



Pygtk Kılavuzu

Sürüm 2.x

Fırat Özgül (istihza)

30/01/2010

Contents

1	Temel Bilgiler	1
2	Pencere İşlemleri	4
2.1	Pencere Oluşturmak	4
2.2	Pencereye Başlık Ekleme	7
2.3	Pencere Boyutunu Ayarlamak	8
2.4	Pencere Konumunu Ayarlamak	9
2.5	Pencereleri Sağlıklı bir Şekilde Kapatmak	10
2.6	Pencerelere Simge Ekleme	14
2.7	Boyutlandırılmayan Pencereler Oluşturmak	15
2.8	Saydam/Şeffaf Pencereler Oluşturmak	15
3	Pygtk’de Pencere Araçları (1. Bölüm)	18
3.1	“Label” Pencere Aracı	18
3.2	“Button” Pencere Aracı	30
4	Pencere Araçlarının Yerleşimi	41
4.1	Fixed() Haznesi	41
4.2	Table() Pencere Aracı	48
4.3	Box() Sınıfı	60
4.4	HBox() Haznesi	60
4.5	VBox() Haznesi	65
4.6	HButtonBox() ve VButtonBox() Hazneleri	68
5	Pygtk’de Pencere Araçları (2. Bölüm)	73
5.1	“Entry” Pencere Aracı	73
5.2	“ToggleButton” Pencere Aracı	89
5.3	“CheckButton” Pencere Aracı	92
5.4	“RadioButton” Pencere Aracı	93

6	Pygtk’de Resimlerle Çalışmak	97
6.1	Pencere Araçlarına Stoktan Resim Eklemek	97
6.2	Pencere Araçlarına Dosyadan Resim Eklemek	99
6.3	Pencere Araçlarına Resim ve Etiket Eklemek	99

Temel Bilgiler

Python programlama dilini belli bir seviyeye kadar öğrendikten sonra muhtemelen konsol tabanlı programlar yazmak artık size yeterli gelmemeye başlayacaktır. Eminim siz de yazdığınız programların güzel ve işlevli bir arayüze sahip olmasını isteyeceksiniz... Python ile arayüz programlamak için elimizde bol miktarda seçenek var. Örneğin, eğer arayüz programlama konusunda yeniyseniz, Python programlama dilinin (yarı-)resmi arayüztakımı olan Tkinter'i öğrenmeyi seçebilirsiniz. Tkinter, Python'un Windows sürümünde kurulumla birlikte gelir. GNU/Linux kullanıcıları ise, kullandıkları dağıtımın paket yöneticisini kullanarak Tkinter'i sistemlerine rahatlıkla kurabilirler. Tkinter arayüz takımını kullanarak gayet işlevli arayüzler hazırlayabilirsiniz. Üstelik Tkinter'i öğrenmek ve kullanmak oldukça basittir. Tkinter'in mevcut arayüz takımları içinde en kolay öğrenileni olduğunu rahatlıkla söyleyebilirim.

Ancak Tkinter bazı yönlerden eksiklikler barındıran bir arayüz takımıdır. Örneğin Tkinter'deki pencere araçlarının yetersizliği herkes tarafından bilinen ve ifade edilen bir gerçek... Bazı işleri Tkinter ile halledebilmek için epey uğraşmak, bol bol kod yazmak gerekebiliyor. Ayrıca Tkinter ile üretilmiş programların görünüş olarak pek cazip olmadığı da yine pek çok kişi tarafından söylenegelmektedir. Ancak yine de Tkinter'in sadeliği ve basitliği, arayüz programlamaya yeni başlayan kimseler için kolaylaştırıcı etkenlerdir. Dediğim gibi, Tkinter'in bazı eksiklikleri olsa da, bu arayüz takımını kullanarak epey faydalı programlar üretebiliriz. Eğer arayüz programlamaya Tkinter ile başlamak isterseniz, sitemizden Tkinter'in nasıl kullanılacağına ilişkin pek çok ayrıntılı bilgiye erişebilirsiniz.

