phpdergi.com

<?php sayı : 1 ?>

Yeni bir yolculuğun tadını çıkartın...

- jQuery kim korkar javascript'ten. jQuery üzerine kısaca uzun bir yazı. - Ali GÜNDOĞDU
- Programlamaya Giriş ve Örnekler - Sinan İŞLER
- Sistem Nedir ?- Ali GÜNDOĞDU
- Programcının yoğurt yiyişi: "Algoritma" - Bahriye Sarıkaya
- Zend Framework Helper kullanımı Örnek: Minfy - Volkan ALTAN
- PHP Kodlama Standartları- Cankat Akdemir
- URL'ler Hakkında Bilmeniz Gereken Herşey- Cankat Akdemir

önsöz...

PHP artık eski PHP değil. 2003'e kadar sadece kişisel kullanımla kısıtlı kalan PHP, 2003'ten sonra Web 2.0 dalgası ve gelişen özellikleriyle birçok alanda kullanılmaya başladı. Her gün milyarlarca sayfa gösterimine sahip olan Facebook'tan bilgilerinizi emanet ettiğiniz bankanızın web sayfasına kadar her yerde PHP'yi görmek mümkün.

PHP'nin bu kadar kullanım oranına ulaşmasındaki en büyük faydalardan biri kolay öğrenilir olarak bilinmesi oldu. Bu sayede kısa sürede geliştirilmiş fakat ileri seviye birçok uygulama görmeye başladık. Her şey varsayılan haliyle derin düşünmeden tasarlandığında gerçekten PHP ile uygulama geliştirmek de kolaydı.

İnternette yer alan hazır kodlar ya da temel seviye eğitim belgelerinde görülenler geniş ölçekli bir uygulama geliştirmek için bile yeterli. Fakat sorun da bundan sonra başlıyor. Her şeyi varsayılan ayarlarında bırakmak, onlarca yıldır yanıla yanıla bulunan doğrulara uymadan geliştirilen uygulamalar bir yerden sonra maalesef patlıyor.

SQL Injection, XSS, CSRF ve daha nice güvenlik açıkları artık kolay şekilde önlenebiliyor ve çok ileri seviye hack yöntemi olmadıkları da aşikâr.

Şu an ilk satırlarını okuduğunuz bu dergi sizlere web uygulamaları geliştirirken yardımcı olmayı hedefliyor. Sadece PHP değil, web uygulaması geliştirirken ihtiyaç duyduğunuz birçok alanda yazılar bulabileceksiniz dergide. JavaScript'ten sistem yönetimine, web sunucu kurulumundan CSS'e kadar hem basit hem ileri seviye birçok yazı okuyacaksınız dergide.

Elinizin altında internet adı verilen hemen her şeyi bulabileceğiniz bir kaynak var. Derginin farkı ise hiçbir yerde olmayan konular ya da fikirler içermesi olacak. Özellikle yukarıda belirttiğimiz gibi deneyimlerle kazanılan bilgiler, ipuçları ve web uygulamalarının güvenliği üzerine yazılara sık sık rastlayacaksınız.

Dergimize siz de katkıda bulunabilirsiniz. Tüm soru, öneri ve yazılarınızı phpdergi@gmail.com adresine gönderebilirsiniz.

İyi okumalar.

phpdergi

jQuery kim korkar javascript'ten. jQuery üzerine kısaca uzun bir yazı.

Ali GÜNDOĞDU

jQuery ismi Javascript ve Query isimlerinin erimesi ile ortaya çıkmış yeni bir kelimedir

jQuery Nedir ?

jQuery, Javascript'in kimi zaman karışık ve meşaketli gelen yazım düzenini, basite indirgeyerek çok hızlı ve kolay yoldan sonuca ulaşmanızı sağlayan javascript kütüphanesidir.
Bu kütüphane kendi içerisinde, Ajax, Koşullar, Efekt gibi zengin özellikleri içerisinde barındırmaktadır. Belkide en önemli özelliği olması, geliştirilmeye müsait bir yapısının olmasıdır. jQuery için her ihtiyaca yönelik geliştirmeler (Plugin) mevcuttur ve bu geliştirmeler sayesinde sonuca dahada hızlı ulaşabilirsiniz.

Javascript içerisinde hakeza bir yanlışlık yaptığınızı varsayalım, hemen ilgili tarayıcınızın JavaScript hata ibaresi ekranın uygun yerinde belirir ve sayfa içerisindeki hiç bir script (betik) kodu çalışmaz. Bu durum da jQuery'de en aza indirgenmiştir.

jQuery ismi Javascript ve Query isimlerinin erimesi ile ortaya çıkmış bir yeni bir kelimedir,temelde yaptığı şey ise Dom nesnesine query (sorgu) göndererek seçim yapmak ve seçilen nesnelerin üzerinde işlem yapmaktır.

Son olarak jQuery 2006 yılının John Resig tarafından duyurulmuş ve o gün için onbeş (15) kişilik bir ekip ile geliştirilmeye başlanmış.

Seçimler Sorgular

jQuery Yazım diline ufak ufak bir giriş yapalım dilerseniz,
şimdiye kadar gerek burada, gerekse bir çok internet dizininde jquery'e metiye düzenlerin hepsinin kullandığı ortak kelimeler, kolay ve hızlı kelimeleri olmuştur. Peki nedir bu kolaylık ve hız ?
Hızın temelinde html nesnelerini zahmetsiz şekilde yakalama ve üzerinde işlem yapma isteği yatar, javascript ile bir nesneyi seçmek istediğimizde o nesnenin ID değerine göre işlem yapmamız gerekmektedir,

diyelim ki ID değeri phpdergi olan bir div nesnesini yakalamak istedik;

Javascript ile :

```
var nesnemiz = document.getElementByID("phpdergi");
```

aynı örnek jQuery ile :

```
var nesnemiz = $("#phpdergi");
```

bakın bakın olay ne kadarda kısaldı dememi bekliyorsunuz ama burada anlatmak istediğim yarıyarıya kısalmış karakter sayısı değil (elbette oda var). Bir nesnenin kendisini bu şekilde yakaladık.
Şimdi gelelim jQuery'nin sihirine,
ID değeri verilen ve kimi seçeğimiz belli durumlarda öyle ahım şahım bir üstünlüğü yokmuş gibi gözüken bu kütüphanenin belirsiz ve çoklu seçimlerdeki üstünlüğüne bir bakalım dilerseniz ;
bir sayfa içerisinde body hyperlink (a) nesnelerinin class isimlerini "ikinci" ismi ile değiştirmek istiyoruz diyelim;
jquery ile

```
$("a").each(function(){$(this).className="ikinci";});
```

bu seçim ve işlem vesilesi ile isterseniz onlarca nesnenin bir anda class name değerini değiştirebilirsiniz. peki bunu JavaScript ile nasıl yaparıza örnek vermiyorum.
jQuery **$("seçici")** yazım şekli ile nesneleri seçer ve sonrasındaki ekler ile seçimi genişletir ve işlem yapar, şimdlik çok basit olarak değinmek gerekirse (ileride çok detaylı şekilde anlatacağım) seçiciler kabaca eğer id değerine göre seçim yapacak isek,seçici kısmının başına # işaretini eklemeliyiz. (yukarıdaki örneklerde olduğu gibi)
eğer class name değerine göre işlem yapacaksak seçici kısmının başına . işaretini eklemeliyiz ;

$(".class") class değeri "class" olan tüm nesneleri yakalar örnek olarak :

```
<a href='phpdergi.com' class='class'>Php dergi link 1</a>
<a href='phpdergi.com' class='sinif'>Php dergi link 2</a>
<a href='phpdergi.com' class='ders'>Php dergi link 3</a>
```

bu jquery seçim kodu sadece php dergi link 1 isimli link nesnesini seçecektir.

bu ön girişten sonra seçimler ve sorgular kısmını biraz açalım,

jQuery selectors ismi ile anılan ve türkçeye seçici veya seçim sorguları şeklinde çevirebileceğimiz bu olay, jquery ile javascript'in ayrıldığı ilk ve en önemli noktadır.

jQuery içerisinde belli başlı sorgu metodları bulunmaktadır bunlar ;

*** Stil Sorguları**

bu sorgu çeşidinde Css dosyası ile bağladığımız etiketleri kullanarak seçim işlemi yapabiliriz,
yukarıda verdiğim ilk örnekteki sorgu çeşidi ile aynıdır.

css temelinde yaygın olarak kullanılan etiketlemeler içerisinde
eğer bir nesneye direk olarak bir atama yapacak isek, nesne global adına özellik atarız,

```
div {
/*özellikler....*/
}

ID değeri belirtilmiş nesnelere atama yapacaksak
#idlinesne {
/*özellikler...*/
}

Class değeri belirtilmiş nesnelere atama yapacaksak
.classlinesne {
/*özellikler...*/
}
yada bir etiketle bağlanmış başka nesneler altındaki belli gruptaki (class,etiket,id) nesnelere ulaşıp atama yapacaksak
div.classlinesne {
/*özellikler...*/
}
#deneme a span.class1 {
/*özellikler...*/
}
.classdenemesi div.divclass a:hover {
/*özellikler...*/
}
```

örneklerde açıklamaya çalıştığım class etiket dizimlerinide aynı şekilde jQuery sorgu seçimi sırasında kullanabiliriz.

```
$("div") //tüm divler seçilir.
$("div a") //div altındaki tüm hyperlinkler seçilir
$(".classisim") // tüm classisim değerindeki nesneler seçilir
$("#etiket") //etiket değeri seçilir.
```

Önemli :: Etiket değeri bir çok html standardına göre bir (1) nesneye verilebilir o yüzden birden fazla etiket ataması yapmayacak şekilde bir desen oluşturmanız ileride oluşması muhtemel sorunların önüne geçecektir.

