerideyiz.

17.1.11 setDate()

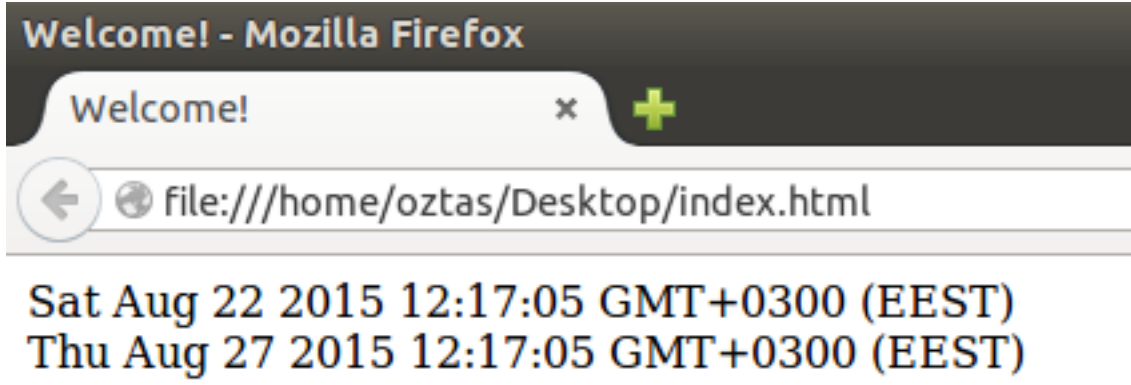
getDate() fonksiyonu bildiğiniz gibi “Bugün ayın kaç ı?” sorusuna cevap veren fonksiyondur. Çıkan değer de local'deki gün değeri idi. setDate() fonksiyonu ile bu dönen değeri değiştirebiliriz. Örneğin bugün ayın 22'si ise biz bunu 1 – 31 arasında herhangi bir değere alabiliriz. setDate() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.setDate(gunParametresi);
```

setDate() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // genel tarih degeri
    document.write(tarihSaat);
    document.write("<BR>");
    // gun degerini degistiriyoruz.
    tarihSaat.setDate(27);
    document.write(tarihSaat);
</script>
</body>
</html>
```


Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktımızda da gördüğünüz gibi; setDate() fonksiyonu ile gerçek gün değerini değiştirip başka bir tarihe aldık. Bu aldığımız tarihe göre tabiki gün ismi de değişti.

Date kütüphanesinden nesne oluştururken kendimizde belli bir tarih değeri gönderebiliyorduk, hatırlarsanız. setDate() fonksiyonu ile bu nesne oluşturulurken gönderilen parametreyi de değiştirebiliriz. Aşağıdaki örnek kodlarımız da olduğu gibi.

```
<script type="text/JavaScript">
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setDate(12); // gun degeri 12 oldu.
</script>
```

17.1.12 setMonth()

getMonth() fonksiyonu bildiğiniz gibi içinde bulunduğumuz ayın değerini bize 0 – 11 arasında bir sayı olarak döndüren fonksiyondur. Çıkan değer de local'deki ay değeri idi. Örneğin şuan aylardan ağustos'tayız. JavaScript'e göre 7'nci ay. setMonth() fonksiyonu ile bu dönen ay değerini değiştirebiliriz. setMonth() fonksiyonunun standart kullanımı aşağıdaki gibidir.

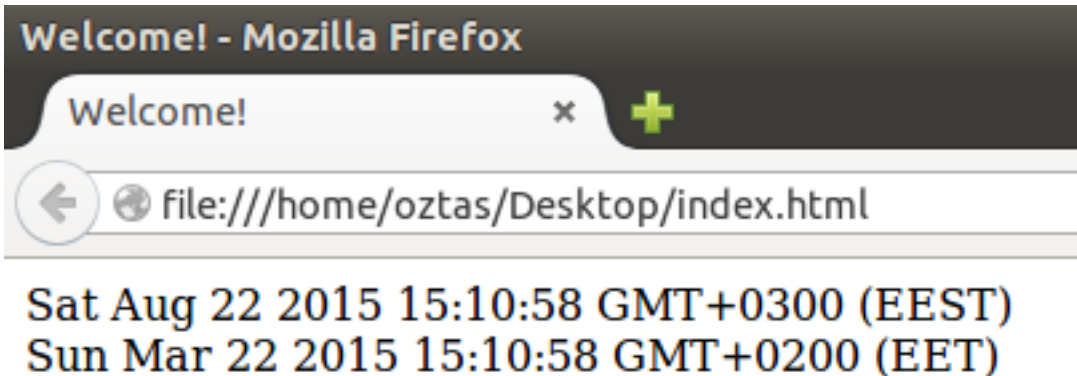
```
tarihsaat.setMonth(ayParametresi);
```

setMonth() fonksiyonu ile içinde bulunduğumuz ay'ı değiştirelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // genel tarih degeri
    document.write(tarihSaat);
    document.write("<BR>");
    // ay degerini degistiriyoruz.
    tarihSaat.setMonth(2);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
```

```
        tarihSaat.setMonth(9);
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Ekran çıktımızda da görüldüğü gibi ay değerini değiştirdiğimiz zaman yaz ve kış saati uygulaması nedeniyle GMT değeri de değişecektir.

17.1.13 setFullYear()

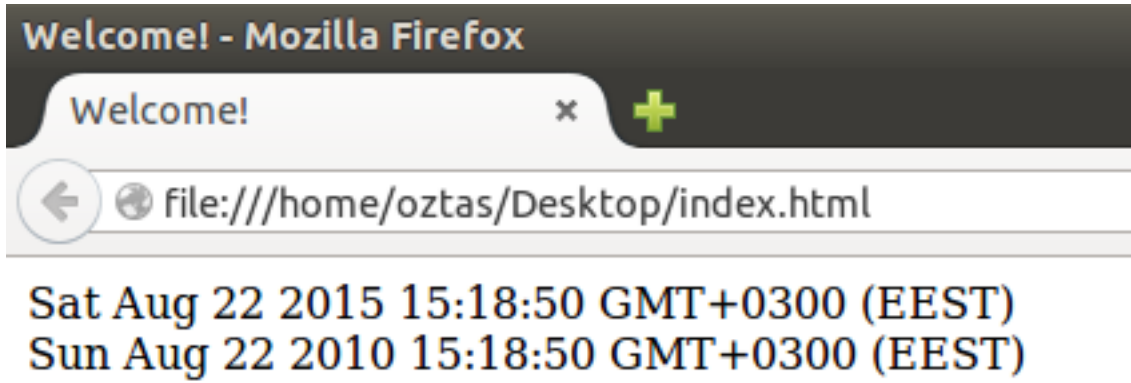
setFullYear() fonksiyonu ile şuan içinde bulunduğumuz veya nesne oluştururken parametre olarak gönderilen tarihin yıl değerini değiştirebiliriz. setFullYear() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setFullYear(yilParametresi);
```

setFullYear() fonksiyonunu kullanarak küçük bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setFullYear(2010);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setFullYear(2009);
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



17.1.14 setMilliseconds()

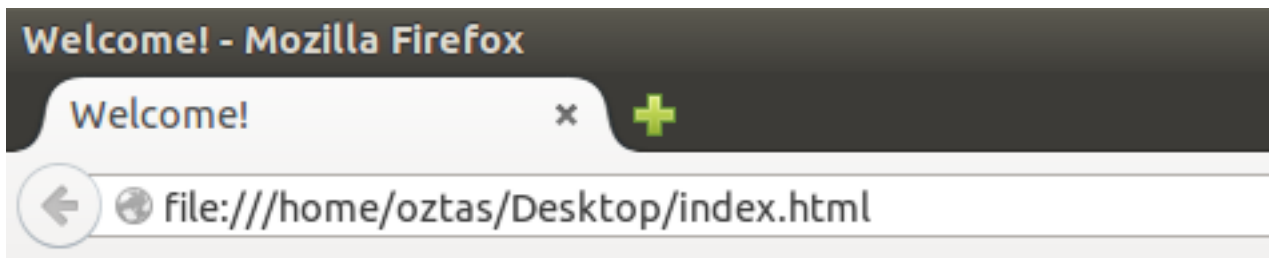
setMilliseconds() fonksiyonu ile şuan içinde bulunduğumuz veya nesne oluştururken parametre olarak gönderilen tarihin milisaniye değerini değiştirebiliriz. setMilliseconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.setMilliseconds(milisaniyeParametresi);
```

setMilliseconds() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setMilliseconds(1000);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin milisaniye degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setMilliseconds(639);
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



17.1.15 setSeconds()