Elbette Python'da arayüz tasarlamak için tek seçeneğimiz Tkinter değil. Tkinter dışında, örneğin GTK adlı arayüz takımını da kullanmayı tercih edebiliriz. Tkinter'in aksine, GTK adlı arayüz takımı oldukça gelişmiş özellikler barındırır. Bu takımını kullanarak hem işlev hem de görünüş olarak son derece olgun programlar yazabilirsiniz. Örneğin GNOME masaüstü ortamının temeli GTK'dır. Ayrıca GNU/Linux'un medarı iftiharını GIMP (GNU Görüntü Düzenleme Yazılımı) bu takım kullanılarak yazılmıştır. Zaten GTK da GIMP programını yazmak için üretilmiştir!.. "GTK"nın açılımına baktığımızda bu durumu net olarak görebiliriz: "*Gimp Toolkit*" (GIMP Araç Takımı).

Dediğimiz gibi GTK, arayüz geliştirmek için tamamen C dili ile yazılmış bir kütüphanedir. GTK'nın lisansı (LGPL), herhangi bir lisans ücreti ödemeksizin, ticari ve ticari olmayan, özgür ve özgür olmayan programlar geliştirmeye izin verir.

GTK'yı Python ile birlikte kullanabilmek için, "Pygtk" adlı bağlayıcı katmandan yararlanacağız. Pygtk, GTK ile Python arasında iletişim kurmamızı sağlayan bir kütüphanedir. GTK'nın öteki dillerdeki bağlayıcılarının listesine şu adresten erişebilirsiniz: <http://www.gtk.org/language-bindings.html>

Pygtk hem Windows'ta hem de GNU/Linux'ta çalıştırılabilir. Bu anlamda Pygtk, birden fazla platforma destek veren bir yapı sunar bize...

Biraz sonra Pygtk ile ilgili ilk örneklerimizi vermeye başlayacağız, ama Pygtk'yi kullanmadan önce bazı modülleri tanımamızda fayda var. Pygtk arayüz takımını kullanırken temel olarak iki modülden faydalanacağız. Bunlardan biri "Pygtk", öbürü ise "gtk" adlı modül. Esasında asıl işi yapan, "gtk" modülüdür. "Pygtk" adlı modülü, temel olarak, kullandığımız sistemdeki Pygtk sürümünü kontrol etmek için kullanıyoruz. Modern Pygtk uygulamaları, Pygtk'nin "2.x" sürümleri kullanılarak yazılıyor. Eğer sistemimizde Pygtk'nin eski ve yeni sürümleri bir arada bulunuyorsa, Pygtk modülünü kullanarak, programınızın istediğiniz sürümle çalışmasını sağlayabilirsiniz. Biraz sonra bunun nasıl olacağını göreceğiz. Ama isterseniz önce sistemimizdeki GTK ve Pygtk sürümlerinin hangileri olduğunu nasıl öğrenebileceğimize bakalım. Bunun için şu kodları kullanıyoruz (Eğer aşağıdaki kodlarda herhangi bir hata alıyorsanız, okumaya devam edin...):

Önce gtk modülünü içe aktaralım:

```
>>> import gtk
```

Ardından gtk sürümünü kontrol edelim:

```
>>> gtk.gtk_version
```

Şimdi de Pygtk sürümüne bakıyoruz:

```
>>> gtk.pygtk_version
```

Eğer isterseniz doğrudan Pygtk modülünü kullanarak da sisteminizdeki bütün GTK sürümlerini öğrenebilirsiniz:

```
>>> import pygtk
>>> pygtk._get_available_versions()
```

Bu komut size sisteminizdeki GTK sürümlerini bir sözlük olarak verecektir. Bu komuttan şuna benzer bir çıktı elde ediyoruz:

```
{'2.0': '/var/lib/python-support/python2.6/gtk-2.0'}
```

Buradan anladığımıza göre, sistemimizde GTK'nin "2.0" sürümü kurulu. Bu sürümün sistemdeki yeri ise `"/var/lib/python-support/python2.6/gtk-2.0"`

Aynı komutu Windows üzerinde verdiğimizde ise şöyle bir çıktı elde ediyoruz:

```
{'2.0': 'c:\\python26\\lib\\site-packages\\gtk-2.0'}
```

Eğer daha `"import gtk"` veya `"import pygtk"` aşamasında hata alıyorsanız, sisteminizde GTK paketi kurulu değil demektir. Eğer bir GNU/Linux kullanıcısı iseniz, paket yöneticinizi kullanarak gtk2 paketini kurabilirsiniz... Ya da isterseniz doğrudan Pygtk paketini kurmayı de tercih edebilirsiniz. Zira Pygtk paketini kurduğunuzda, muhtemelen bu paketin bağımlılıkları arasında gtk2 de bulunacağı için gtk2 paketi de sisteminize kurulacaktır...