Bilgilendirme : Html nesnelerine ait seçimler bir çok kaynakta ayrı bir dal altında olabilir, ben burada stil sorgusu içerisine ekledim,buna ek olarak bu seçimlere ek olarak virgül (,) karakteri ile birden fazla html nesnesini seçebilirsiniz.

```
$("div,a,span") //divleri hyperlinkleri ve spanları seçti.
```

*** sembol veya dayandırmalı sorgular:**

hyperlink (a) nesnesinin href değerine atıfta bulunarak anlatmak gerekirse bu değer hyperlink nesnesinin sembolüdür.
ingilizce **attribute** kelimesini sözlükten çevirirsek nitelik, atfetmek gibi anlamları vardır ancak özellik veya nitelik türkçe olarak tamda bu anlamı taşımıyor onun yerine *sembol* kullanmak daha mantıklı.

bu sorgu çeşidi bir nevi regex yada sqldeki like gibi çalışır, bellir bir nesnenin sembol değerleri içerisinde belirttiğimiz regex sorgusu ile aynı olanları seçer ve bize sunar.

```
<a href='www.phpdergi.com'>AnaSayfa</a>
<a href='http://www.phpdergi.com'>AnaSayfa</a>
<a href='index.php?olay=anasayfa'>Anasayfa</a>
<a>anasayfa</a>
```

html kodları üzerinden gidecek olursak,
sembolleri, www ile başlayan hyperlink (a) nesnelerini seçmek istersek

```
$("a[href^=www]")
```

sorgusunu kullanmamız yeterli. Sembol değeri içerisinde www geçen nesneler için ise,

```
$("a[href*=www]")
```

sorgusu, sembol değeri sonu anasayfa ile biten nesneler için ise,

```
$("a[href$=anasayfa]")
```

sorgusunu kullanmamız yeterli.

sadece href sembol değeri olanları seçmek içinse

```
$("a[href]")
```

sorgusu yeterli olacaktır.

bu sorgulama yönetimi örnekler ile açıklamanın daha faydalı olacağını düşündüğüm için bu şekilde açıkladım.

bir diğer sorgulama metodu ,

*** Form eleman sorguları** :

Bir çok yazılımcının, tasarımcının en büyük korkulu rüyası arabirim tasarlama sırasında defalarca yazmak zorunda olduğumuz form elemanlarıdır.
her yazdığımız elemanın akibetini kontrol etmek bizim vazifemizdir.İşte bu yüzden oluşturduğumuz projelerin büyük bir kısmı kullanıcı ile etkileşmemiz (onlardan bilgi alıp işlem yapma) gereken yerlerde geçer. jQuery ile oluşturduğunuz form nesnelerine ulaşmanız mümkün.

bunlar ;
:input <input etiketi ile başlayan bütün form elemanlarını seçer.
:text <input type='text' olan tüm metin kutularını seçer.
:file <input type='file' olan tüm dosya seçim kutularını seçer.
:password <input type='password' olan tüm şifre giriş alanlarını seçer.
:radio <input type='radio' olan tüm radio seçim kutularını seçer.
:checkbox <input type='checkbox' olan tüm işaretleme seçim kutularını seçer.
:submit <input type='submit' olan tüm form gönderme butonlarını seçer.
:image <input type='image' olan tüm form resimlerini seçer.
:reset <input type='reset' olan tüm form reset butonlarını seçer. //genelde 1 adet olur
:button <input type='button' olan tüm butonları seçer.
:hidden <input type='hidden' olan tüm gizli form

elemanlarını seçer.

burada hemen küçük bir hatırlatma, yazdığımı gözden geçirince gerek anlatım gerekse üslup olarak sanki <form></form> tagları arasındaki nesneleri seçer gibi bir izlenim oluşturmuşum. bu form eleman sorgularını kullandığınızda seçilmesini istediğiniz elemanların <form></form> etiketleri arasında olmasına **gerek yoktur**.

ve son olarak

*** jQuery Özel Sorguları (yardımcı sorgular) :**

jQuery diğer sorgular ile erişemediğiniz nesnelere kolay yoldan erişmek için yine kendisi çözümler sunmaktadır. bunlardan kısaca bahsetmek gerekirse

:even ve **:odd** sorguları (tek mi çift mi [*yes mi no yes mi no*]) bu sorgular kendi başlarına kullanılmamakla birlikte yanlarında kullanıldıkları nesnenin yukarıdan aşağı (kod yazımına göre) sıra numarası tek veya çift sayı olanları seçer,

```
<span>sıfır</span>
<span>bir</span>
<span>iki</span>
<span>üç</span>
<span>dört</span>
<span>beş</span>
<span>altı</span>
<span>yedi</span>
```

html kodları referansı ile

```
$("span:even") //sıfır,iki,dört,altı değerindeki span nesnelerini seçer.
$("span:odd") //bir,üç,beş,yedi isimli nesneleri seçer.
```

hemen bir hatırlatma: span değerleri içerisinde yazdığım rakam isimleri sadece bilgilendirme amacı ile yazılmıştır,jQuery sıra seçerken o rakamlara dikkat etmemektedir (karışıklık olmasın diye okunuşlarını yazdım).

biz yukarıdaki html kodu içerisinde sadece beşinci span nesnesini seçmek istersek o zaman devreye **:eq( )** sorgusu geliyor bu sorguda parantez içerisine yazacağınız rakamdaki sırayı seçer,
yani yukarıdaki örneğe göre beşinci sıradakini seçmek için <kod>$("span:eq(5)")</kod> yazmamız yeterli bu sorgunun ardından bize değeri dört olan span nesnesini verir. Seçim yaptığımız sayı beş ama aldığımız değer dört, bunun nedeni sayma işlemini sıfırdan başlatır bu yüzden dönecek olan nesne hep (n-1) formülüne göre belirlenir.

yine yukarıdaki örnek üzerinde sıra numarası 2'den önceki ve sonraki nesneleri ayrı ayrı almak istersek imadımıza **:gt( )** ve **:lt( )** yardımcı sorguları yetişiyor.
sorguğumuz soruya cevap verecek olursak,

```
$("span:gt(2)") // iki üç dört beş altı yedi ((saymaya sıfırdan başladı))
$("span:lt(2)") // sıfır bir
```

aynı örneğin ilk veya son nesnesini seçmek için **:first** veya **:last** yan sorgularını kullanabiliriz,

```
$("span:first") // sıfır
$("span:last") // yedi
```

bunların dışında gözümüzün gördüğü ve görmediği etiketleride kendi arasında seçmek mümkün bunun için kullandığımız yan sorgular **:visible** ve **:hidden**. Tahmin edebileceğiniz gibi **:visible** yan sorgusu, gördüğünüz yani css değeri *display:none* yada *visibility:hidden* olmayan tüm nesneleri seçer, aynı şekilde gözünüzün görmediği ve css değeri *display:none* yada *visibility:hidden* olan tüm nesneleride **:hidden** yan sorgusu seçer.

tüm bu yan sorgularla birlikte içinde belirttiğimiz bir değeri barındıran nesneleri seçmek için ise **:contains(' ')** yan sorgusunu kullanabiliriz.

```
<kod>
$("p:contains('php dergisi')") //içerisinde php dergisi kelimeleri geçen tüm p (paragraf) nesnelerini seçer.
</kod>
```

seçimler ve sorgular kısmını tanışma tadında bırakıyorum (muhtemelen bir sonraki yazım seçicilerin tümünü örneklerle anlatan bir yazı olacak)

sevdik, sorguladık, seçtik ,ettik peki nasıl kullanacağız ?

Bunca satırdır jQueryden bahsediyorum ama hala sayfa içerisinde nasıl kullanılacağı konusuna değinmedim, hemen ondanda bahsedelim,

defalarca dile getirdiğim üzre, jQuery bir javaScript kütüphanesidir ve sayfaya script etiketleri ile eklenir

head etiketleri ile jQuery Core dosyasını eklediğinizi varsayarak anlatıyorum,

sayfanın herhangi bir yerine (ki head etiketleri arasına tavsiye edilir)

```
<script type='text/javascript'>
  $(document).ready(function(){
    //Kodlar

  });
</script>
```

şeklinde eklenir, bir çok ekleme şekli olsada bu yöntemin ağırlığı fazladır.

Programlamaya Giriş ve Örnekler

Sinan İŞLER

Siz fıkraya gülerken biz klavyeye dökülen suya gülebiliyoruz.

Merhaba arkadaşlar.

Acemiler için başlangıç klavuzu sayılabilecek bu bölümü ben üstlendim ve her ay kısmetse yazmaya devam edeceğim. PHP'ye yeni başlayanlar veya bildiklerini pekiştirmek isteyenler için olabildiğince eğlenceli bir bölüm gibi yazmaya çalışacağım. Tabi bu eğlence yazacağımız scriptlerin bizde bıraktığı akıl kadar olacaktır. Bin satır kod yazdıktan sonra bir geliştiricinin espri anlayışı değişiyor.

Siz fıkraya gülerken biz klavyeye dökülen suya gülebiliyoruz.

Ayrıca bu bölümü yazma nedenlerinden birisi de benim sizler gibi acemi olmam PHP ile tanışıklığım çok eski değil o yüzden öğretirken öğreneceğim.

Temel ögelerden başlayalım.

Temel programlama mantığı hayatın içinde yaşanmaktadır.

Eğer daha önce bir programlama dilini öğrenmediyseniz, script yazmadıysanız, programlama okumadıysanız olayın mantığını o ufak nüansı yakalamanız zor oluyor. Kendimden biliyorum dedim ya bende acemiyim diye benim açımdan zor olmuştu kolaylaştırmak için bol bol örnek çözmek ve problemleri kendinize göre uyarlamanız gerekmekte.

PHP ye geçmeden önce genel bir yapıyı örneklemeye çalışacağım.

Programlamaya emirler silsilesi diyebiliriz. Bilgisayar yorulmaz bir köledir ve biz ona sürekli emirler veririz. Nasıl mı?

```php
<?php
$ahmetinparasi  = 10; //Ahmet'in 10 TL'si var
$cevdetinparasi = 20; //Cevdet'in 20 TL'si var

/*
Anlayacağınız Ahmet ve Cevdet çok fakir fazla paraları yok fakat her ikiside sinemaya gitmek istiyor. Sinema bileti 15 TL evet yukarıdaki değerlere (diğer adıyla: değişkenlere) bakıyoruz. Ahmet'in sinemaya gitmesi imkansız çünkü onun 10 TL'si var fakat cevdet ile birlikte paralarını toplarlarsa sinemaya girebilirler. O zaman toplayalım.
*/

$ortakpara = $ahmetinparasi + $cevdetinparasi;
/*Yukarıda değişkenleri topladık ve eğer kavradıysanız değişkenlerin yapısını az ğrendiniz demektir. Evet her değişken bir veri taşır. Burada değişkenlere Ahmet ve Cevdet'in paralarını taşıtıyoruz. Toplama işlemiylede paraların toplam miktarını $ortakpara değişkenine atıyoruz.
Eğer $ortakpara değişkenini ekrana yazdırırsak karşımıza 30 rakamı çıkacaktır.
Echo ekrana değişkeni basmak için kullanılan komutlardan birisidir kullanalım.
*/

echo $ortakpara;
//hepsi bu yukarıdaki komut bize 30 sonucunu verecektir.

/* şimdi programımızı devam ettirelim ben bu örneği sevdim Ahmet ile Cevdet'i sinemaya yollayalım ve bilet satın alsınlar.
aşağıdaki if else yapısı en kısa haliyle eğer doğruysa, eğer yanlışsa anlamına gelir. İleride bu konuları ayrıntılı işleyeceğiz fakat konunun özünü kavramaya çalışın.
şimdi bir kontrol yapısı oluşturalım */

$BiletFiyati = 15;
$ikiKisilikBiletFiyati = 30;

if (ikiKisilikBiletFiyati==$ortakpara )
{
echo 'Bilet fiyatını ödeyebildiniz buyrun biletiniz iyi seyirler';
} else{
//peki paraları yetmeseydi? işte o zaman aşağıdaki satır çalışırdı.
echo 'Paranız yetmedi bedava film izletemeyiz. Güvenlik! alın şu arkadaşları.';
}
?>
```

Yukarıda yazdığımız onca satır php'nin temellerini ve programlamanın mantığını oluşturuyor. Bilgisayar dillerinin temel mantığı çoğunlukla aynıdır. Bir yapının yapay zekası olmadığı sürece vereceği karar 0 yada 1 dir. Yani 'if' yada 'else'.