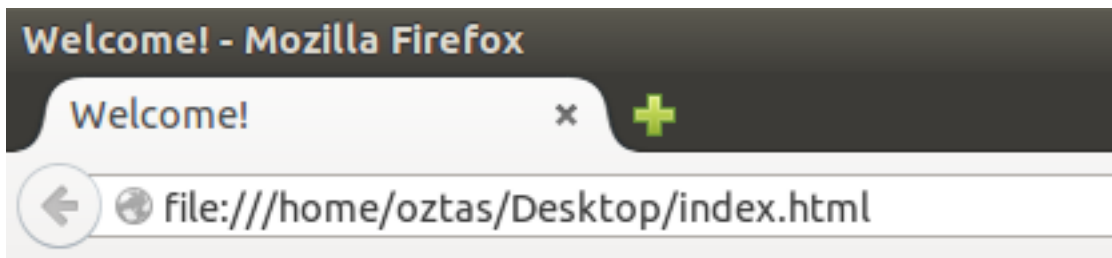
setSeconds() fonksiyonu ile şuan içinde bulunduğumuz veya nesne oluştururken parametre olarak gönderilen tarihin saniye değerini değiştirebiliriz. setSeconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.setSeconds(saniyeParametresi);
```

setSeconds() fonksiyonu ile de bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setSeconds(15);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin saniye degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setSeconds(30);
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize aiy olan ekran çıktısını alalım.



17.1.16 setMinutes()

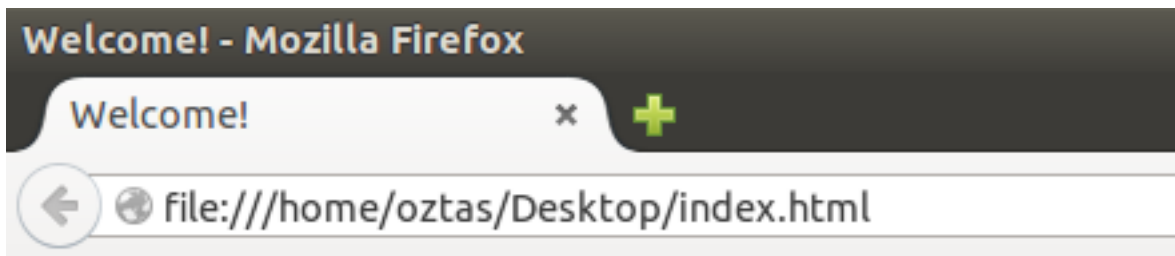
setMinutes() fonksiyonu ile şuan içinde bulunduğumuz veya nesne oluştururken parametre olarak gönderilen tarihin dakika değerini değiştirebiliriz. setMinutes() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.setMinutes(dakikaParametresi);
```

setMinutes() fonksiyonu basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setMinutes(7);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin dakika degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setMinutes(3);
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



17.1.17 setHours()

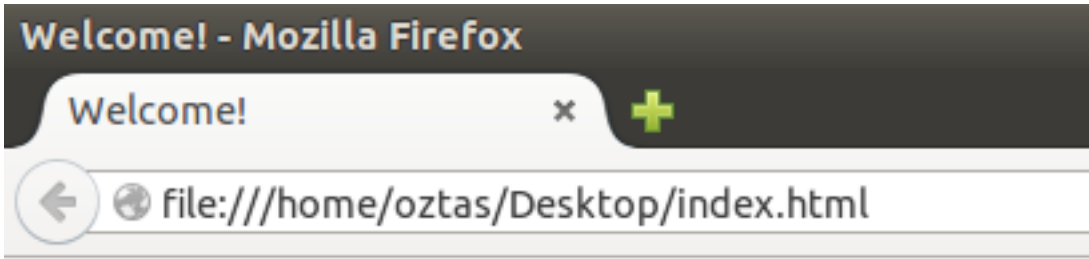
setHours() fonksiyonu ile şuan içinde bulunduğumuz veya nesne oluştururken parametre olarak gönderilen tarihin saat değerini değiştirebiliriz. setHours() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setHours(saatParametresi);
```

Her zaman olduğu gibi bilgilerimizi bir örnekle pekiştirelim. SetHours() fonksiyonu ile yaptığımız örnek aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setHours(20);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin saat degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setHours(17);
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Sat Aug 22 2015 15:35:04 GMT+0300 (EEST)
Sat Aug 22 2015 20:35:04 GMT+0300 (EEST)

Son örneğimizle birlikte; local'deki tarih ve saat ile ilgili işlemlerimizi bitirdik. İlk olarak local'de tarih ve saat ile ilgili değer nasıl alınır bundan bahsettik. Daha sonra bu local'deki aldığımız değerleri değiştirebileceğimizi gördük.

17.2 Global: Tarih ve Saat

Bu ana başlık altında göreceğimiz tarih ve saat işlemleri UTC (Universal Time Coordinated) yani dünya için genel tarih ve saat değerleridir. Global olarak belirtmemizin sebebi budur. Bu başlık altında göreceğimiz fonksiyonların döndürdüğü zaman kullanıcının bilgisayarındaki local zaman olmayıp genel tarih ve saat değerleri olacaktır. Benim bilgisayarım genel tarih / zaman ile aynı olduğu için herhangi bir değişiklik olmayacaktır. Bunun dışında Local: Tarih ve Saat başlığı altındaki fonksiyonların örneklerini UTC'ye göre revize edeceğiz. Ayrıca örneklerimize ait olan ekran çıktılarını eklemeyeceğiz çünkü çıkacak sonuçları herkes tahmin edebilecek seviyeye geldiğine inanıyorum

17.2.1 getUTCDate()

getUTCDate() fonksiyonu genel tarih değerini döndürecektir. “Yani bugün ayın kaç?” sorusuna cevap veren fonksiyondur. getUTCDate() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCDate();
```

getDate() fonksiyonunda verdiğimiz örneğimizi getUTCDate() fonksiyona göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write("UTC Gün: " + tarihSaat.getUTCDate());
</script>
</body>
</html>
```

17.2.2 getUTCDay()

getUTCDay() fonksiyonu, şuan içinde bulunduğumuz günün genel olarak haftalık bazda değerini verir. Yani daha açıklayıcı olmasını istersek; bugün haftanın hangi günü ise bu değeri bize sayısal olarak verir. getUTCDay() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCDay();
```

Hatırlarsanız; JavaScript'e göre haftanın günleri: 0 – 6 arasındaydı ve 0 nolu indis pazar gününe aitti.

getDay() fonksiyonunda verdiğimiz örneğimizi getUTCDay() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var gun = tarihSaat.getUTCDay();
    document.write("Haftanın " + gun + "nci günü. ");
    document.write("Bugün günlerden; ");
    switch(gun){
        case 0: document.write("Pazar");
        break;
        case 1: document.write("Pazartesi");
        break;
        case 2: document.write("Salı");
        break;
        case 3: document.write("Çarşamba");
        break;
        case 4: document.write("Perşembe");
        break;
        case 5: document.write("Cuma");
        break;
        case 6: document.write("Cumartesi");
        break;
    }
</script>
</body>
</html>
```

17.2.3 getUTCMonth()

getUTCMonth() fonksiyonu, içinde bulunduğumuz ay'ı genel olarak sayısal değerini döndürür. Yani “İçinde bulunduğumuz ay hangisi” sorusuna yanıt veren fonksiyondur. getUTCMonth() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCMonth();
```

Hatırlarsanız; JavaScript'e göre yılın ayları günleri: 0 – 1 arasındaydı ve 0 nolu indis ocak ayına aitti.

getMonth() fonksiyonunda verdiğimiz örneğimizi getUTCMonth() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
```



```

<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var ay = tarihSaat.getUTCMonth();
    document.write("UTC: Yılın " + ay + "nci ayı. ");
    document.write("Aylardan; ");
    switch(ay){
    case 0: document.write("Ocak"); break;
    case 1: document.write("Şubat"); break;
    case 2: document.write("Mart"); break;
    case 3: document.write("Nisan"); break;
    case 4: document.write("Mayıs"); break;
    case 5: document.write("Haziran"); break;
    case 6: document.write("Temmuz"); break;
    case 7: document.write("Ağustos"); break;
    case 8: document.write("Eylül"); break;
    case 9: document.write("Ekim"); break;
    case 10: document.write("Kasım"); break;
    case 11: document.write("Aralık"); break;
    }
</script>
</body>
</html>

```

17.2.4 getUTCFullYear()

getUTCFullYear() fonksiyonu, şuan genel olarak belirlenmiş tarihin yıl değerini döndürür. getUTCFullYear() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCFullYear();
```

getFullYear() fonksiyonunda yaptığımız örneğimizi getUTCFullYear() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var yil = tarihSaat.getUTCFullYear();
    document.write("İçinde bulunduğumuz yıl: " + yil + " yılıdır.");
</script>
</body>
</html>

```

17.2.5 getUTCMilliseconds()

getUTCMilliseconds() fonksiyonu, genel tarih ve zaman standartlarında; şuanki aktif zamanın milisaniyesini verir. getUTCMilliseconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCMilliseconds();
```

Hatırlarsanız; milisaniye değeri 0 – 999 arasında bir sayı olacaktır.

getMilliseconds() fonksiyonunda verdiğimiz örneğimizi getUTCMilliseconds() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var miliSaniye = tarihSaat.getUTCMilliseconds();
    document.write("UTC Milisaniye: " + miliSaniye);
</script>
</body>
</html>
```

17.2.6 getUTCSeconds()

getUTCSeconds() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; aktif saniyesini verir. Bu saniye değeri; 0 – 59 arasında bir sayı olacaktır. getUTCSeconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCSeconds();
```

getSeconds() fonksiyonunda vermiş olduğumuz örneğimizi getUTCSeconds() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var saniye = tarihSaat.getUTCSeconds();
    document.write("UTC Saniye: " + saniye);
</script>
</body>
</html>
```

17.2.7 getUTCMinutes()

getUTCMinutes() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; şuanki dakikasını verir. Bu dakika değeri; 0 – 59 arasında bir sayı olacaktır. getUTCMinutes() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCMinutes();
```

getMinutes() fonksiyonunda yapmış olduğumuz örneğimizi getUTCMinutes() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var dakika = tarihSaat.getUTCMinutes();
    document.write("UTC Dakika: " + dakika);
</script>
</body>
</html>
```

17.2.8 getUTCHours()

getUTCHours() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; şuanki saatini verir. Bu saat değeri; 0 – 23 arasında bir sayı olacaktır. getUTCHours() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.getUTCHours();
```

getHours() fonksiyonunda yapmış olduğumuz basit örneğimizi getUTCHours() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    var saat = tarihSaat.getUTCHours();
    document.write("UTC Saat: " + saat);
</script>
</body>
</html>
```

17.2.9 setUTCDate()

setUTCDate() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; gün değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif gün değerini 1 – 31 arasında herhangi bir gün değeri ile değiştirebiliriz. setUTCDate() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCDate(gunParametresi);
```

setDate() fonksiyonunda yapmış olduğumuz örneğimizi setUTCDate() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
```