Windows kullanıcıları ise Pygtk'yi kullanabilmek için şu paketleri indirip kuracak:

1. `gtk+-win32-runtime`
2. `Pygtk`

3. `pycairo`

4. `pygobject`

Windows kullanıcıları yukarıda verdiğim bağlantılara tıklayarak, ilgili paketlerin en son sürümlerini sistemlerine kurabilir. Elbette sisteminizde Python'un da kurulu olduğunu varsayıyorum...

Not: Eğer Windows'ta yukarıdaki adımları takip ederek PyGtk'yi kurduktan sonra herhangi bir PyGtk programını çalıştırmak istediğinizde `ImportError: DLL load failed: Belirtilen modül bulunamadı.` şeklinde bir hata alıyorsanız <http://www.istihza.com/blog/windowsta-bir-pygtk-hatasi.html/> adresinde gösterdiğimiz yöntemi uygulayın.

Gördüğümüz gibi, GTK ve Pygtk'yi kurmak GNU/Linux sistemlerinde çok basittir. Eğer GNU/Linux dağıtımlarından birini kullanıyorsanız paket yöneticiniz aracılığıyla gerekli paketleri kurmak birkaç tıklamalık bir iştir. Pygtk'yi Windows'a kurmak ise bazı durumlarda biraz uğraştırabilir. Windows'ta özellikle bazı paketlerin birbiriyle uyuşma sorunu nedeniyle Pygtk'yi kurmakta zorlanabilirsiniz. Eğer Pygtk'yi kurarken altından kalkamadığınız bir sorunla karşılaşırsanız `kistihza [at] yahoo [nokta] com` adresinden bana ulaşabilirsiniz. Ben size elimden geldiğince yardımcı olmaya çalışırım...

Pencere İşlemleri

Pygtk ile çalışabilmek için gereken temel bilgileri bir önceki bölümde verdiğimizde göre artık asıl konumuza gelebiliriz. Bu bölümde Pygtk'yi kullanarak nasıl pencere oluşturacağımızı, oluşturduğumuz bu pencereyi nasıl şekillendirebileceğimizi inceleyeceğiz.

Bu bölümde inceleyeceğimiz ilk konu, boş bir pencere oluşturmak... O halde hiç gecikmeden yolumuza devam edelim.

2.1 Pencere Oluşturmak

Arayüz programlamada en temel konu, içi boş da olsa çalışan bir pencere oluşturabilmektir. İşte biz de bu bölümde bu en temel konuya değineceğiz.

Pygtk ile boş bir pencere oluşturmak için şu kodları yazıyoruz:

```
import gtk

pencere = gtk.Window()
pencere.show()

gtk.main()
```

Gördüğümüz gibi, Pygtk ile sadece dört satır yazarak boş bir pencere meydana getirebiliyoruz. Şimdi biz bu satırları tek tek inceleyerek, durumu olabildiğince anlaşılır bir hale getirmeye çalışacağız...

İlk satırda gördüğümüz “import gtk” kodu, ana modülümüz olan “gtk”yi içe aktarmamızı sağlıyor. Bu modülü mutlaka içe aktarmamız gerekir. Aksi halde yazacağımız kodlar hata vermekten başka bir işe yaramayacaktır.

Daha sonraki satırda “pencere = gtk.Window()” biçiminde bir ifade görüyoruz. Burada gtk modülünün Window() adlı fonksiyonunu çağırdık ve buna “pencere” adını verdik. Bu satır bizim boş bir pencere oluşturmamızı sağlıyor. Ama bu, oluşturduğumuz pencerenin ekranda görünmesi için yeterli olmayacaktır. Oluşturduğumuz pencerenin ekranda görünmesi için ilk olarak “pencere.show()” satırı yardımıyla bir de “penceremizi göstermemiz” gerekiyor.