Yukarıda yazdığım örnekler sinemaya giriş yerine bir formun dolu olup olmadığını kontrol eden bir yapı olabilirdi, bir mail üyelik sistemi olabilirdi ve mail girişinin doğruluğunu kontrol edebilirdi, bir yazı kaydedici olabilirdi ve yazı kaydı gerçekleşirse sonucu bildirirdi, Bir hesap makinesi olabilirdi ve toplama işlemi yaptırabilirdiniz.

Bir programlama dili ile yapabilecekleriniz hayal gücünüzle sınırlıdır.

Biri PHP dilini yapmış yıllar sonra bu dili kullanarak facebook yapılmış. Kim bilir belki sizde bir facebook yazabilirsiniz. PHP size çekiç, çivi, testere, beton verir bunları birleştirip bina yapmak sizin işiniz.

Bu haftaki yazımızda sadece genel mantığı ele aldık ileriki yazılarımızda PHP konusuna tam giriş yapıp terimleri ve yapıyı öğrenmeye başlayacağız.

Programcının yoğurt yiyişi: "Algoritma"

Bahriye SARIKAYA

Evde oturuyorsunuz. Birden bire canınız makarna istedi ve pişirmeye karar verdiniz. Önce evde makarna olup olmadığına baktınız. Eğer yoksa, markete gidip bir paket makarna alacağınızı düşündünüz. Ah evde makarna varmış! O halde, bir sonraki adıma geçerek tencereye suyu koydunuz. Ocağı yaktınız ve suyun kaynamasını beklemeye başladınız. Su kaynadı mı? Henüz değil. O halde, biraz daha beklediniz. Su kaynayınca makarnayı attınız tencereye, tuzunu ekleyip, pişmesini beklemeye başladınız. Makarna pişti mi? Hmm, henüz değil. O halde biraz daha beklediniz. Makarna pişince, süzdünüz. Yağını ekleyince, makarnanız artık afiyetle yemeye hazır hale geldi.

Konumuz makarna pişirmek mi? Elbette değil. Konumuz algoritma. Algoritma, bir problemi çözmek için uygulanacak, başlangıçtan sona kadar detayları belirlenmiş adımlar ve izlenecek yoldur.Evet, doğru! Biraz önceki örnek durum, makarna pişirme algoritmasıydı. Sizi sıkmamak için çok detaylandırmadım, ama tenceremiz var mı, ocağımızın gazı yetecek mi gibi soruları da bu algoritmaya ekleyebilirdik.

Algoritma kelimesi, cebir – matematik, coğrafya ve astronomi konularında dünyanın ilk eserlerini yazmış, İranlı bilim adamı El-Harezmi'nin adından gelir. El-Harezmi'nin "Hint-Arap dillerinde hesaplama" adındaki eseri, 12. yüzyılda latinceye "Algoritmi de numero Indorum" olarak çevrildi. Çevirmenin, adının muhtemel olarak "Algoritmi'nin Hint numaraları çalışmaları" anlamına gelmesini beklediği bu eser, okuyucular tarafından "Algoritmi, Hint numaraları çalışmaları" olarak anlaşılınca, metodun adı "Algoritmi" olarak kaldı. Böylece, "algoritma" kelimesi latinceye, "hesaplama metodu" manasıyla girmiş oldu.

Tarihte bilinen ilk algoritmaları milattan önce 1600'lerde Babillilerin geliştirdiği düşünülüyor. Bu algoritmalar, çarpanlarına ayırma ve kök bulma algoritmaları. Bunu milattan önce 300'lü yıllarda Öklid algoritması izliyor.

Günümüzde ise algoritma için bilgisayarların bilgiyi temel işleme yöntemi olduğunu söyleyebiliriz. Bilgisayarlar, net olarak ne dersek onu yapan ve ne demezsek onu "yapmayan" aygıtlar olduğundan bizim, adım adım, ne yapması gerektiğini detaylı olarak bilgisayara söylememiz gerekir. Bir bilgisayarın iki sayıyı toplaması için şöyle bir algoritma izlerdik:

Başla
İlk sayıyı kullanıcıdan al
İkinci sayıyı kullanıcıdan al.
İki sayıyı topla.
Ekrana yazdır.
Dur.

Algoritma, herhangi bir programlama dilinden bağımsız, sözel olarak yazılır. Aynı zamanda akış diyagramı adı verilen, her bir şeklin anlamı olan ve ok işaretleriyle yönü belirtilen bir diyagram sistemi ile de çizilebilir.

Peki algoritma biz programcılar için neden önemli? Eğer işimiz, iki sayıyı toplamak değilse, daha karmaşık bir sistem inşaa etmek ise, yolumuzu her an kaybedebilme riskimiz var. Problemi çözmek üzere yazdığımız kodun adımları başlangıçta net değilse, kodu yazmaktan çok, hata ayıklamak ile ve sistemi pek de belli olmayan kodu yeniden ve yeniden düzenlemekle vakit harcarız. Bir algoritma üzerinden programlama yapmak, ekip çalışması içindeyseniz, tüm ekibin bütünlüğünü sağlayacaktır. Çözüme ulaşmak için kullanılacak yollar açık ve detaylı olarak belirlendiğinde, büyük sürprizlere yer kalmayacaktır.

Bilgi teknolojilerinde kullanılan temel algoritmalar, arama algortimaları, sıralama algoritmaları, birleştirme algoritmaları, çizim algoritmaları ve karakter dizileri üzerine algoritmalar olarak sıralanabilirler. Örneğin, sıralama algoritmaları arasında en popüler olanları, baloncuk sıralaması (bubble sort), birleştirme sıralaması (merge sort) ve ekleyerek sıralamadır (insertion sort). Sıralama algoritmaları, bir dizi içindeki elemanları belirli bir şarta göre sıralamak için kullanılırlar. Aynı şekilde, Bilgisayarların varlığının en büyük sebebi, veriyi tutmak, işlemek ve ihtiyaç duyulduğunda kolayca erişmek ihtiyacı olduğunu düşünürsek, arama algoritmaları da neredeyse bilgi teknolojilerinin temel taşıdır. Arama algoritması denince akla gelen algoritmalara doğrusal arama (linear search) ve ikili aramayı (binary search) örnek verebiliriz.

Ünlü algoritmaların dışında, sizin spesifik bir problem için hazırladığınız algoritma, size özeldir. Bir bilgisayar programı onlarca farklı algoritma ile yazılabilir. Bir programlamacının ne kadar işinde yetenekli olduğunu yazdığı programın algoritmasından anlayabilirsiniz. Aynı iş için, bir diğerinden daha dolaylı yazılmış program, sistem kaynaklarını gereksiz yere harcayabilir. Algoritmanızı, hedefe ulaşacak en doğru ve en kısa yol ile oluşturmalısınız. Örneğin sıralama algoritmalarının birden fazla olmasının sebebi budur. Milisaniyelerle ölçülse dahi, sıralama algoritmalarının her birinin sonuçlanma hızları farklıdır. İşte bu yüzden "algoritma becerisi", bir programcının "yoğurt yiyişidir".

Her yerde işittiğimiz şehir efsanesi ile bitirmek istiyorum. En az 5 yıl bilgisayar programcılığı yapanların, Amerika'da şahitliğinin kabul edilmediği söylenir. Bunun temel sebebi, programcıların artık algoritmik düşündüğü ve mahkemeyi farkettirmeden yönlendirebilecekleri düşüncesidir.

Resimin Türkçe tercümesi:

"Kadın, programcı olan eşinden markete gitmesini rica eder:

-Hayatım, lütfen yakındaki markete gidip ekmek al. Eğer markette yumurta varsa, 6 tane al.

-Tamam tatlım.

Yirmi dakika sonra adam eve 6 somun ekmekle döner. Kadın afallamıştır:

-Hayatım, söyler misin, neden 6 somun ekmek aldın?

-Çünkü,markette yumurta vardı"

Sistem Nedir ?

Ali GÜNDOĞDU

Sistem belli yada belirsiz bir yöndeki ihtiyacı gidermek üzere bir arada çalışan ve birbirleri ile sürekli etkileşim halinde bulunan bireylerden oluşan, giriş ve çıkışları olan alt ve üst sınırları kesin şekilde belirlenmiş bir bütündür.Bu anlatılanlara göre sistem bireylerden oluşmakta ve bu bireyler arasında ilişki bulunmakra ve bir amaca hizmet etmektedir.

Sistem ile alakalı 3 temel kavram vardır:

Birey : Sistemin en küçük elemanıdır. Birey bir makinanın bir dişlisi yada bir uçağın motoru gibi fiziksel bir öğe olabileceği gibi, bir fabrikanın pazarlama departmanı gibi kavramsal bir düşüncede olabilir. Her bireyin görevi kesin ve net çizgilerle belirlenmelidir.

İlişki : Sistem içerisinde bireyler arasında her tür ve yöndeki bilgi alış verişi ile akış ilişkisi olarak isimlendirilir. Bireyler arasındaki ilişki ise zamanlanmış, neden sonuç mantığında, enerji düzeyinde, mantıksal yada matematiksel, fiziksel olabilir.

Hedef: Teknolojinin oluşturduğu her nesne belli yada belirsiz bir ihtiyacın telebini hedef alarak doğmuştur.Temelde hedef bu gereksinimleri ortadan kaldıracak sistemi oluşturmaktır.Hedeflere ulaşmak,insan ve otomasyon sistemleri oluşturmak için harcanan enerji ve kaynağın, ekonomik yönden doğrulanmasından geçer.