```

</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // genel tarih degeri
    document.write(tarihSaat);
    document.write("<BR>");
    // UTC gun degerini degistiriyoruz.
    tarihSaat.setUTCDate(27);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin gun degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setUTCDate(9);
    */
</script>
</body>
</html>

```

17.2.10 setUTCMonth()

setUTCMonth() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; ay değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif ayın değerini 0 – 11 arasında herhangi bir ay değeri ile değiştirebiliriz. setUTCMonth() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCMonth(ayParametresi);
```

setMonth() fonksiyonunda yaptığımız örneğimizi setUTCMonth() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // genel tarih degeri
    document.write(tarihSaat);document.write("<BR>");
    // UTC ay degerini degistiriyoruz.
    tarihSaat.setUTCMonth(2);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin ay degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setUTCMonth(6);
    */
</script>
</body>
</html>

```

17.2.11 setUTCFullYear()

setUTCFullYear() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; yıl değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif yıl değerini herhangi bir yıl değeri ile değiştirebiliriz. setUTCFullYear() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCFullYear(yilParametresi);
```

setFullYear() fonksiyonu bölümünde yaptığımız örneğimizi setUTCFullYear() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head><body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setUTCFullYear(2010);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin yil degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setUTCFullYear(2009);
    */
</script>
</body>
</html>
```

17.2.12 setUTCMilliseconds()

setUTCMilliseconds() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; milisaniye değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif milisaniye değerini 0 – 999 arasında herhangi bir milisaniye değeriyle değiştirebiliriz. setUTCMilliseconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCMilliseconds(milisaniyeParametresi);
```

setMilliseconds() fonksiyonunda yaptığımız örneğimizi setUTCMilliseconds() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8"></head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setUTCMilliseconds(1000);
    document.write(tarihSaat);
```

```

/*
ya da asagidaki yontemle
belirlenen tarihin milisaniye degerini
degistirebiliriz.
var tarihSaat = new Date("September 27, 2010 12:13:14");
tarihSaat.setUTCMilliseconds(639);
*/
</script>
</body>
</html>

```

17.2.13 setUTCSeconds()

setUTCSeconds() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; saniye değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif saniye değerini 0 – 59 arasında herhangi bir saniye değeriyle değiştirebiliriz. setUTCSeconds() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCSeconds(saniyeParametresi);
```

setSeconds() fonksiyonunda yaptığımız örneğimizi setUTCSeconds() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title><meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setUTCSeconds(15);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin saniye degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setUTCSeconds(30);
    */
</script>
</body>
</html>

```

17.2.14 setUTCMinutes()

setUTCSeconds() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; dakika değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif dakika değerini 0 – 59 arasında herhangi bir dakika değeriyle değiştirebiliriz. setUTCMinutes() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCMinutes(dakikaParametresi);
```

setMinutes() fonksiyonunda yaptığımız örneğimizi setUTCMinutes() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setUTCMinutes(7);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin dakika degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setUTCMinutes(3);
    */
</script>
</body>
</html>

```

17.2.15 setUTCHours()

setUTCHours() fonksiyonu, genel olarak belirlenmiş dünya standartlarındaki zamanın; saat değerini değiştirmek için kullanılan bir fonksiyondur. İçinde bulunduğumuz aktif saat değerini 0 – 23 arasında herhangi bir saat değeriyle değiştirebiliriz. setUTCMinutes() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.setUTCHours(saatParametresi);
```

setHours() fonksiyonunda yaptığımız örneğimizi setUTCHours() fonksiyonuna göre revize edelim. Örneğimiz aşağıdaki gibi olacaktır.

```

<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    document.write(tarihSaat);
    document.write("<BR>");
    tarihSaat.setUTCHours(20);
    document.write(tarihSaat);
    /*
    ya da asagidaki yontemle
    belirlenen tarihin saat degerini
    degistirebiliriz.
    var tarihSaat = new Date("September 27, 2010 12:13:14");
    tarihSaat.setUTCHours(17);
    */
</script>
</body>
</html>

```

17.3 Extras

Bu başlık altında tarih ve saat formatıyla ilgili bir takım fonksiyonlar üzerinde durmaya çalışacağız. Şimdi bu başlık altındaki fonksiyonların kullanımıyla başlayalım.

17.3.1 toDateString()

Bu zamana kadar hep `Date` kütüphanesinden bir nesne oluşturduk ve bu nesneyi doğrudan ekrana yazdırdığımızda bize aşağıdaki gibi bir çıktı verdi.

```
Sun Aug 23 2015 18:10:32 GMT+0300 (EEST)
```

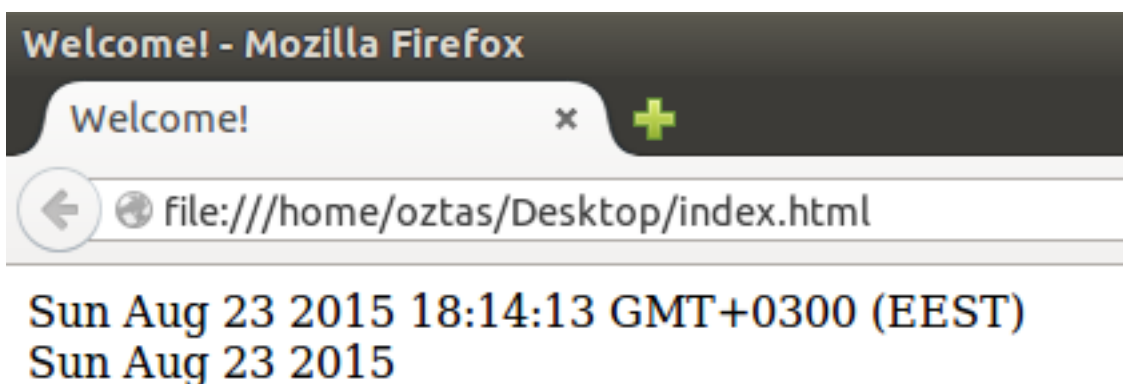
Yukarıdaki çıktı da görüldüğü gibi tarih ve saat bilgileri bir arada bulunmaktadır. `toDateString()` fonksiyonu ile sadece tarihe ait olan kısım alınmaktadır. `toDateString()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.toDateString();
```

Fonksiyonumuzla alakalı basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // normal yazım.
    document.write(tarihSaat);
    document.write("<BR>");
    // formatli yazım.
    document.write(tarihSaat.toDateString());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Yukarıdaki ekran çıktımız da görüldüğü gibi `toDateString()` fonksiyonu ile aldığımız çıktı da sadece tarih değeri yer almaktadır. Bir nevi genel formdaki tarih ve saat değerini ikiye bölüp, sadece tarihe ait olan bilgileri aldık, diyebiliriz.

17.3.2 toTimeString()

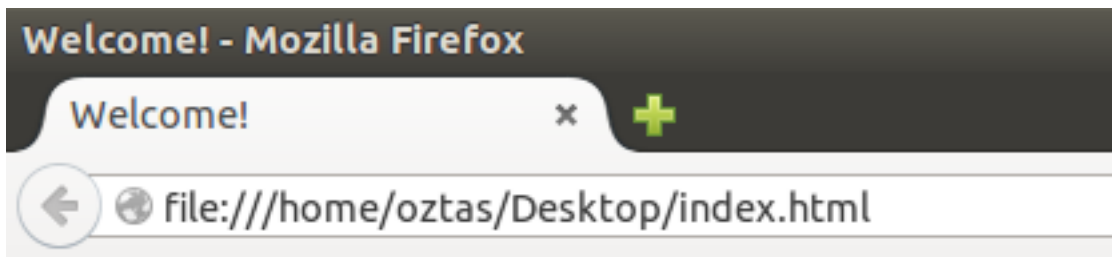
toTimeString() fonksiyonu genel tarih ve saat formatından; sadece saat değerini alarak geri döndürür. ToTimeString() fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihsaat.toTimeString();
```

toTimeString() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // normal yazım.
    document.write(tarihSaat);
    document.write("<BR>");
    // formatlı yazım.
    document.write(tarihSaat.toTimeString());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Sun Aug 23 2015 22:19:11 GMT+0300 (EEST)
22:19:11 GMT+0300 (EEST)

Yukarıdaki örnek ekran çıktımızda dda görüldüğü gibi sadece saat değerini aldık ve ekrana yazdırdık.

17.3.3 toLocaleDateString()

toLocaleDateString() fonksiyonu local'de çalışan bir fonksiyondur. Local'deki tarih değerini saat değerinden ayrıştırarak döndürür. Bir nevi kullandığımız işletim sisteminin kendi formatında yazmasıdır. toLocaleDateString() fonksiyonunun standart kullanımı aşağıdaki gibidir.