Aslında şimdiye kadar penceremizi ekranda gösterebilmek için yapmamız gereken her şeyi yaptık. Ama henüz programımızın kendisini başlatacak kodu yazmadık. Pencerenin ekranda

görünebilmesi için bir adım daha atmamız gerekecek. Bir Pygtk uygulamasının hayat bulabilmesi için gereken kod “gtk.main()”dir. Bunu da yukarıdaki kodlarda en son satıra yazdık.

Burada şunu bilmemiz gerekiyor: “gtk.Window()”, pencere **oluşturmamızı** sağlayan koddur. “pencere.show()” oluşturduğumuz bu pencereyi arayüz üzerinde **gösterebilmek** için gereklidir. “gtk.main()” satırı ise genel olarak uygulamanın bütününe hayat verme işlevi görür. Dolayısıyla uygulamanın bütününe hayat vermediğiniz sürece sadece pencerenin ekranda görünmesini sağlamak işinize yaramayacaktır...

Yukarıda yazdığımız kod ekrana boş bir pencere açmamızı sağlasa da aslında bu işi yapmak için standart yol değildir. Bir Pygtk uygulaması yazarken, yapmak istediğimiz asıl işi yapan kodları yazmanın yanısıra bazı başka hususları da göz önünde bulundurmamız gerekir. Mesela bir önceki bölümde bahsettiğimiz, “eski Pygtk sürümü” meselesi... Pygtk uygulamalarında, sistemdeki Pygtk sürümünü kontrol etmek oldukça standartlaşmış bir uygulamadır. O yüzden isterseniz biz de bu standarda uyup kodlarımızı biraz düzenleyelim:

```
import pygtk
pygtk.require20()
import gtk

pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
pencere.show()

gtk.main()
```

Burada yeni olarak, ilk baştaki iki satırı görüyoruz:

```
import pygtk
pygtk.require20()
```

Bu iki satırın her zaman en başa yazılması gerekir. Daha doğrusu bunları mutlaka gtk modülünden önce içe aktarmalıyız... Burada ilk satır pygtk adlı modülü içe aktarıyor. “pygtk.require20()” satırı ise, programımız için Pygtk’nin 2.x sürümünü kullanmak istediğimizi belirtiyor. Böylelikle, eğer sistemimizde mesela Pygtk’nin 1.x numaralı bir sürümü varsa, programımızın yanlışlıkla o sürümle açılmasını engellemiş oluyoruz...

Bu iki satırın bazı programlarda şöyle yazıldığını da görebilirsiniz:

```
import pygtk
pygtk.require("2.0")
```

İsterseniz bu yazım şeklini de benimseyebilirsiniz, ama yeni kodlarda artık genellikle “pygtk.require20()” biçimi tercih ediliyor.

Yukarıdaki kodlardaki başka bir yenilik ise “gtk.WINDOW_TOPLEVEL” parametresi. Bu parametre, penceremizin, bir pencerenin sahip olması gereken bütün özellikleri taşımasını sağlar. Biz bu parametreyi yazmasak da Pygtk varsayılan olarak o parametre yazılmış gibi davranacaktır... Dolayısıyla bu olmazsa penceremiz bozuk görünmez, ama yine de bu parametreyi kullanmak işimizi sağlama almamızı sağlar... Bu parametrenin tam olarak ne işe yaradığını anlamak için isterseniz “gtk.WINDOW_TOPLEVEL” yerine, “gtk.WINDOW_POPUP” yazarak programı çalıştırmayı deneyebilirsiniz. Bu şekilde oluşturulan pencerenin başlık çubuğu olmayacaktır. Dolayısıyla bu pencereyi ne sürükleyebilirsiniz, ne de kapatabilirsiniz... Böyle bir pencere oluşturmanın gerektiği yerler de vardır, ama şimdi değil... Bu pencereyi kapatmak için CTRL+C veya CTRL+Z tuşlarına basmanız gerekiyor...