Özellikler

Sistem özellikleri : bireyler, değişkenler, sabit değerler,ilişkiler, aktarma birimleri, ortam,sınır, yetkiler ve kıyaslamalar olarak belirlenebilir.

Bireyler : yukarıdada bahsedildiği gibi, sistemi oluşturan parçaların genel adıdır.
Değişkenler : Sistemin veri trafiğini sırtlanan özelliklerindendir, her duruma göre farklı içeriğe sahip olabilirler.
Sabit Değerler : Tasarım ve İşleyiş sürecinin olmazsa olmazısıdırlar, asla değişmezler ve sabittirler.
İlişkiler: Şimdiye kadar saydığım özelliklerin arasında bağlayıcı görevi görürler, İlişkiler her bir olayı kontrol eder, denetler, her duruma ve sabit değerlere göre yeni değerleri değişkenlere aktarırlar.

Aktarma Birimleri : Sistemin bireylerinin birbirleri ile karşılaştığı alandır. Aktarma birimlerinin vazifeleri genelde, gizlilik, bireylere gerektiği kadar bilgi verme,gelen giden girdi çıktıları şifreleme veya şifreçözme, Hata kontrolü, Adressiz verileri en mantıklı adreslere teslim etmektir.

Yetkiler : değişkenlerin deperleri veya kaynakların nasıl tahsis edileceği üzerindeki sınırlamalardır. Bu kısıtlar, tasarımcı tarafından koyulacağı gibi, sistemin doğasından da olabilir. Tasarımcı tarafından koyulan kısıtlar üzerinde oynanabilmesi, bunların sıkılaştırılıp gevşetilmesi mümkündür.

Kıyaslamalar : Yargılama standardı olarak tanımlanabilir. Kıyaslama fonksiyonu ise sistemin hedeflerinin veya amaçlarının ve bunların nasıl değerlendirileceğinin bir durumudur. Buna göre kıyaslama fonksiyonunun iki büyük önemi vardır. Birincisi, modelin tasarımı ve işletilmesi üzerinde büyük etkisi vardır.İkincisi, kıyaslamaların yanlış tanımlanması yalnış sonuçlar doğurur.

Sınır : sistemin oluşturduğu ortamı belirler. Bir sistemin yeteneği sınırına bağlıdır.

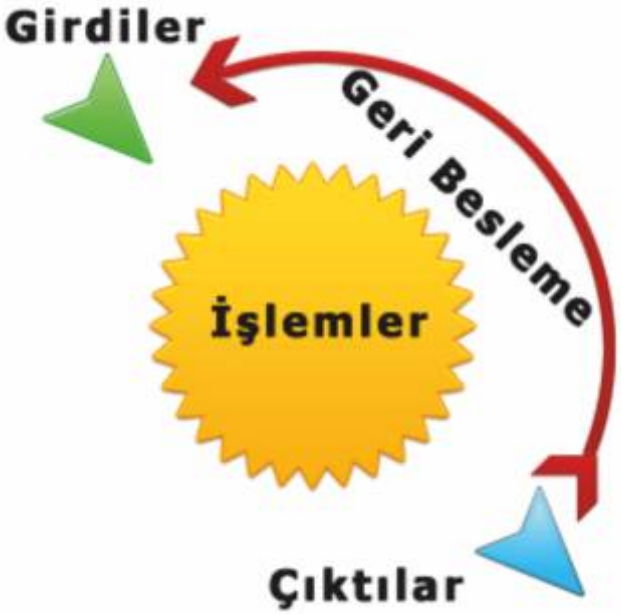
Arabirim : Bireylerin birbirleri ile karşılaştığı alandır, her bir bireyin birbirine geçişi sırasındaki bütünlüğü ve kolaylığı sağlar. Arabirimin belli başlı özellikleri vardır.bunlar ;
Gizlilik, konuya dahil olmaması gereken bireylere karşı içeriği gizleme görevi taşır,
Filtre , yine aynı şekilde bireyler arası bilgi alışverişinin sadece istenen bilgi düzeyince olmasını sağlar.
Şifreleme, bilgi akışı sırasında bilgilerin şifrelenmesi ve bu şifrenin çözülmesi görevini üstlenir.
Hata ayıklama, Veri sağlamasını yaparak olası hataları önceden tahmin etmeyi görev bilir.

Kısıtlama : Değişken bireylerin alabileceği değerler ile "sistem" kaynaklarının nasıl ve ne şekilde düzenlenip tahsis edileceğini kapsayan kurallar bütünüdür.
Bu kısıtlamalar sistem tasarımcısı tarafından oluşturulabildiği gibi yine sistem tasarımcısı tarafından sistemin işleyişine göre değişkenlik gösterebilecek şekilde ucu açık bırakılabilinir. bu sayede tüm kısıtlama kuralları dinamik olarak değişkenlik gösterebilir.Bu kısıtlamalar içerisinde sadece tasarımcı tarafından yerleştirilen kısıtlamar rahatça oynanabilinmekte, ucu açık ve dinamik kısıtlamaların takibi ve sağlamasında ileri zamanlarda sorunlar çıkabilmektedir.

Kısıtlar: Kontrol ve takip kurallarının tümüne verilebilecek kümenin adıdır. bu küme, sistemin başı ile sonunda ulaşılması gereken hedeflerin kontrolünü üstlenir. En önemli durum kısasların değerlendirilmesinde ortaya çıkar. Çünkü sistemin bu kısaslar ile amacına ulaşıp ulaşamadığı tespit edilebilecektir, bu sebepten ötürü kısaslar belirlenirken çok iyi düşünülmeli ve belirlenmelidir.

Platform (ortam): Sistem bireyleri dışında kalan ve tanımlanmamış herşeye verilen genel isimdir.

Temel bir sistem mimarisi şu şekildedir,



Şekil 1 : Sistem girdi - çıktı ve geri besleme şeması.

Zend Framework Helper kullanımı Örnek: Minfy

Volkan Altan

Helper Kullanım Amacı;

Projelerde bazen sürekli kullanmak zorunda olduğumuz metodlar vardır. Bu metodları bir yere toplayıp ordan kullanmak ilerde bakımını yapmamızı da kolaylaştıracaktır.

Mesela tarih gösterimi için bir formatımız olsun gün/ay/yıl şeklinde. Müşteri 1 ay sonra bunu değiştirmemizi istediğinde kodların içine girip her tarih gördüğümüz yeri değiştirmek

pek mantıklı olmayacaktır :) bunu yerine bir metod yazıp kendisine "içine gelen zaman damgasını bizim istediğimiz bir formata çevir" özelliğini verirsek sadece ilgili metodda güncelleme yaparak

koskoca bir işten kurtulmuş oluruz.

Tabi bunları yapmak için sırasıyla takip etmemiz gereken işler var. önce bir php dosyası oluşturup içine kodları yazacağız sonra bu dosyayı ilgili yerde include etmek zorunda kalacağız.

Bunu zend Framework ile yapmak istersek;

Kullanmak istediğimiz modülün içerisinde views/helpers içerisine metod adımızla bir dosya oluşturup içine Zend_View_Helper_DosyaAdi ile başlayan bir sınıf oluşturmalıyız.

Bu sınıf içerisine public olarak metodumuzu oluşturabiliriz.

Dosya adı: TarihDonusturucu.php

```
<?php

class Zend_View_Helper_TarihDonusturucu
{

/**
* @var Zend_View_Interface
*/
public $view;

public function tarihDonusturucu($zamanDamgasi) {
   return data('d/m/Y');
}

public function setView(Zend_View_Interface $view) {
$this->view = $view;
}

}

?>
```

Daha sonra bu metodu view içerisinde kullanmak istediğimiz de yapmamız gereken sadece "echo $this->tarihDonusturucu($zamanDamgasi);" demek.
Olurda controller de lazım olursa "$this->view->tarihDonusturucu($zamanDamgasi);" şeklinde kullanabiliriz.

Gördüğünüz gibi oldukça basit. Şimdi biraz daha gelişmiş bir şekilde helper i kullanalım.

Sorun: kullandığımız javascript ve css dosyalarını sıkıştırıp servere atmak ve sonra geri alıp düzenlemek istediğimizde eski haline getirmenin oldukça uzun bir iş olması.
Çözüm: Sunucuya script ve css lerin normal halini atıp çalışma anında ziplenip belleğe alınmasını sağlamak.

Bir sorunla karşılaştığımda önce bunun açık kaynak (open source) çözümü olup olmadığına bakarım. Her şeyi yeniden yazıp dünyayı yeniden keşfetmeye gerek yok. Zaten yazılmış ve verimli çalışan
uygulamalar varsa onları kullanmak bazen işlerinizi çok daha kolay yapabilir.

Burda araçtırma yaptığımızda sorunun çözümü için minify isimli uygulamanın sorunlarımızı çöceğini anlıyoruz. (http://code.google.com/p/minify/)

Şimdi bunu ZF ile yazılmış uygulamamız gerek.

ZF nin layout kısmında headScript, headLink ismindeki helper yardımıyla script ve css dosyalarını projeye ekleyebiliyoruz. Eklenen scriptleri görmek içinse
helper çağrılır.

```
<?php
echo $this->headLink ();
echo $this->headScript ();
?>
```

Biz burda ekleme kısmını hiç ellemeden sadece çağrılma helperini değiştirerek minify ı uygulamamıza ekleyeceğiz.

İşe başlarken bu helperin kullanım alanını düşünmek gerekiyor. Bütün modüllerde kullanacağım için kendisini bir modülün altında tutmak yerine library altında bir yere koyup bütün modüllerin hizmetine
sunmanın mantıklı olacağını düşünüyorum.

public altına minify isimli bir klasör oluşturup içerisine minify dosyalarını atıyorum.

libraryCMSAddonViewHelper altına MinifyScript.php ve MinifyHeadLink.php isminde iki adet dosya oluşturuyorum.