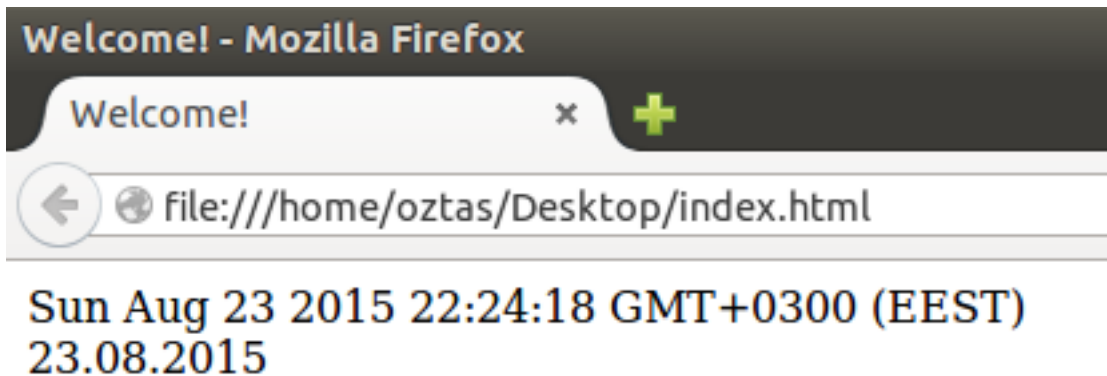
```
tarihsaat.toLocaleDateString();
```

toLocaleDateString() fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
```

```
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // normal yazim.
    document.write(tarihSaat);
    document.write("<BR>");
    // formatli yazim.
    document.write(tarihSaat.toLocaleDateString());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



17.3.4 toLocaleTimeString()

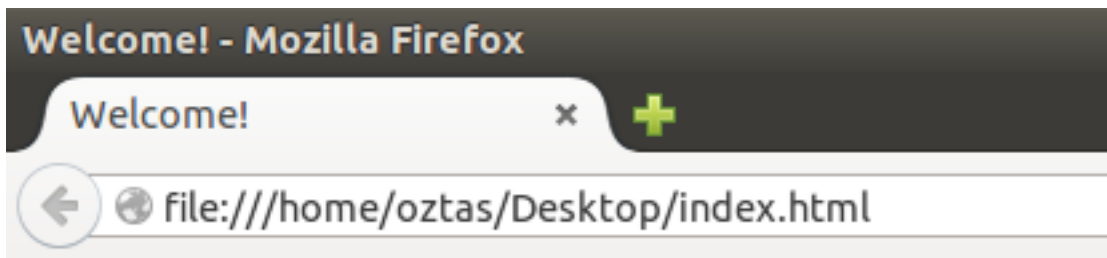
`toLocaleTimeString()` fonksiyonu local'de çalışan bir başka fonksiyondur. Local'deki saat değerini tarih değerinden ayrıştırarak döndürür. Bir nevi kullandığımız işletim sisteminin kendi formatında yazmasıdır. `toLocaleTimeString()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.toLocaleTimeString();
```

`toLocaleTimeString()` fonksiyonu ile basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // normal yazim.
    document.write(tarihSaat);
    document.write("<BR>");
    // formatli yazim.
    document.write(tarihSaat.toLocaleTimeString());
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



17.3.5 toLocaleFormat()

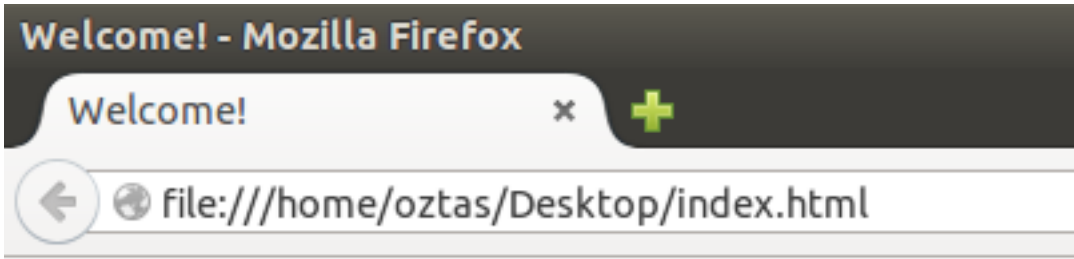
`toLocaleFormat()` fonksiyonu, genel tarih değerini klavyemizin diline göre formatlı yazdırmak için kullanılan bir fonksiyondur. Örneğin benim işletim sistemimin dili İngilizce fakat klavye düzenim: Turkish Q. Bazı tarayıcılarda bu fonksiyon çalışmayabilir, baştan belirtelim. `toLocaleFormat()` fonksiyonunun standart kullanımı aşağıdaki gibidir.

```
tarihSaat.toLocaleFormat();
```

`toLocaleFormat()` fonksiyonu bazı parametreler alarak daha formatlı bir yapıya bürübilir. Aşağıdaki örneğimizde hem parametrelili hemde parametresiz `toLocaleFormat()` fonksiyonunu kullanalım.

```
<!DOCTYPE html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
    var tarihSaat = new Date();
    // normal yazım.
    document.write(tarihSaat);
    document.write("<BR>");
    // formatli yazım.
    document.write(tarihSaat.toLocaleFormat());
    document.write("<BR>");
    //ekstra formatli yazım.
    document.write(tarihSaat.toLocaleFormat("%e %B %A - %Y"));
    /*
    %A: gun adi
    %B: ay adi
    %e: gun degeri
    %Y: yil degeri
    */
</script>
</body>
</html>
```

Yukarıdaki örneğimize ait ekran çıktısını alalım.



Sun Aug 23 2015 22:42:11 GMT+0300 (EEST)
Paz 23 Ağu 2015 22:42:11 EEST
23 Ağustos Pazar - 2015

Yukarıdaki ekran alıntısında görüldüğü gibi web browser'ımın dili Türkçe olduğu için toLocaleFormat() fonksiyonunun döndürdüğü tarih değeri de Türkçe oldu. Ayrıca bu fonksiyona gerekli parametreleri gönderdiğimiz de daha formatlı bir yapı elde edebiliriz.

BÖLÜM 18:

Errors and Exception

Kodlarımızı yazarken hataların oluşması kuvvetle muhtemeldir. Zira insan elinin değdiği hangi şey mükemmeldir ki? Hiç birşey. O yüzden hatasız program yoktur. JavaScript ortamında; Yazım Hataları, Çalışma Zamanı Hataları ve Mantıksal Hatalar gibi 3 tip hata vardır. Aslında bu hatalar genel programlama dillerine has hatalardır.

Yazım Hataları, programcılarının sıklıkla yaptıkları hatalardır. Hele de JavaScript ortamında bu tip hatalar sıklıkla başımıza gelir. Bu hatalardan; WebStorm gibi ücretli bir araçla veya Firefox için geliştirilmiş Firebug gibi bir eklentiyle kısmen kurtulmak mümkün. Size tavsiyem; Firebug eklentisini mutlaka edinin. Gerçekten olağanüstü bir araç. Bende sıklıkla kullanıyorum.

Çalışma Zamanı Hataları, kodun çalışması sırasında meydana gelen herhangi bir olayda sekteye uğramasıdır. Örneğin kullanıcıAdı adındaki bir değişkenin satırlar arasında bulunmaması fakat farklı kod bloklarında bu değişkene atıfta bulunulması gibi.

Mantıksal Hatalar, tamamen programcı tarafından kaynaklı hatalardır. Yazılan kodlar çalışır fakat üretilen sonuçlar veya işlemler yanlıştır. Örneğin matematiksel olarak; üs alma işleminde üs ve tabanın yerlerinin yanlış yazılması gibi. Bu tip hataların tek çözüm yöntemi; mantıksal hatanın olduğu yeri bulup düzeltmektir. Bu tip hataların çözümü bazen oldukça zor olabilir.

Sözün özü; hatasız program olmaz. Peki ne yapacağız? İlk olarak dikkatinizi JavaScript ortamına çekmek istiyorum. Biliyorsunuz yazdığımız kodlarımız her tarayıcıda çalışmaz. Çünkü her tarayıcının JavaScript'i destekleme oranı farklıdır. Eski tarayıcılar, JavaScript'e yeni eklenen kütüphane veya kodları desteklemez daha doğrusu destekleyemez. O yüzden web sayfalarımızın kesintiye uğramadan çalışması için bu kodları saklamamız gerekiyor. Aşağıdaki kullanım ile kodlarımızı desteklemeyen tarayıcılardan saklayabiliriz.

```
<script type="text/JavaScript">
<!--
    /*
        kodlar.
    */
//-->
</script>
```

Yukarıdaki örnek kullanımda da görüldüğü gibi <!-- //--> karakterleri arasına kodlarımızı yazabiliriz. Bu karakterler arasına yazılan kodlar sadece eski tarayıcılardan kodlarımızı saklamaz. Mantıksal hatalar dışında oluşabilecek olan diğer hatalara karşıda bir blok görevi görür. Yani meydana gelecek hata durumunda herhangi bir kesintiye uğramadan kodları çalıştırmaya devam eder. O yüzden JavaScript kodlamaya başlarken ilk olarak bu karakterler ile başlayalım.

Herhangi bir hata durumunda; hata mesajı verdirmek istiyorsak; yukarda belirttiğimiz karakterler işe yaramaz. Çünkü bu karakterler sadece hataların üzerini örter ve kodların çalışmasını durdurmaz. İşte herhangi bir hata durumunda hata mesajı verdirmek için try...catch yapısını kullanmak gerekir.

18.1 try...catch

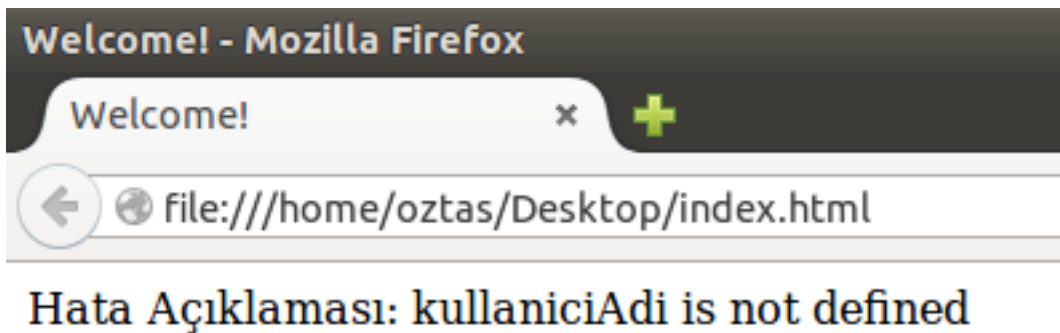
`try...catch` yapısı kullanılarak; oluşabilecek herhangi bir hata durumunda bu hatanın ne tür bir hata olduğunun mesajı alınabilir. `try...catch` yapısının standart kullanımı aşağıdaki gibidir.