Yukarıdaki kodları kullanarak gayet başarılı bir şekilde ilk penceremizi oluşturduk. Ama itiraf etmek gerekirse, Pygtk uygulamalarının yukarıdaki şekilde yazıldığı pek görülmez. Özellikle

internet üzerinde bulacağımız kaynaklar, kodları hep sınıflı yapı içinde gösteriyor. Hem kodlarımızın daha düzenli olması, hem sonradan kodlarımızın bakımını kolaylaştırmak hem de internet üzerindeki kaynaklardan daha verimli yararlanabilmek için biz de bu belgede kodlarımızı sınıflı yapı içinde sunmayı tercih edeceğiz. O halde yukarıdaki kodları standart bir Pygtk uygulaması haline nasıl getireceğimize bakalım:

```
import pygtk
pygtk.require20()
import gtk

class IlkPencere(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.show()

    def main(self):
        gtk.main()

ilk = IlkPencere()
ilk.main()
```

Gördüğünüz gibi, sınıfsız kodları sınıflı bir yapıya çevirmek o kadar da zor değil... Önce “IlkPencere” adında bir sınıf oluşturuyoruz. Sınıfımız, Python’daki yeni tip sınıfların gerektirdiği şekilde “object” sınıfını miras alıyor... Bu arada, eğer sınıflı yapılar konusunda eksikleriniz varsa [Nesne Tabanlı Programlama](#) konusuna göz gezdirmek isteyebilirsiniz...

Sınıfımızı tanımladıktan sonra, ilk olarak bir “__init__” fonksiyonu yazıyoruz. Programımız çalışmaya başlar başlamaz, bu fonksiyon içinde yer alan kodlar işletilecektir. Bu fonksiyon içinde boş penceremizi oluşturacak kodları yazıyoruz.

Programımızın dinamosu niteliğindeki “gtk.main()” için ise ayrı bir fonksiyon yazdık. Bu fonksiyonu “main()” olarak isimlendirdik. Elbette siz bu fonksiyona daha farklı bir isim verebilirsiniz. Ama açıkçası, bu fonksiyon adı da tıpkı “self” gibi kemikleşmiş bir adlandırmadır. Yazdığınız kodları okuyan kişilerin bir an da olsa duraksamaması için bu fonksiyonu sizin de “main()” olarak adlandırmanızı tavsiye ederim.

Kapanış bölümünde “IlkPencere()” adlı sınıfımızı [örnekliyoruz](#) (instantiate). Sınıf örneğimizin adı “ilk”. Sınıfımızı bu şekilde örnekledikten sonra, “ilk.main()” ifadesi yardımıyla “IlkPencere” adlı sınıfımızın içindeki “main()” fonksiyonunu çağırıyoruz. Böylece programımız çalışmaya başlıyor...

Yukarıda yazdığımız kodları çalıştırmak için, programımızın bulunduğu dizin içinde bir komut satırı açıp, orada şu komutu veriyoruz (programımızı “deneme.py” olarak adlandırdığımızı varsayarsak...)

```
python deneme.py
```

Bu arada, Pygtk henüz Python’un 3.x sürümleriyle uyumlu değildir. Dolayısıyla programımızı çalıştırmak için Python’un 2.x sürümlerinden birini kullanacağız. Özellikle Windows kullanıcılarının, “python” komutunu verdiklerinde Python’un hangi sürümünün çalıştığına dikkat etmeleri gerekiyor...

Bu noktada bir şey dikkatinizi çekmiş olmalı: Programlarımızı başarıyla çalıştırabiliyoruz, ama aynı başarıyla kapatamıyoruz!... Pencere üzerinde çarpı işaretine tıkladığımızda penceremiz kayboluyor, ama aslında programımız alttan alta çalışmaya devam ediyor. Programı komut satırından çalıştırdıysanız, programı kapattıktan sonra imlecin alt satıra geçmemiş olmasından

bunu anlayabilirsiniz... Bu durumda programı kapatmak için CTRL+C veya CTRL+Z tuşlarına basmamız gerekiyor...