MinifyScript.php için;

```
<?php

class CMS_Addon_View_Helper_MinifyScript extends
Zend_View_Helper_HeadScript {
/**
* @var Zend_View_Interface
*/
public $view;
   public function minifyScript()
   {
    $scripts = array();

    foreach ($this as $item)
    {
    if ($item->type == 'text/javascript' &&
    !empty($item->attributes) &&
```

```
    !empty($item->attributes['src']))
    {
    $scripts[] = $item->attributes['src'];
    }
    }

    $item = new stdClass();
    $item->type = 'text/javascript';
    $item->attributes = array();
    $item->attributes['src'] = $this->getMinUrl() . '?f=' . implode(',',
$scripts);

    return $this->itemToString($item, null, null, null);
   }

private function getMinUrl() {
return $this->getBaseUrl() . '/minify/';
}
private function getBaseUrl() {
return Zend_Controller_Front::getInstance()->getBaseUrl();
}


/**
* Sets the view field
* @param $view Zend_View_Interface
*/
public function setView(Zend_View_Interface $view) {
$this->view = $view;
}

}
```

MinifyHeadLink.php için;

```
<?php

class CMS_Addon_View_Helper_MinifyHeadLink extends
Zend_View_Helper_HeadLink {
/**
* @var Zend_View_Interface
*/
public $view;

 public function minifyHeadLink()
 {
  $styles = array();
  foreach ($this as $item)
  {
  if ($item->type == 'text/css' &&
  !empty($item->media) &&
  !empty($item->href))
  {
  $styles[] = $item->href;
  }
  }

  $item = new stdClass();
  $item->type = 'text/css';
  $item->rel = 'stylesheet';
  $item->media = 'screen';
  $item->href = $this->getMinUrl() . '?f=' . implode(',', $styles);

  return $this->itemToString($item, null, null, null);
 }

private function getMinUrl() {
return $this->getBaseUrl() . '/minify/';
}
private function getBaseUrl() {
return Zend_Controller_Front::getInstance()->getBaseUrl();
}


/**
* Sets the view field
* @param $view Zend_View_Interface
*/
public function setView(Zend_View_Interface $view) {
$this->view = $view;
}

}
```

helper lar kullanıma hazır. layout da kullanmak için ;

```
<?php
echo $this->minifyScript();
echo $this->minifyHeadLink();
//echo $this->headLink ();
//echo $this->headScript ();
?>
```

Sizde her yerde kullanacağınız helper i library altında tutarak çalıştırabilirsiniz.

Firefox eklentisi olan "google page speed" ile sitemizin performansındaki artışı gözlemleyebiliriz.

PHP Kodlama Standartları

Cankat Akdemir

İster evinizde kendiniz için küçük programlar yazıyor olun, ister büyük bir grup ile büyük bir uygulama geliştiriyor olun PHP kodu yazarken dikkat etmeniz gereken konulardan biri her zaman *kod yazım adetleri*dir. Kimse size bunları zorlamaz. Kodlarınızı bu şekilde yazmadığınız için dayak yemezsiniz ama sizin kodunuzu okumak ve yorumlamak zorunda olan iş arkadaşlarınızı kızdıracağınız kesin.
Belli kodlama standartlarına uymak hem çalıştığınız her projeyi kapsayan bir tutarlılık sağlar. Ayrıca dünyanın herhangi bir yerindeki biri tarafından yazılmış PHP kodu size tanıdık geleceği gibi sizin yazdığınız kod da her yerde rahatlıkla yorumlanabilir.
Bu standartlar yıllar boyu geliştirilen seziler ve en rahat edilen yöntemler seçilerek belirlenmiştir. Kodlama standartları her türlü geliştirme ortamı için önemli olsa da en önemli olduğu durum şüphesiz ki birden çok geliştiricinin ortak çalıştığı projelerdir. Kodlama standartları kodun yüksek kaliteli, az hatalı ve kolay yönetilebilir olmasını sağlar.
Aşağıdaki standartlar Zend'in Zend Framework için belirlediği standartlardır. Orijinal açıklamaları http://denemec.com sayfasından okuyabilirsiniz.

PHP Dosyalarının Biçimlendirmesi

Genel

İçerisinde sadece PHP kodu barındıran dosyalar için kapatma etiketini kullanmanız yasaklanmış değildir. Fakat ?> etiketi bu PHP tarafından yorumlanmaz. Yani, gereksizdir. Ayrıca bu etiketi yazmayarak yanlışlıkla dosyanın sonuna ekleyeceğiniz fazladan boşluk karakterlerini de önlemiş olursunuz.

Girintili Satırlar

Girintili yazmanız gereken satırlar için girinti 4 boşluk karakterinden oluşur. Boşluk yerine TAB kullanmamalısınız.

Satır Uzunluğu

Okunabilirlik açısından bir satır en fazla 80 karakter barındırmalıdır. Bazı durumlarda daha fazla karakter barındırılması sorun yaratmaz. PHP içinse en uzun satır 120 karakterden oluşur.

Satır Sonu

Satır sonu için Unix metin dosyalarının standartları uygulanır. Satırlar tek bir *linefeed (LF)* karakteri ile sonlandırılmalıdır. Linefeed karakteri sıralı değerde 10, altılı değerde ise 0x0A ile ifade edilir.
Apple'a ait işletim sistemlerinde kullanılan *carriagereturn (CR)* ya da Windows'un kullandığı *carriagereturn – linefeed* birleşimi (*CRLF*) karakteri kullanmamalısınız.

İsimlendirme Adetleri

Sınıfları, fonksiyonları ve değişkenleri isimlendirirken eski köye yeni adet getirmeyin ve belirlenmiş adetleri kullanın. İşte onlar.

Sınıflar

Son yıllarda sınıf isimlerinin o sınıfın diskteki yolunu işaret ettiği ve *Pear Style* olarak adlandırılan isimlendirme yöntemi hemen her projede kullanılıyor. Zend Framework de bunlardan biri. Örneğin, Zend_Db_Table sınıfı Zend/Db/Table.php dosyasına işaret eder. Bu isimlendirme proje dosyalarınızın yönetimini de oldukça kolaylaştırır.
Sınıf isimlerinde sadece alfa numerik karakterler kullanılabilir. Alt çizgi ise sadece yukarıdaki örnekte gördüğünüz gibi yol ayracı olarak kullanılabilir.
Sınıfı adı birden çok sözcük barındırıyorsa bu sözcüklerden her birinin ilk harfi büyük harf olmalıdır. Bu isimlendirme şekline *Macar Stili* denir. Tümü büyük harf olan sınıf isimleri kabul edilmez. Zend_PDF kabul edilmezken Zend_Pdf uygundur.
Bu isimlendirme şekli uygulamanıza bir nevi *namespace* desteği de sağlamış olur. PHP 5.3 ile gelen namespace desteğinden tümüyle faydalanana kadar uygulamanızdaki kod yapısını derli toplu tutmak için bu isimlendirme yöntemini kullanabilirsiniz.

Abstract Sınıflar ve Arabirimler

Abstract sınıflar da diğer sınıflarla aynı isimlendirme yöntemlerini kullanır. Tek fark, bu sınıflar Abstract terimiyle biterler fakat bu isimden önce alt çizgi karakteri gelmez. Yani, Zend_Controller_Plugin_Abstract geçersiz iken Zend_Controller_Plugin_PluginAbstract geçerli bir sınıf ismidir.
Arabirimler (*interfaces*) de abstract sınıflarla aynı kurala tabidir. Sadece Abstract yerine Interface terimi gelir.

Dosya Adları

Dosya adlarında sadece alfa numerik karakterler, alt çizgi ve tire (-) karakterlerine izin verilir. Boşluk karakteri ise kesinlikle kullanılmamalıdır. PHP kodu barındıran dosyalar kesinlikle PHP uzantısı ile bitmelidir.

Fonksiyonlar ve Metotlar

Fonksiyon isimleri sadece alfa numerik karakterler barındırabilir. Alt çizgi kullanılmamalıdır. Rakamlara izin verilir fakat genelde kullanılmaz. Fonksiyon isimleri her zaman küçük harfle başlar. Eğer isim birden çok sözcük barındırıyorsa, yukarıda bahsettiğim Macar Stili ile biçimlendirilir. Her sözcüğün baş harfi büyük yazılır.
Konu fonksiyon isimleri olunca birçok tartışmaya internette çeşitli yerlerde rastlayabilirsiniz. Bu konudaki en geçerli yöntem *gereksizlik ilkesi* olarak adlandırılır. Bir fonksiyon ismi gereksizce uzun olmamalı ama amacını ve davranışı tam olarak belirtmelidir.
Aşağıdakiler kabul edilebilir fonksiyon isimlerine örnektir.

```
filterInput();
getElementById();
widgetFactory();
```

Nesne yönelimli programlamada değişkenlere erişen (*accessor*) fonksiyonlar ve değer atayan (*setter*)

fonksiyonlar get ve set başlangıcına sahip olmalıdır.
Sınıflarda yer alan *protected* ve *private* metotlar alt çizgi karakteri ile başlamalıdır. Bir metot adında alt çizginin yer aldığı tek uygulama budur. *Public* metotların isimlerinde ise asla alt çizgi yer alamaz.
Global kapsamındaki fonksiyonlar kabul edilir olsa da uygun görülmezler. Bu fonksiyonları *static* metot olarak bir sınıfta toplayabilirsiniz.

Değişkenler
Değişken isimleri sadece alfa numerik karakterler barındırabilir. Alt çizgiler yasaktır. Rakamlar kullanılabilir fakat kullanıldığına çok rastlanmaz.
Sınıflarda yer alan *protected* ve *private* değişkenler alt çizgi karakteri ile başlamalıdır. Bir değişken adında alt çizginin yer aldığı tek uygulama budur. *Public*değişkenlerin isimlerinde ise asla alt çizgi yer alamaz.
Fonksiyon isimlerinde olduğu gibi değişken isimlerinde de birden çok sözcük olduğunda Macar Stili uygulanır.
Gereksizlik İlkesi değişkenlere de uygulanır. Bir değişken barındırdığı değeri tanımlayacak kadar uzun ama gereksiz olmayacakkadar kısa isimlendirilmelidir.
Değişkenlerde sık karşılaştığımız $i ve $n gibi isimlendirmeler sadece çok kısa döngülerde kullanılmalıdır. Eğer bir döngü 20 satırdan çok kod içeriyorsa indeks değişkenleri daha tanımlayıcı isimlere sahip olmalıdır.

Sabitler
Sabitlerin isimlendirilmesinde hem alfa numerik karakterler hem de alt çizgiler kullanılabilir. Sabitlerin isimlerinde rakamlar da kullanılabilir. Sabitlerde tüm karakterler büyük harf olmalıdır. Birden çok sözcük varsa, bu sözcükler alt çizgi ile ayrılır.
EMBED_SUPPRESS_EMBED_EXCEPTION kurallara uygun bir sabit adıyken EMBED_SUPPRESSEMBEDEXCEPTION uygun değildir.
Tüm uygulama çapında kullanılacak sabitler define fonksiyonu ile tanımlanabilirken uygulanmaması tercih edilir. Sınıflarda tanımladığınız sabitler için const değiştiricisini kullanabilirsiniz.