```
try {  
    // kodlar.  
} catch (err) {  
    // hata durumunda yapacaklar.  
}
```

`try...catch` yapısı ile basit bir örnek yazalım. Örneğin tanımlanmayan bir değişkeni ekrana yazdırmaya çalışalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome!</title>  
<meta charset="utf-8">  
</head>  
<body>  
<script type="text/JavaScript">  
<!--  
    try {  
        document.write(kullaniciAdi);  
    } catch (err) {  
        document.write("Hata Açıklaması: " + err.message);  
    }  
//-->  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Eğer yukarıdaki örneğimizde; `try...catch` ile bir hata yakalama ve mesaj verme işleminde bulunmamış olsaydık; boş bir pencere bizi karşılayacaktı. Kısaca hatanın nedenini anlayamayacaktık. Ayrıca `.message` özelliği ile `catch(err)` parantezleri içerisinde belirtilen `err` hata nesnesinin mesajı alınır.

18.1.1 finally

`finally` anahtar kelimesi, `try...catch` yapısında son sırada bulunan bir bloktur. Bu blok; hata oluşsun oluşmasın her iki durumda da çalışan bir yapıdır. `try...catch...finally` yapısının standart kullanımı aşağıdaki gibidir.

```
try {
    // kodlar.
} catch (err) {
    // hata durumunda yapicaklar.
} finally {
    // her iki durumda da bu blok calisir.
}
```

İlk verdiğimiz örneğimize finally bloğunu da ekleyerek yeniden yazalım.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
<!--
    try {
        document.write(kullaniciAdi);
    } catch (err) {
        document.write("Hata Açıklaması: " + err.description);
    } finally{
        document.write("finally hep çalışır!");
    }
//-->
</script>
</body>
</html>
```

18.1.2 throw

throw anahtar kelimesi ile hata mesajlarını daha detaylı olarak verebiliriz. Bu da daha açıklayıcı bir hata yönetimi elde etmemize olanak sağlar. throw anahtar kelimesi try...catch bloğunda kullanılmaktadır. throw anahtar kelimesinin örnek kullanımı aşağıdaki gibidir.

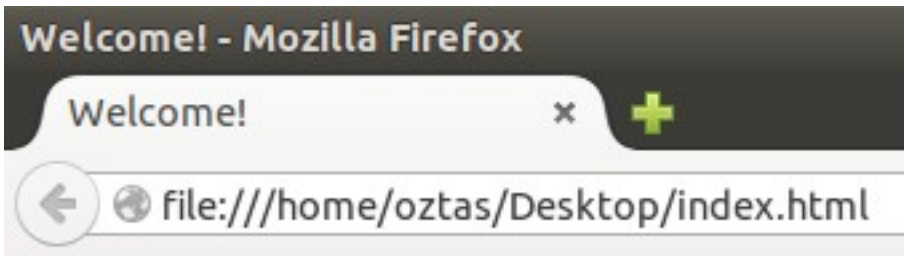
```
throw "Hatalı yazım!";
throw "HTTP STATUS 404";
throw 1453;
// veya asagidaki gibi yazilabilir.
throw("Hatalı yazım!");
throw("HTTP STATUS 404");
throw(1453);
```

throw yapısını kullanarak basit bir örnek yapalım. Örneğimiz aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome!</title>
<meta charset="utf-8">
</head>
<body>
<script type="text/JavaScript">
<!--
    var x = 15;
```

```
var y = 0;
var z;
var kullanıcıAdi = "emre";
try {
    if(y == 0) throw "Sıfıra bölme hatası!";
    else z = x / y;
    if(kullanıcıAdi != "oztas") throw "Kullanıcı adı hatalı!";
} catch (e) {
    document.write(e);
}
//-->
</script>
</body>
</html>
```

Yukarıdaki örneğimizi inceleyelim. İki tane throw yapısı kullandık. Bunun bir sınırlaması yok istenildiği kadar kullanılabilir. Fakat throw'un çalışma mantığı gereği; eğer herhangi bir hata oluşması durumunda diğer throw'lar çalışmaz. Yani kodların işletilmesi orada durur. Yukarıdaki örneğimizin ekran çıktısını alalım, ne demek istediğimi çok iyi anlayacaksınız.



Sıfıra bölme hatası!

Yukarıdaki ekran alıntısında da görüldüğü gibi throw ile ilk hata yakalandıktan sonra diğer durumlara hiç bakılmamış, kodların işletilmesi orada durdurulmuştur.

BÖLÜM 19:

External JavaScript Files

Buraya kadar olan bölümde hep JavaScript kodlarımızı HTML etiketleri arasında gömülü bir şekilde kullandık. Fakat JavaScript kodlarımızı HTML etiketleri arasında kullanmak zorunda değiliz. Harici JavaScript dosyaları oluşturarak, bu oluşturduğumuz harici JavaScript dosyalarımızı HTML sayfasında çağırarak kullanabiliriz.

HTML etiketleri arasına JavaScript kodlarımızı yazarken; fonksiyon içeren kodları `<head></head>` arasında ve fonksiyon içermeyen, basit işlemler için yazdığımız kodlarımızı da `<body></body>` etiketleri arasında yazdık. HTML etiketleri arasında JavaScript kodlarımızı kullanırken aşağıda olduğu gibi script etiketleri arasında yazdık.

```
<script type="text/JavaScript">
    // kodlar...
</script>
```

Harici JavaScript dosyalarında yukardaki script etiketlerine ihtiyacımız yoktur, yazılmaz. Ayrıca harici JavaScript dosyaları: `.js` uzantısıyla diske kaydedilirler. Örneğin;

```
// dosya ismine herhangi bir isim verebiliriz.
// .js uzantısıyla diske kaydetmeliyiz, unutmayalım.
kontrolEt.js
```

Harici JavaScript dosyalarımızı oluşturduk ve diske kaydettik diyelim. Peki HTML sayfasında bu harici dosyaları nasıl çağıracağız? Çok basit aşağıdaki yine script etiketlerini kullanacağız fakat dosyamızın yolunu `src` özelliğinde belirtmeliyiz. Aşağıdaki kullanıma bakalım.

```
<script type="text/JavaScript" src="js dosya yolu"></script>
```

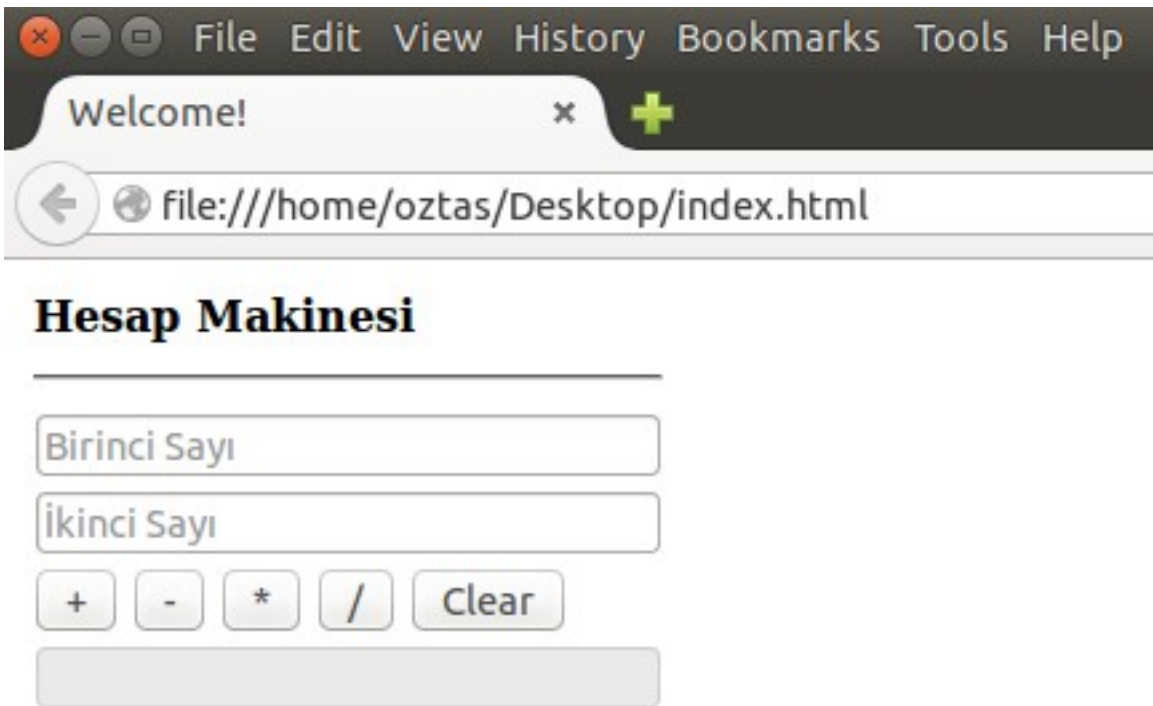
Harici JavaScript dosyalarını; `<head></head>` etiketleri veya `<body></body>` etiketleri arasında çağırabiliriz. Fakat sizin tercihiniz `<head></head>` etiketleri arasında olsun.

Harici JavaScript dosyalarını detaylı bir şekilde açıkladıktan sonra bir örnek ile anlama katsayımızı artıralım. Öncelikle örneğimizi belirleyelim. Mesela hesap makinesi yapalım. İlk olarak web sayfamızı tasarlayalım. Sayfamız aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome!</title>
</head>
<body>
<table>
<tr>
<td><b>Hesap Makinesi</b></td>
</tr>
```

```
<tr>
<td><hr></td>
</tr>
<tr>
<td><input type="text" name="s1" placeholder="Birinci Sayı" id="s1"/></td>
</tr>
<tr>
<td><input type="text" name="s2" placeholder="İkinci Sayı" id="s2"/></td>
</tr>
<tr>
<td><input type="button" value="+"/>
<input type="button" value="-"/>
<input type="button" value="*"/>
<input type="button" value="/"/>
<input type="button" value="Clear"/>
</td>
</tr>
<tr>
<td>
<input type="text" name="sonuc" id="sonuc" readonly>
</td>
</tr>
</table>
</body>
</html>
```