Kodlama Stili

PHP Kodlarının Sınırlandırılması
PHP kodları her zaman tam PHP etiketleriyle sınırlandırılmalıdır. Aşağıdaki gibi:

```
<?php
?>
```

Kısa etiketler XML benzeri yapılarla çakışabileceği için asla kabul edilmez. Sadece PHP kodu içeren dosyalarda kapanış etiketini yazmanıza gerek yoktur.

Metinler

Salt Metinler
Eğer bir metinde hiçbir değişkene başvuru içermiyorsa sadece tek tırnak içerisine alınmalıdır. Örnek:
$metin = 'deneme metni';

Tek Tırnak İçeren Metinler
Bir salt metin tek tırnak karakterleri içeriyorsa metni çift tırnak ile sınırlamak tercih edilmelidir. Bu uygulama özellikle SQL deyimleri için uygundur:

```
$sql = "SELECT `id`, `name` from `people` "
   . "WHERE `name`='Fred' OR `name`='Susan'";
```

Bu yöntem okunabilirliği artırmak için tek tırnak işaretlerini *escape* etmek yerine tercih edilebilir.

Değişken Konumlandırma
Metinlerde kullanılacak değişkenler aşağıdaki iki şekilde kullanılabilir:

```
$selam = "Merhaba $isim, hoş geldiniz!";
$selam = "Merhaba {$isim}, hoş geldiniz!";
```

Tutarlılık için aşağıdaki yöntem ise kabul edilmez:

```
$selam = "Merhaba ${isim}, hoş geldiniz!";
```

Metin Birleştirme
Metinler "." (*nokta*) operatörü kullanılarak birleştirilmelidir. Operatörden önce ve sonra bir boşluk karakteri okunabilirliği artırmak için eklenmelidir.

```
$company = 'Zend' . ' ' . 'Technologies';
```

Nokta kullanarak metin birleştirirken okunabilirliği artırmak için metni birden çok satıra bölebilirsiniz. Bu gibi durumlarda, her satır nokta karakteri eşittir karakterinin altına gelecek şekilde boşluk kullanılarak girintili yazılmalıdır.

```
$sql = "SELECT `id`, `name` FROM `people` "
   . "WHERE `name` = 'Susan' "
   . "ORDER BY `name` ASC ";
```

Diziler (*Array*)

Nümerik İndeksli Diziler
Dizilerde indis olarak negatif sayılar kullanılamaz.
Bir indis negatif olmayan herhangi bir sayı ile başlayabilir fakat 0 (*sıfır*) dışındakiler önerilmez.
Array fonksiyonu kullanarak indeksli bir dizi oluşturuyorsanız, her virgülden sonra bir boşluk bırakarak okunabilirliği artırabilirsiniz:

```
$ornekArray = array(1, 2, 3, 'Zend', 'Studio');
```

Array fonksiyonunu dizinin elemanlarını birden çok satıra yazarak da kullanabilirsiniz. Bu durumda her satır hizalanacak şekilde girinti kullanmalısınız.

```
$ornekArray = array(1, 2, 3, 'Zend', 'Studio',
                    $a, $b, $c,
                    56.44, $d, 500);
```

Bu şekilde biçimlendir yaparken her satırın sonuna virgül koyduğunuzdan emin olun. Virgüller hem o satıra yeni dizi

değerleri eklemenizi kolaylaştırır hem de PHP çalışırken virgül olmamasından kaynaklanan işleme hatalarını engeller.

Associative Diziler

Array fonksiyonu ile associative diziler tanımlarken her anahtar ve içerik çift ayrı satıra gelmeli ve anahtarlar alt alta olacak şekilde girintili yazılmalıdır.

```
$sampleArray = array('firstKey'  => 'firstValue',
                     'secondKey' => 'secondValue');
```

Alternatif olarak ilk dize girdisi yeni satırda başlayabilir. Bu durumda o satır, dizenin tanımlandığı satıra göre en az bir kademe girintili yazılmalıdır. Takip eden anahtar-değer satırları da aynı girinti kullanılarak yazılmalıdır. Kapatma parantezi ise yeni bir satırda ve array fonksiyonunun olduğu satırla aynı girintide olmalıdır. Okunabilirlik için => eşleştirme karakteri diğerleri ile alt alta gelecek şekilde girintili olabilir.

```
$sampleArray = array(
    'firstKey'  => 'firstValue',
    'secondKey' => 'secondValue',
);
```

Bu biçimde yazarken son dize girdisinden sonra da virgül koyduğumuzu unutmayın. Bu virgül hem dizeye yeni girdiler eklememizi kolaylaştırır hem de PHP çalışırken *eksik virgül* gibi bir hatayla karşılaşma ihtimalimizi azaltır.

Sınıflar

Sınıf Tanımlama

Bir sınıf tanımlarken açılış kırlangıç parantezi ({) sınıf adının altındaki ilk satıra yazılmalıdır.
Her sınıf PHPDocumentor standartlarına uyan bir belgelendirme bölümü barındırmalıdır.
Sınıf içerisindeki her satır dört boşluk karakteri ile girintili yazılmalıdır.
Bir PHP dosyasında bir sınıf olmalıdır.

Sınıftan ayrı olarak ek PHP kodlarının dosyada yer alması mümkündür fakat uygun görülmez. Böyle biri durumda sınıfın bitişinden sonra iki satır boşluk bırakarak ek kodları yazabilirsiniz.
Aşağıdaki örnek kabul edilebilir bir sınıf tanımlamasıdır:

```
/**
 * DocumentationBlock Here
 */
classSampleClass
{
  // allcontents of class
  // must be indentedfourspaces
}
```

Diğer sınıfları genişleten (*extend*) ya da arabirim uygulayan (*interface*) sınıflar bağımlılıklarını mümkünse aynı satırda belirtmelidir.

```
classSampleClassextendsFooAbstractimplementsBarInterface
{
}
```

Eğer bağımlılıklar *satırdaki en fazla karakter sayısı*nı geçiyorsa anahtar kelimeler yeni satırda başlayacak şekilde bölünebilir. Bu satırlar bir birim girintili olarak yazılır.

```
classSampleClass
extendsFooAbstract
implementsBarInterface
{
}
```

Eğer sınıf birden çok arabirimi uyguluyorsa ve burumda bir satırdaki en fazla karakter sayısı aşılıyorsa her arabirim adından sonra gelen virgüle göre satırlara bölünebilir. Sınıf adları hizalanmış olacak şekilde de girintili yazılır.

```
classSampleClass
implementsBarInterface,
BazInterface
{
}
```

Sınıf Değişkenleri

Sınıf değişkenleri de önceden bahsettiğimiz gibi isimlendirilir.
Sınıfta tanımlanmış herhangi bir değişken sınıfın en üstünde, fonksiyonlardan önce yer almalıdır.
var yapıcısına izin (*var construct*) izin verilmez. Sınıf değişkenleri görünürlüklerini private, public ya da protected değiştiricileri ile belli ederler. Bir sınıf değişkenini public olarak niteleyip erişmeye izin verilse de *set* ve *get* erişim metotlarını kullanmak tercih edilir.

Fonksiyonlar ve Metotlar

Fonksiyon ve Metot Tanımlama

Fonksiyonlar da önceden bahsettiğimiz şekilde isimlendirilir.
Sınıflarda yer alan metotlar görünürlüklerini private, protected ya da public değiştiricileri ile belli ederler.
Sınıf tanımlarken olduğu gibi kırlangıç parantezler fonksiyonlarda da yeni satırda yazılır. Fonksiyon adı ile parametre listesinin başlangıç parantezi arasında boşluk karakteri olmamalıdır.
Global kapsama sahip fonksiyonların kullanılmaması şiddetle önerilir.
Aşağıda bir sınıf içinde tanımlanmış fonksiyon için kabul edilebilir bir örnektir:

```
/**
 * DocumentationBlock Here
 */
classFoo
{
   /**
    * DocumentationBlock Here
    */
publicfunction bar()
{
      // allcontents of function
      // must be indentedfourspaces
}
```

```
}
```

Parametre listesinin bir satırda yer alabilecek en fazla karakter sayısını aştığı durumlarda birden çok satıra bölebilirsiniz. Karakter sayısını aşan parametreler alt satıra aktarılıp, fonksiyon ya da parametre tanımlama bölümüne göre bir birim girintili olarak yazılır. Bu durumda parametre listesinin kapanış parantezi de tek başına bir satırda yer alır ve fonksiyonun açılış kırlangıç paranteziyle aralarında bir karakter boşluk olacak şekilde aynı satırda yer alır. Aşağıdaki kod bu duruma örnektir:

```
/**
 * DocumentationBlock Here
 */
classFoo
{
   /**
    * DocumentationBlock Here
    */
publicfunction bar($arg1, $arg2, $arg3,
      $arg4, $arg5, $arg6
   ) {
      // allcontents of function
      // must be indentedfourspaces
}
}
```

Bir fonksiyon tanımlarken parametre iletmenin önerilen tek yolu referans olarak iletmektir (*pass-by-reference*).

```
/**
 * DocumentationBlock Here
 */
classFoo
{
   /**
    * DocumentationBlock Here
    */
publicfunction bar(&$baz)
   {}
}
```

Fonksiyonda tanımlanmamış olmasına rağmen referans olarak iletilen parametreler kesinlikle önerilmez. Fonksiyonlardan döndürülen değerler parantez içinde olmamalı. Bu durum okunabilirliği azaltır.

```
/**
 * DocumentationBlock Here
 */
classFoo
{
   /**
    * YANLIŞ
    */
publicfunction bar()
{
return($this->bar);
}

   /**
    * DOĞRU
    */
publicfunction bar()
{
return $this->bar;
}
```

Fonksiyon ve Metot Kullanımı

Fonksiyon parametreleri virgülden sonra birer boşluk karakteri ile ayrılmalı. Aşağıdaki örnek üç parametre alan bir fonksiyon için uygun yazım biçimidir.

```
threeArguments(1, 2, 3);
```

Fonksiyon tanımlanırken belirtilmediyse, çağırma sırasında parametreleri referans olarak iletmek kesinlikle önerilmez. Bir önceki kısımda bunun uygun yöntemini belirtmiştik. Bir fonksiyona parametre olarak dize gönderilecekse, fonksiyon çağrımı *array* oluşturucusunu içerebilir ve dizeler bölümünde anlattığımız yazım biçimleri yine geçerlidir.

```
threeArguments(array(1, 2, 3), 2, 3);

threeArguments(array(1, 2, 3, 'Zend', 'Studio',
                     $a, $b, $c,
                     56.44, $d, 500), 2, 3);

threeArguments(array(
   1, 2, 3, 'Zend', 'Studio',
   $a, $b, $c,
   56.44, $d, 500
), 2, 3);
```