Tasarlamış olduğumuz sayfamızın ekran görüntüsüne bakalım.



Sayfamızın ekran görüntüsü fena değil. Tabiki CSS ile daha hoş bir tasarıma sahip olabilir. Bizde sayfamızı CSS ile düzenleyebildik lakin bu kitabın konusu CSS olmadığı için web sayfalarımıza herhangi bir müdahalede bulunmuyorum.

Şimdi harici JavaScript dosyamızı yazalım. Daha sonra yukarıdaki tasarlamış olduğumuz sayfamızda çağırabiliriz.

Harici JavaScript dosyamız aşağıdaki gibi olacaktır.

```
// harici js dosya adi: hesapMakinesi.js
var total;
var n1;
var n2;
function toplama() {
    n1 = document.getElementById("s1").value;
    n2 = document.getElementById("s2").value;
    n1 = parseInt(n1);
    n2 = parseInt(n2);
    total = n1 + n2;
    document.getElementById("sonuc").value = total;
}
function cikar() {
    n1 = document.getElementById("s1").value;
    n2 = document.getElementById("s2").value;
    n1 = parseInt(n1);
    n2 = parseInt(n2);
    total = n1 - n2;
    document.getElementById("sonuc").value = total;
}
function carp() {
    n1 = document.getElementById("s1").value;
    n2 = document.getElementById("s2").value;
    n1 = parseInt(n1);
    n2 = parseInt(n2);
    total = n1 * n2;
    document.getElementById("sonuc").value = total;
}
function bol() {
    n1 = document.getElementById("s1").value;
    n2 = document.getElementById("s2").value;
    n1 = parseInt(n1);
    n2 = parseInt(n2);
    total = n1 / n2;
    document.getElementById("sonuc").value = total;
}
function sil() {
    document.getElementById("s1").value = "";
    document.getElementById("s2").value = "";
    document.getElementById("sonuc").value = "";
}
}
```

Hazırlamış olduğumuz harici JavaScript dosyamızı, kullanacağımız web sayfasının hemen yanına konumlandıralım. Bu konuda özgürsünüz, başka herhangi bir alana da konumlandırabilirsiniz. Yeter ki dosyanın yolunu doğru bir şekilde src özelliğinde belirtebilsin.

Hazırlamış olduğumuz web sayfamızı; çağrılacak harici JavaScript dosyasına ve fonksiyonlara göre yeniden düzenleyelim. Sayfamızın son hali aşağıdaki gibidir.

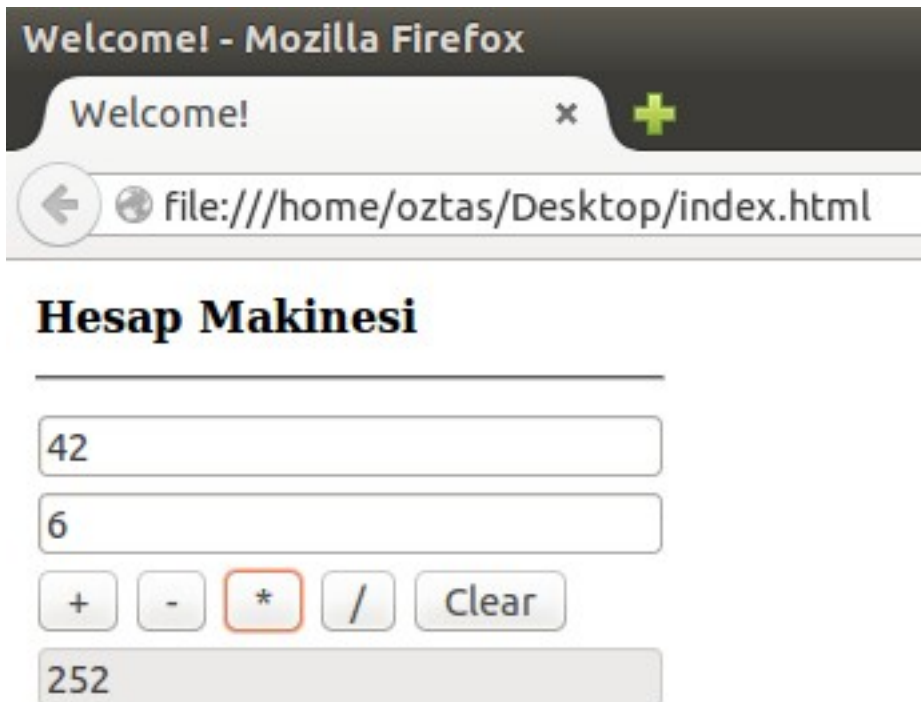
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome!</title>
<!-- harici JavaScript dosyasını cagiralim -->
<script type="text/JavaScript" src="hesapMakinesi.js"></script>
</head>
<body>
<table>
```

```

<tr>
<td><b>Hesap Makinesi</b></td>
</tr>
<tr>
<td><hr></td>
</tr>
<tr>
<td><input type="text" name="s1" placeholder="Birinci Sayı" id="s1"/></td>
</tr>
<tr>
<td><input type="text" name="s2" placeholder="İkinci Sayı" id="s2"/></td>
</tr>
<tr>
<td><input type="button" value="+" onclick="topla()"/>
<input type="button" value="-" onclick="cikar()"/>
<input type="button" value="*" onclick="carp()"/>
<input type="button" value="/" onclick="bol()"/>
<input type="button" value="Clear" onclick="sil()"/>
</td>
</tr>
<tr>
<td>
<input type="text" name="sonuc" id="sonuc" readonly>
</td>
</tr>
</table>
</body>
</html>

```

Yukarıdaki kodlarımıza dikkat ederseniz; gömülü olarak kullandığımız JavaScript kodları ile harici bir dosya olarak oluşturup kullandığımız JavaScript kodlarının kullanımında herhangi bir fark yoktur. Fark; sadece harici JavaScript dosyaları oluşturup bunların kullanılacağı web sayfasında çağırılmasıdır. Yazdığımız örneğimizin ekran çıktısını alalım.



20.1 JSON Nedir?

JSON yani JavaScript Object Notation, JavaScript'ten türetilmiş bir data barındırma, paylaşma ve taşıma formatıdır. JavaScript bir programlama dili olmadığı için JSON da kesinlikle bir programlama dili değildir.

JSON, XML gibi karmaşık bir yapıda olmayıp tam tersi herkesin okuyup anlayabileceği bir yapıdır. Ayrıca veri paylaşımında XML dosyalarından daha az bir yer kaplamaktadır. Bu da onu son yıllarda XML'in bir adım önüne geçirmiştir.

JavaScript ile kullanılması öngörülerek geliştirilen JSON, diğer programlama dilleri; Java, Python, PHP, C# gibi bir çok programlama dili ile birlikte de kullanılabilir durumdadır. Dolayısıyla tamamen herhangi bir dile bağlı değildir.

20.2 JSON Kullanımı

JSON, harici bir dosya olarak: .json uzantısı ile diske kaydedilir.

```
jsonDosyaAdi.json
```

Ya da HTML sayfalarında; script etiketleri arasında kullanılır.

```
<script type="text/JavaScript">  
    // JSON ifadeler.  
</script>
```

JSON'un çok basit bir söz dizimi vardır ve anlaşılması kolaydır. Örneğin aşağıdaki JSON satırlarını açıklayarak söz dizimini anlamaya çalışalım.

```
{ "adi": "emre can", "soyadi": "oztas", "numarasi" : 15 }
```

Yukarıdaki satırımızı inceleyelim. adi, soyadi, numarasi alanları birer attribute (özellik) görüldüğü gibi. Bu herhangi bir veri satırıdır.

Yukarıdaki JSON satırını biraz daha genişletelim.

```
{  
  "ogrenci" : [  
    { "adi": "onur", "soyadi": "duyar", "numarasi": 21 },  
    { "adi": "tümer", "soyadi": "şibik", "numarasi": 41 },  
    { "adi": "recep", "soyadi": "saygili", "numarasi": 12 },  
    { "adi": "savaş", "soyadi": "ev", "numarasi": 52 }  
  ]  
}
```

Yukarıdaki JSON satırlarımızı inceleyelim. Toplam 4 veri satırı var. öğrenci, bizim nesnemin adı. Yani bir entity gibi düşünelim. adı, soyadı, numarası alanları ise birer attribute yani öğrenci entity'sinin birer özellikleri. Bu örneğimizde başka bir özellikten bahsedelim. Dikkat ederseniz hem metinsel (String) hemde sayısal olarak ifadelerimiz JSON veri satırlarında bulunabiliyor. Burada hareketle; diğer veri tipleri de: boolean, null, float v.s gibi bulunabilir.

Veri satırlarımıza yeni veriler ve alanlar ekleyelim.

```
{
  "ogrenci" : [
    { "adi": "onur", "soyadi": "duyar", "numarasi": 21 },
    { "adi": "tümer", "soyadi": "şibik", "numarasi": 41 },
    { "adi": "recep", "soyadi": "saygılı", "numarasi": 12 },
    { "adi": "savaş", "soyadi": "ev", "numarasi": 52 }
  ],
  "yonetici": [
    { "adi": "süleyman", "soyadi": "altan", "bolum": "fizik"},
    { "adi": "yasin", "soyadi": "sandıkçı", "bolum": "işletme"},
    { "adi": "muhammet", "soyadi": "küçükbarıdaslı", "bolum": "tıp"}
  ]
}
```

Yukarıdaki satırlar söylemek istediklerimize tercüman olacaktır. Öncelikle 2 tane entity var. Birinci entity yani öğrenci, 4 tane veri satırı barındırmakta, ikinci entity yani yönetici 3 veri satırı barındırmaktadır. öğrenci entity'sinin alanları: adı, soyadı ve numarası, yönetici entity'sinin alanları: adı, soyadı ve bölüm. Buraya kadar herşey tamam fakat bu verilerin bir sahibi bir anlamda root'u yok. Root belirlemek içinde bir Object (nesne) oluşturmalıyız.

Bu bölümde; JSON'u genel olarak HTML etiketleri arasında göreceğiz. Şimdi JavaScript etiketleri arasında JSON yazmaya ve okumaya başlayalım. İlk olarak bize bir nesne lazım olacaktır. Bu nesnemizi aşağıdaki gibi tanımlayabiliriz.

```
var jsonObject = new Object();
```

Ya da

```
var jsonObject = {};
```

En son yazdığımız veri satırlarımıza bir root belirleyelim. Aşağıdaki örneğimize bakalım.

```
<script type="text/JavaScript">
var okul = {
  "ogrenci" : [
    { "adi": "onur", "soyadi": "duyar", "numarasi": 21 },
    { "adi": "tümer", "soyadi": "şibik", "numarasi": 41 },
    { "adi": "recep", "soyadi": "saygılı", "numarasi": 12 },
    { "adi": "savaş", "soyadi": "ev", "numarasi": 52 }
  ],
  "yonetici": [
    { "adi": "süleyman", "soyadi": "altan", "bolum": "fizik"},
    { "adi": "yasin", "soyadi": "sandıkçı", "bolum": "işletme"},
    { "adi": "muhammet", "soyadi": "küçükbarıdaslı", "bolum": "tıp"}
  ]
}
</script>
```

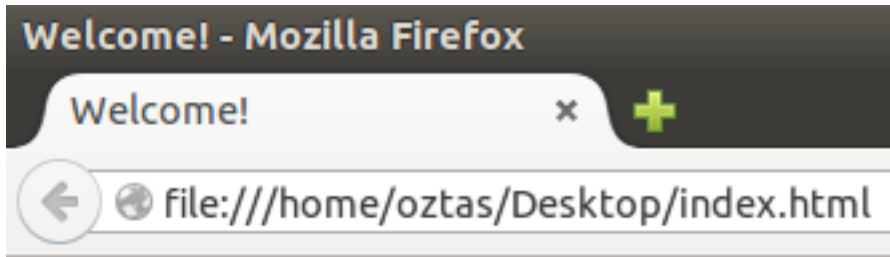
Yukarıdaki örneğimizde görüldüğü gibi, oluşturmuş olduğumuz JSON veri satırlarımızı okul nesnesine bağladık. Burada okul kelimesi veri tabanının adı gibi de düşünülebilir. Şimdi gelelim bu JSON verilerimizi çağırma. Aşağıdaki örnek kullanıma bakalım.

```
nesneAdi.tabloAdi[kayitNo].tabloAlani;  
// bir degiskene baglasak daha iyi olacak.  
var kayit = nesneAdi.tabloAdi[kayitNo].tabloAlani;
```

JSON veri formatlarımızı, HTML etiketleri ile kullanacağımızı söylemiştik. Web sayfamızın son hali aşağıdaki gibi olacaktır.

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Welcome!</title>  
<meta charset="utf-8">  
</head>  
<body>  
<script type="text/JavaScript">  
var okul = {  
    "ogrenci" : [  
        { "adi": "onur", "soyadi": "duyar", "numarasi": 21 },  
        { "adi": "tümer", "soyadi": "şibik", "numarasi": 41 },  
        { "adi": "recep", "soyadi": "saygili", "numarasi": 12 },  
        { "adi": "savaş", "soyadi": "ev", "numarasi": 52 }  
    ],  
    "yonetici": [  
        { "adi": "süleyman", "soyadi": "altan", "bolum": "fizik"},  
        { "adi": "yasın", "soyadi": "sandıkçı", "bolum": "işletme"},  
        { "adi": "muhammet", "soyadi": "küçükbarıdaslı", "bolum": "tıp"}  
    ]  
}  
// ogrencileri ekrana yazdiralim.  
document.write("Öğrenci Bilgileri: " + "<BR>");  
for(var i = 0; i < okul.ogrenci.length; i++){  
    document.writeln(okul.ogrenci[i].numarasi + ". ");  
    document.writeln(okul.ogrenci[i].adi);  
    document.writeln(okul.ogrenci[i].soyadi);  
    document.write("<BR>");  
}  
document.write("<BR>");  
// yoneticileri ekrana yazdiralim.  
document.write("Yönetici Bilgileri: " + "<BR>");  
for(var i = 0; i < okul.yonetici.length; i++){  
    document.writeln(okul.yonetici[i].adi);  
    document.writeln(okul.yonetici[i].soyadi + " - ");  
    document.writeln(okul.yonetici[i].bolum);  
    document.write("<BR>");  
}  
</script>  
</body>  
</html>
```

Yukarıdaki örneğimize ait olan ekran çıktısını alalım.



Öğrenci Bilgileri:

21. onur duyar

41. tümer şibik

12. recep saygili

52. savař ev

Yönetici Bilgileri:

süleyman altan - fizik

yasin sandıkçı - işletme

muhammet küçükbarıaslı - tıp

JSON yapısını kullanarak çeřitli; veri iletimi ve veri alınması işlemini gerçekleřtirebilirsiniz. JSON çok kullanışlı bir yapıya sahip olmakla beraber; herhangi bir insanın yazılan kodlara bakıp kolayca anlayabilmesi muhtemel. JSON konusunda söyleyeceklerimiz bu kadar. Daha doğrusu JavaScript içerisinde JSON için söyleyeceklerimiz bu kadar.

BÖLÜM 21:

JavaScript Code Compress

Bir web sayfasına istek gönderildiği zaman, bu web sayfasının içeriği bilgisayarınıza indirilir. Çünkü tarayıcıda görüntülecektir. Dolayısıyla web sayfasının bir tarayıcı için getirdiği yük ne kadar az olursa sayfanın yüklenmesi ve gösterilmesi o oranda kısa olacaktır. Bu daha çok Responsive ve tasarım için gereklidir. Bildiğiniz gibi responsive tasarımda; web sayfasının çeşitli cihazlarda kullanımına göre sayfanın ve içerik elementlerinin değişmesidir. Bu cihazlar; bilgisayarlar, akıllı telefonlar, tabletler v.s olabilir.

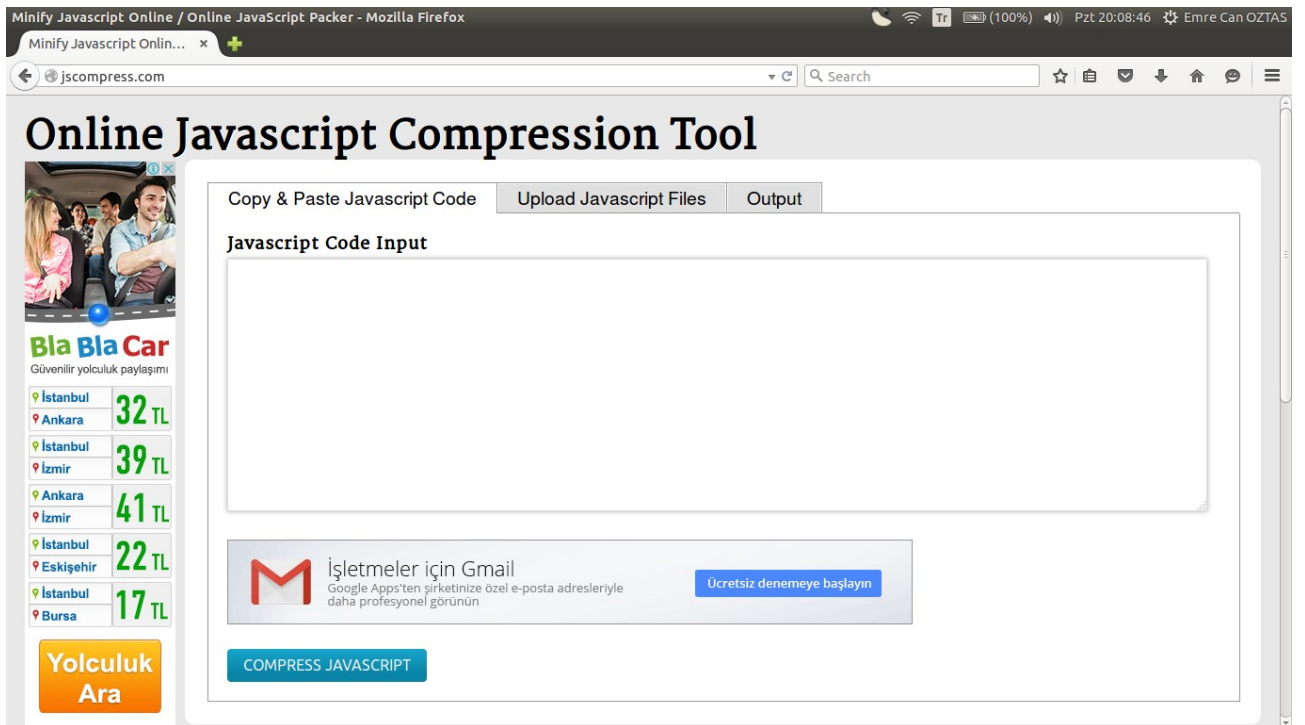
Kullanıcının bağlandığı web sayfasının yüklenme hızı önemlidir. Çünkü hiç bir kullanıcı geç yüklenen bir sayfada kalmak istemez ve sayfayı terk eder. Bu da biz geliştiriciler için kötü bir durumdur. Web sayfasının yükünü hafifletmemiz hayati bir ihtiyaç olmaktadır.