Kontrol Yapıları

If/Else/Elseif

If ve *elseif* üzerine kurulu kontrol yapıları koşul bölümünün açılış parantezinden önce ve kapanış parantezinden sonra bir boşluk karakteri ile ayrılmalıdır.
Koşul bölümünde yer alan operatörler okunabilirlik için boşluk karakteri ile ayrılmalıdır. Daha büyük koşul bölümleri için mantıksal olarak daha fazla bölüm ve bunları kapsayan parantezler kullanılabilir.
Kontrol yapılarının açılış kırlangıç parantezi koşul bölümü ile aynı satıra yazılırken kapanış kırlangıç parantezi ise kendi satırına sahip olmalıdır. Kırlangıç parantezler arasında yer alan kodlar ise dört boşluk ile girintili olarak yazılmalıdır.

```
if ($a != 2) {
   $a = 2;
}
```

Eğer koşul bölümü birden çok koşuldan oluşuyorsa ve bir satırda yer alabilecek en fazla karakter sayısını aşıyorsa birden çok satıra bölünebilir. Bu durumda her mantıksal operatörden önce satırı bölüp her satırı mantıksal operatörler alt alta gelecek şekilde girintili yazmalısınız. Koşul bölümünün kapanış parantezi kontrol yapısının açılış kırlangıç paranteziyle aynı satırda yazılmalı ve aralarında bir boşluk karakteri olmalıdır.

```
if(($a == $b)
&& ($b == $c)
   || (Foo::CONST == $d)
```

```
) {
   $a = $d;
}
```

Bu şekilde girintili yazarak sonradan koşul eklemeyi ve silmeyi kolaylaştırabilirsiniz.
Elseif ya da *else* barındıran *if* yapılarının biçimlendirilmesi de sadece *if* barındıran kontrol yapılarına benzer.
Aşağıdaki kodlar bu durumun örnekleridir.

```
if ($a != 2) {
   $a = 2;
} else {
   $a = 7;
}

if ($a != 2) {
   $a = 2;
}elseif ($a == 3) {
   $a = 4;
} else {
   $a = 7;
}

if(($a == $b)
&& ($b == $c)
   || (Foo::CONST == $d)
) {
   $a = $d;
} elseif(($a != $b)
        || ($b != $c)
) {
   $a = $c;
} else {
   $a = $b;
}
```

URL'ler Hakkında Bilmeniz Gereken Herşey

Cankat Akdemir

Bu günlerde ve bu dönemde bir uygulama geliştiricisiyseniz kariyerinizin bir yerinde mutlaka web ile ilgili bir işle ilgilenmeniz gerekecektir. Bu da URL'ler ile şimdi ya da sonra mutlaka uğraşmanız gerektiğini gösterir. Hepimiz URL'lerin ne olduklarını biliyoruz. Fakat URL'leri bir kullanıcı olarak bilmekle bir programcının bilmesi gerektiği gibi bilmek arasında fark var.
Bir programcı olarak URL'lerle ilgili bir şeyleri bilmeme hakkınız yok. Çünkü zaten bilinecek çok da şey yok. Fakat birçok deneyimli programcının bile URL'lerle ilgili bilgilerinde kara noktaların olduğuna şahit oldum. Böylece, URL'lerle ilgili her şeyi içeren hızlı bir turun işe yarayacağını düşündüm. Hazırlanın, çok uzun sürmeyecek.

Bir URL'nin Yapısı

Kolay yerden geldi. Bir URL, HTTP ile başlar ve .com ile biter değil mi? Birçok URL benzer şekildedir. URL'ler aşağıda gördüğünüz dokuz yapıdan oluşur.
<şema>://<kullanıcı>:<parola>@<sunucu>:<port>/<yol>;<parametreler>?<sorgu>#<bölüm>
URL'ler her zaman bu yapıların hepsini kapsamayabilir. En çok kullanılan yapılar, sizin de bileceğiniz üzere, *şema*, *sunucu* ve *yol*dur. Şimdi bu dokuz yapıya sırasıyla bakalım.

- **Şema –** URL tarafından adresi gösterilen kaynağa hangi protokol (*http, ftp gibi*) kullanılarak ulaşılacağını belirtir. Birçok farklı şema mevcuttur. Bir şema IANA'a kayıt olduğunda resmilik kazansa da resmi olmamasına rağmen sık kullanılan birçok şema da (*sftp ya da svn gibi*) vardır. Şema bir harf ile başlamak zorundadır ve URL'nin kalanından URL'deki ilk : (*iki nokta üst üste*) karakteri ile ayrılır.
- **Kullanıcı –** Parola, sunucu ve port kısımlarıyla birlikte URL'nin yetkilendirme bölümünü oluşturur. Bazı şemalar kaynaklara ulaşmak için yetkilendirme bilgisine ihtiyaç duyarlar. Burada belirttiğiniz kullanıcı adı bu yetkilendirme için gerekli olan kullanıcı adıdır. Kullanıcı ad ve parola bilgileri en çok FTP bağlantılarında kullanılır. HTTP bağlantılarında ise seyrek de olsa denk gelebilirsiniz.
- **Parola –** URL'nin yetkilendirme bölümünün diğer üyesi olan parola, kullanıcı adından : (*iki nokta üst üste*) karakteri ile ayrılır. Kullanıcı adı ve parola bölümü de beraberce sunucu adından @ işareti ile ayrılırlar. URL'de sadece kullanıcı adını ya da hem kullanıcı adını hem de parolayı belirtebilirsiniz. Örneğin:
ftp://kullanici@deneme.com
ftp://kullanici:parola@deneme.com
Eğer erişmeye çalıştığınız URL yetkilendirmeye gerek duymasına rağmen siz bir kullanıcı adı ve parola belirtmediyseniz, kullandığınız program (*örneğin internet tarayıcınız*) bazı varsayılanları kullanmayı deneyebilir.
- **Sunucu –** Yukarıda belirttiğim gibi sunucu da URL'nin yetkilendirme bölümlerinden birini oluşturur. Sunucu, bir alan adı ya da IP adresi olabilir. Hepimizin bileceği gibi alan adı zaten *DNS lookup* ile bir IP adresine çözümlenecektir.
- **Yol –** Yol URL'deki diğer kısımlardan / (*slash*) karakteri ile ayrılır. Yol, / karakteri ile ayrılmış birden çok bölümden oluşabilir. Temel olarak yol bize sunucuda yer alan bir kaynağın nerede bulunduğunu belirtir. Her yol bölünü ; (*noktalı virgül*) ile ayrılmış parametreler barındırabilir. Örneğin:
http://www.deneme.com/falanca;parametre1=foo/deneme;parametre2=bar/yol.html
Yukarıdaki URL tümüyle geçerli olsa da yol bölümlerinin parametre barındırması özelliği neredeyse hiç kullanılmadı (*kişisel olarak ben hiç görmedim*).
- **Parametreler –** Parametreler genelde yoldan sonra fakat sorgudan önce yer alır. Her parametre ; (*noktalı* virgül) karakteriyle diğerinden ayrılır. Örneğin:
http://www.deneme.com/deneme/icin/yol.html;param1=foo;param2=bar
Yukarıda söylediğim gibi bu da pek sık kullanılmaz.
- **Sorgu –**Sorgular, her web geliştiricinin bildiği gibi oldukça yaygındır. Sunucuda yer alan bir kaynağa parametre göndermenin en sık başvurulan ve tercih edilen yöntemi sorgulardır. Birbirinden & işareti, URL'den de ? (*soru işareti*) ile ayrılan **anahtar=değer** çiftlerinden oluşur. Bilmiyor olabileceğiniz bir özellikleriyse birbirlerinden ; (*noktalı virgül*) karakteri ile de ayrılabilir oldukları. Aşağıdaki iki URL de aynı kaynağı işaret eder:
http://www.deneme.com/deneme/icin/yol.html?param1=foo&param2=bar
http://www.deneme.com/deneme/icin/yol.html?param1=foo;param2=bar
- **Bölüm –** URL'nin opsiyonlu alanlarından biri olan bölüm, sunucuda yer alan belirli bir alanı işaret eder. Bunu genelde HTML belgesinin bir kısmına bağlantı verirken kullanırız. URL'nin işaret ettiği kaynak sunucudan istenirken, istemci (*örneğin tarayıcı*) bu kısmı sunucuya göndermez (*en azından HTTP bununla ilgilenmez*). Kaynak sunucudan geldiğinde, istemci belirtilen kısmı kendi görüntüler.

URL'lerin yapısıyla ilgili bilmeniz gerekenler bu kadar.

URL'lerde Özel Karakterler

Bir URL'de kullanılacak güvenli karakterlerin hangileri olduğu ve URL'nin nasıl kodlanması gerektiğiyle ilgili fazlaca kafa karışıklığı var. Geliştiriciler bu sorunları genel bilgilerine dayanarak aşmaya çalışıyor (*örneğin, / ve : karakterleri URL'de özel bir anlam taşıdığı için kodlanmalı gibi*). Fikir yürütmenize gerek yok. Oldukça kolay olan bu konuyu hemen öğrenebilirsiniz.
Söz konusu URL'ler olunca, dikkat etmeniz gereken birkaç