External (Harici) olarak kullanılan JavaScript dosyalarının boyutlarını küçültmemiz gerekir. CSS ve HTML sayfalarının boyutlarını da azaltabiliriz. Fakat bu kitabın konusu JavaScript olduğu için şuan sadece JavaScript dosyaları üzerinde duracağız.

Peki JavaScript dosyalarının boyutlarını nasıl azaltabiliriz? Bunun için geliştirilmiş çeşitli araçlar ve web siteleri mevcuttur. Benim beğendiğim ve kullandığım bir siteyi sizlerle paylaşmak isterim. Sizlere tavsiye etmek istediğim web sayfasına aşağıda vermiş olduğum linke tıklayarak ulaşalım.

<http://jscompress.com/>

Yukarıdaki adrese gittik ve aşağıdaki sayfa açıldı.



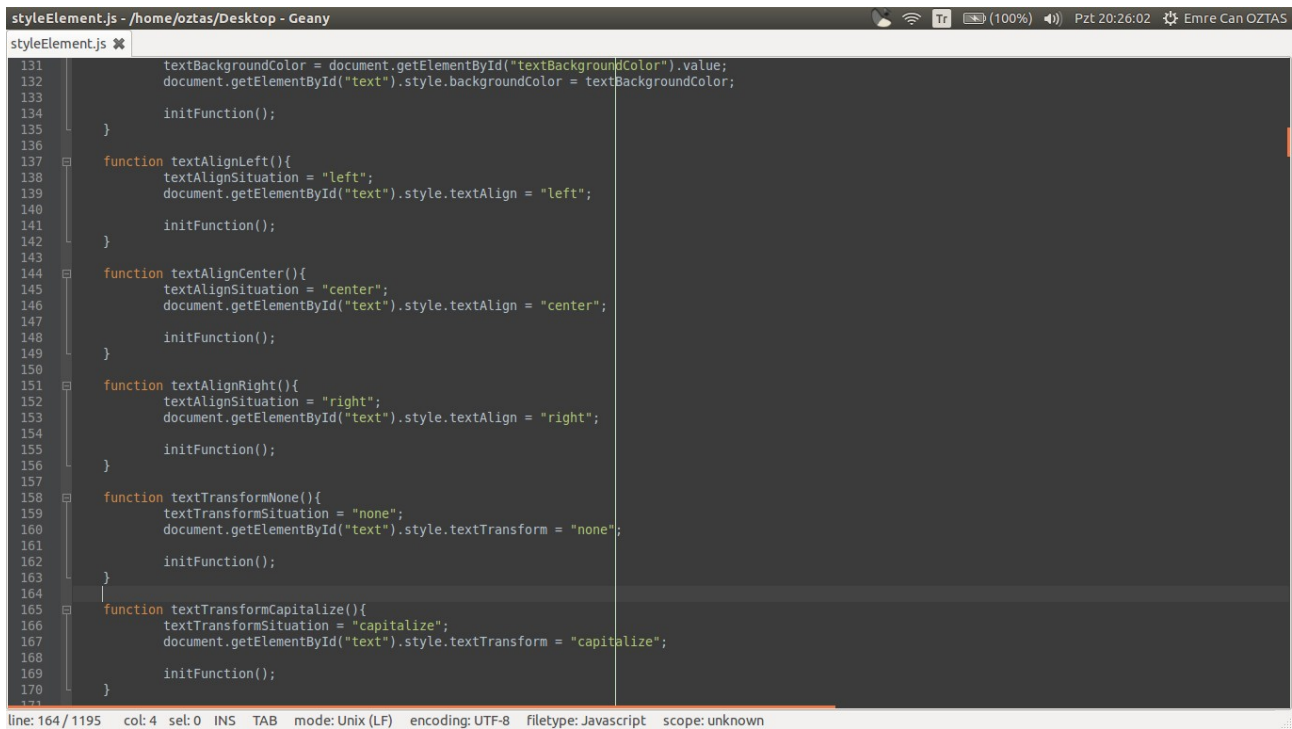
Görüldüğü gibi bol reklamli bir site. Neyse bunun konumuzla bir alakası yok.

Açılan sayfadan bahsetmek isterim. Görüldüğü gibi ana ekranda;

- **Copy & Paste JavaScript Code**
JavaScript kodlarınızı bu sekmedeki alana yapıştırarak; kodlarınızın compress (sıkıştırılması) işlemini gerçekleştirebilirsiniz.
- **Upload JavaScript Files**
Harici olan JavaScript dosyalarınızı bu sekmeden sisteme yükleyerek sıkıştırılma işlemini gerçekleştirebilirsiniz. Birden fazla JavaScript dosyası yükleyebilirsiniz. Sistem bu dosyaları birleştirecektir.
- **Output**
kopyala / yapıştır yapılarak eklenen JavaScript kodları veya harici JavaScript dosyalarının sisteme yüklenip, sıkıştırılma işlemi tamamlandıktan sonra bu alanda JavaScript kodlarımızın çıktıları gösterilir.

alanları vardır.

Örneğin benim elimde, daha önce bir projede kullandığım harici JavaScript dosyası var. Örnek olması açısından bir ekran alıntısı ekliyorum.



```
styleElement.js - /home/oztas/Desktop - Geany
styleElement.js
131 textBackgroundColor = document.getElementById("textBackgroundColor").value;
132 document.getElementById("text").style.backgroundColor = textBackgroundColor;
133
134 initFunction();
135 }
136
137 function textAlignLeft(){
138 textAlignSituation = "left";
139 document.getElementById("text").style.textAlign = "left";
140
141 initFunction();
142 }
143
144 function textAlignCenter(){
145 textAlignSituation = "center";
146 document.getElementById("text").style.textAlign = "center";
147
148 initFunction();
149 }
150
151 function textAlignRight(){
152 textAlignSituation = "right";
153 document.getElementById("text").style.textAlign = "right";
154
155 initFunction();
156 }
157
158 function textTransformNone(){
159 textTransformSituation = "none";
160 document.getElementById("text").style.textTransform = "none";
161
162 initFunction();
163 }
164
165 function textTransformCapitalize(){
166 textTransformSituation = "capitalize";
167 document.getElementById("text").style.textTransform = "capitalize";
168
169 initFunction();
170 }
171 }
```

line: 164 / 1195 col: 4 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: Javascript scope: unknown

Yukarıdaki JavaScript dosyasının boyutu:

styleElement.js: 37,4 kB

Şimdi dosyamızı sisteme yükleyelim. Örnek olması açısından ekran alıntısı aşağıdaki gibidir.

Dosyamızın sıkıştırılmış hali aşağıdaki gibidir.

Görüldüğü gibi tek bir satır haline getirilmiş ve aralardaki boşluklar kaldırılmış. Yani olabildiğince minimalist hale getirilmiş.

JavaScript dosyamızın boyutu:

output.min.js: 28,5 kB

JavaScript dosyamızdan veri kazancımız; 8.9 kB. Bu sizin gözünüzde çok küçük bir rakam olarak gözükebilir, haklısınız. Lakin büyük resmi görmek önemli. Bazen öyle büyük projelerde çalışırsınız ki belki 30 belki 40 veya daha fazla harici JavaScript dosyaları oluşturmak zorunda kalabilirsiniz. Bunun içine bir de CSS dosyalarını da katarsanız, veri kaybı oldukça büyümeye başlar. Dolayısıyla web sitenizin yüklenme oranı düşer, bu da can sıkıcı durumları doğurur. Zira hiç kimse sitesinden kullanıcının kaçmasını istemez.

Kaliteli bir web sayfası için doğru kodlama ve performans önemlidir. Bu bölümde önemli bir performans kazancı yaptık. Bundan sonraki zamanlarınızda hazırlayacağınız JavaScript dosyalarınızı minimalist şekle getirmenizi önemle tavsiye ederim.

<http://www.tutorialspoint.com/javascript/index.htm>
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>
<http://www.w3schools.com/js/>
<https://www.javascript.com/>
<https://www.codeschool.com/paths/javascript>