grup karakter var. İlk önce, **rezerve karakterler** denen ve URL içerisinde özel anlam ifade eden karakterler:
";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
Bunlar URL içerisinde aynen yazıldıkları gibi kullanılırlar ve URL'nin anlam ifade etmesinde önemlidirler (*örneğin bazı kısımları birbirinden ayırmak gibi*). Eğer URL'nin bir yerinde bu karakteri anlamı dışında kullanmanız gerekirse, mutlaka karakteri *escape* etmeniz gerekir.
Diğer grup ise **rezerve olmayan karakterler**. Aşağıdaki karakterlerden oluşuyor:
"-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
Bu karakterleri URL'nin istediğiniz bölümünde yazıldığı gibi kullanabilirsiniz (*dikkat edin bu karakterler URL için özel anlam ifade etmiyor*). Bu da demektir ki bu karakterleri kullanırken *encode* ya da *escape* gibi işlemler yapmanıza gerek yok.**URL'nin anlamını değiştirmeden bu karakterleri escape edebilirsiniz fakat önerilmez.**
Üçüncü grup ise **'akılsızca'** karakterler grubu. Bu gruptan bir karakteri URL'de kullanmak pek de mantıklı değil.
"{" | "}" | "|" | "\" | "^" | "[" | "]" | "`"
Bu karakterlerin URL'de kullanılmasının akılsızca olmasının sebebi ağ geçitlerinin bazı durumlarda bu karakterleri değiştirme ihtimalinin olması. Ayrıca bu karakterler çoğu zaman sınırlayıcı olarak kullanılır. Bu karakterler URL'de kullanıldığında mutlaka bozulur diye bir şey yok fakat olabilir. Yani, bu karakterleri escape etmeden URL'de barındırmak sizin kendi sorumluluğunuzdadır. Kısaca, **bu karakterleri barındırma ihtimali olan URL bölümlerini (*örneğin sorgu parametresi gibi*) her zaman escape etmelisiniz.**
Son karakter grubu ise **hariç tutulanlar** grubu. Bu grup tüm ASCII kontrol karakterleri ve boşluk karakterine ek olarak aşağıdakileri de içeriyor (*sınırlayıcı karakterler olarak da bilinirler*):
"<" | ">" | "#" | "%" | '"'
Kontrol karakterleri yazdırılamaz olan US-ASCII karakterlerdir (*örneğin altılı 00-1F ve 7F gibi*). Bu karakterler herhangi bir URL bölümünde kullanılırken mutlaka *escape* edilmelidir. Birçoğu, # (*hash*) ve % (*yüzde*) gibi, URL'de özel anlam ifade etmektedir. Bu sebeple de *rezerve karakterler* ile eşit tutulmalıdır. Bu setteki diğer karakterler yazdırılamaz olduğundan onları göstermenin tek yolu *escape* etmektir. **<,> ve " işaretleri mutlaka escape edilmelidir. Çünkü bu karakterler URL'de kısıtlayıcı olarak görev yaparlar.**
URL'de yer alan bir karakteri *escape/encode* etmek için o karakteri temsil eden iki karakterli ASCII altılı (*hexedecimal*) değeri % (*yüzde*) karakterinin sonuna yazarız. *Örneğin, boşluk karakterinin encode edilmiş hali %20'dir ki bunu birçoğumuz görmüşüzdür. % Karakterinin kendisi de %25 ile encode edilir.*
URL'lerde kullanılan özel karakterler hakkında bilmeniz gerekenler bu kadar. Ayrıca, alfa numerik karakterleri URL'lerde istediğiniz gibi kullanabileceğinizi ve encode etmeniz gerekmediğini unutmayın.
Bunlara ek olarak unutmamanız gereken birkaç şey daha var.
Bir URL her zaman encode edilmiş haliyle kullanılmalıdır. Bunun tek istisnası ise bir sebeple URL'yi parçalamak istemenizdir. URL'nin her bölümü ayrı ayrı encode edilmelidir. Sebebi belli: Zaten encode edilmiş bir URL'yi yeniden encode etmek istemezsiniz. Eğer böyle bir hata yaparsanız rezerve karakterler ne zaman bir kendi görevlerinde kullanılmış (*encode gerektirmeyen durum*) ve ne zaman bir URL bileşeninin değeri olarak kullanılmış (*encode gerektiren durum*) ayırt edemezsiniz. Kısaca, asla bir URL'yi iki defa encode ya da decode etmeyi denememelisiniz. URL'yi bir defa encode ettikten sonra iki defa decode ederseniz o URL'yi mahvetmiş olursunuz.
Örneğin:
http://blah.com/yadda.html?param1=abc%613
Bu URL encode edildiğinde aşağıdaki şekilde görünür:
http://blah.com/yadda.html?param1=abc%25613
Bu URL'yi şimdi iki defa decode ederseniz ne olacağına bakalım.
Birinci decode işleminde
http://blah.com/yadda.html?param1=abc%613 elde edersiniz ki bu olması gerekendir. Fakat ikinci encode işleminden sonra
http://blah.com/yadda.html?param1=abca3 URL'sini elde edersiniz ve bu URL olması gereken değildir.
Bu arada bu tanımların hepsinin RFC 2396'da anlatıldığını belirteyim (http://www.ietf.org/rfc/rfc2396.txt). Sayfada bu tanımların hepsini okuyabilirsiniz fakat okuması en keyifli yazılardan biri olduğunu garanti edemem.

URL'lerde Görecelik Kuramı ya da Absolute vs Relative URL

Bir geliştiricinin URL'ler hakkında bilmesi gereken son şey göreceli (relative) ve kesin (*absolute*) URL arasındaki fark ve bir göreceli URL'yi nasıl kesin URL'ye dönüştürebileceğidir. *Bu arada gelin bunlar da Türkçe olmasın. Relative ve absolute olarak kalsınlar. (Y.N)*
Kesin URL'leri tanıması kolay. Bir URL'nin başında şema bölümü (*HTTP gibi*) yer alıyorsa bu URL absolute biçimli bir URL'dir. Relative URL'ler ise, hm, biraz daha karışık.
Göreceli bir URL her zaman bir başka URL'ye göre göreceli olarak yorumlanır (*tanıma bak*), **bu diğer URL ise temel (*base*) URL olarak bilinir.** Relative bir URL'yi absolute biçimine dönüştürmek için önce temel URL'yi bulmamız gerekir. Ardından relative URL'nin yazımına göre base URL ile birleştirerek absolute URL'yi elde ederiz.
Genelde relative URL'ler HTML belgelerinde yer alır. Bu durumda temel URL'yi bulmanın iki yolu vardır:

1. Temel (*base*) URL, HTML belgesi içinde ***<base>*** etiketi ile tanımlanmıştır.
2. Etiket belirtilmediyse mevcut HTML belgesinin URL'si temel URL sayılır.

Temel URL'yi bulduktan sonra relative URL'yi absoluıte hale dönüştürebiliriz. İlk önce relative URL'yi parçalarına ayırmamız gerek (*yukarıdaki bölümde okuduğunuz gibi şema, yetki, yol, bölüm gibi parçalar*). Bu işlem bittikten sonra birkaç özel duruma dikkat etmelisiniz. Eğer bu özel durumlardan biri gerçekleşiyorsa, relative sandığınız URL aslında hiç de relative değildir.

- Şema, yetki ya da yol bölümleri yoksa bu relative URL, temel URL'ye bir referanstır.

- Eğer şema bölümü varsa, bu relative URL aslında absolute bir URL'dir ve öyle değerlendirilmelidir.
- Yetkilendirme kısmı varsa (*sunucu, port*) ama şema kısmı yoksa bu URL büyük ihtimalle bir ağ konumuna işaret ediyordur. Temel URL'den şemayı alıp :// ekledikten sonra relative URL'yi aynen ekleriz.

Eğer bu üç durumdan biri gerçekleşmiyorsa elimizde tam anlamıyla bir relative URL var demektir. Bu durumda şöyle devam ederiz:

- Şema ve yetki (*sunucu, port*) bilgilerini temel URL'den alırız.
- Eğer relative URL / ile başlıyorsa aslında absolute URL'dir. Temel URL'den aldığımız şema ve yetki kısmına uygun ayraçlarla ekleyerek absolute URL'yi elde ederiz.
- Eğer relative URL / ile başlamıyorsa, temel URL'de sondaki / işaretinden sonra gelen her şeyi yok sayarak yol kısmını alırız.
- Ardından relative URL'yi alır ve elde ettiğimiz yola ekleriz. Bundan sonra ise relative URL'nin ilk birkaç karakterini bakarak ne yapmamız gerektiğini çözebiliriz.
- Sonuçta elimize geçen URL'de ./ varsa bunu komple sileriz.
- ../ varsa bir önceki bölümle beraber sileriz. Örneğin, ***<bolum>/../*** silinir. Bunu URL'de hiç ../ kalmayana kadar yaparız.
- URL'nin sonunda .. varsa bir önceki bölümle beraber sileriz. Örneğin, ***<bolum>/..*** silinir. Bunu URL'de hiç .. kalmayana kadar yaparız.
- Eğer URL . (*nokta*) ile bitiyorsa onu da sileriz.

Bu işlemden sonra ise relative URL'de yer alan sorguları ve parametreleri uygun ayraçlarla elde ettiğimiz URL'ye ekleyerek absolute haline ulaşırız.
Yukarıdaki algoritma kullanılarak oluşturulmuş absolute URL'leri aşağıdaki örneklerde bulabilirsiniz.

Temel URL:
http://www.blah.com/yadda1/yadda2/yadda3?param1=foo#bar
Relative URL: rel1
Son Absolute: http://www.blah.com/yadda1/yadda2/rel1

Temel URL:
http://www.blah.com/yadda1/yadda2/yadda3?param1=foo#bar
Relative URL: /rel1
Son Absolute: http://www.blah.com/rel1

Temel URL:
http://www.blah.com/yadda1/yadda2/yadda3?param1=foo#bar
Relative URL: ../rel1
Son Absolute: http://www.blah.com/yadda1/rel1

Temel URL:
http://www.blah.com/yadda1/yadda2/yadda3?param1=foo#bar
Relative URL: ./rel1?param2=baz#bar2
Son Absolute:
http://www.blah.com/yadda1/yadda2/rel1?param2=baz#bar2

Temel URL:
http://www.blah.com/yadda1/yadda2/yadda3?param1=foo#bar
Relative URL:..
Son Absolute: http://www.blah.com/yadda1/

Artık herhangi bir relative URL'yi absolute haline dönüştürebilirsiniz. Farklı relative URL biçimlerini ve anlamlarını da öğrenmiş oldunuz. Bu bilgiler benim web geliştirme yaptığım her zaman işime yaradı.
URL'ler hakkında bilmeniz gerekenler –*göreceli olarak*- gerçekten bu kadar. Artık bir dahaki sefere *ben bilmiyordum* deme hakkınız yok . Bir dahaki sefere demişken, konu URL'ler olunca önemli işlerden biri de bir metnin URL barındırıp barındırmadığını anlamaktır. Bir dahaki yazıda, Regular Expression kullanarak bir metnin URL olup olmadığını anlamak ve metinden URL'leri ayıklamak işlemlerinden bahsedeceğim.
Okuduğunuz yazıAlan Skorkin'in 4 Mayıs 2010 tarihinde sitesinde yayınladığı "What Every Developer Should Know About URLs" yazısının Cankat Akdemir tarafından yapılmış çevirisidir. Yazının sonunda belirtildiği gibi Skorkin sitesinde yeni bir makale yayınladığı zaman dergimizde de Türkçe olarak yayınlanacaktır.
Skorkin'in web sitesi: http://www.skorks.com
Yazının sayfası: http://www.skorks.com/2010/05/what-every-developer-should-know-about-urls/

www.phpdergi.com