Elbette programımız görünmediği halde alttan alta çalışmaya devam etmesi tercih edilecek bir şey değil. Yazdığımız programların kapanmasını nasıl sağlayabileceğimizi bir-iki bölüm sonra inceleyeceğiz. Biz şimdilik CTRL+C veya CTRL+Z tuşlarını kullanmayı sürdürüelim...

2.2 Pencereye Başlık Ekleme

Bir önceki bölümde boş bir pencereyi nasıl oluşturacağımızı görmüştük. İsterseniz artık oluşturduğumuz pencereyi biraz düzenlemeye, bu pencereye yeni özellikler eklemeye girişelim. Mesela oluşturduğumuz pencereye bir başlık ekleyelim...

Pygtk ile yazılmış bir programa başlık ekleme işi `set_title()` metoduyla yapılır. Gelin isterseniz hemen bununla ilgili küçük bir örnek verelim:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title("Bu bir başlıktır!")
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Eğer istediğimiz şey bir pencerenin başlığını ayarlamak değil de başlığın ne olduğunu bulmak ise, `get_title()` metodu işimize yarayacaktır:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title("Bu bir başlıktır!")
        print self.pencere.get_title()
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Programımızı çalıştırdığımızda, pencere başlığının konsol ekranına yazıldığını göreceğiz... Eğer pencere başlığı komut satırına yazılırken Türkçe harfler düzgün görünmüyorsa ilgili satırı şöyle yazabilirsiniz:

```
print unicode(self.pencere.get_title(), "utf-8")
```

Burada Türkçe karakterleri düzgün gösterebilmek için pencere başlığını Unicode kod çözücülerinden biri olan "utf-8" ile kodladık. Eğer Unicode konusu ilginizi çekiyorsa veya bu konuda eksikleriniz olduğunu düşünüyorsanız <http://www.istihza.com/py2/unicode.html> adresindeki makalemizi inceleyebilirsiniz...

Gördüğümüz gibi, penceremizin başlığını `set_title()` metodunu kullanarak kolayca belirleyebiliyoruz. `get_title()` metodunu kullanarak ise penceremizin başlığının ne olduğunu öğrenebiliyoruz. Ancak burada dikkatimizi çeken başka bir nokta var. Pencere boyutları tam istediğimiz gibi değil. Mesela pencere başlığını görebilmek için pencereyi elle boyutlandırmanız gerekiyor. İşte bir sonraki bölümümüzün konusu pencere boyutlarını elle değil de kodlar yardımıyla ayarlamak...

2.3 Pencere Boyutunu Ayarlamak

Pygtk ile oluşturduğumuz bir pencere varsayılan olarak 200x200 piksel boyutunda olacaktır. Ama bizim ihtiyacımız olan pencere boyutu her zaman bu olmayabilir. Eğer Pygtk penceresinin boyutunu kendimiz belirlemek istersek, `resize()` adlı metottan yararlanacağız:

```
#!/*-coding: utf-8 -*-  
  
import pygtk  
pygtk.require20()  
import gtk  
  
class Uygulama(object):  
    def __init__(self):  
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)  
        self.pencere.set_title("Bu bir başlıktır!")  
        self.pencere.resize(500, 500)  
        self.pencere.show()  
  
    def main(self):  
        gtk.main()  
  
uyg = Uygulama()  
uyg.main()
```

Bu şekilde, 500x500 piksel boyutunda bir pencere oluşturmuş olduk. `resize()` metodunun formülü şöyledir:

```
pencere.resize(genişlik, yükseklik)
```

Dolayısıyla, parantez içinde parametre olarak vereceğimiz ilk değer pencerenin genişliğini, ikinci değer ise pencerenin yüksekliğini belirleyecektir...

Eğer oluşturduğumuz pencerenin tam ekran olarak açılmasını istersek `maximize()` adlı metottan yararlanabiliriz:

```

#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title("Bu bir başlıktır!")
        self.pencere.maximize()
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Eğer programımızın çalışması esnasındaki bir aşamada tam ekran halindeki bir pencereyi tam ekran konumundan çıkarmak istersek de `unmaximize()` metodunu kullanabiliriz...

Pencerenin boyutunu değiştirmeyi öğrendik. Peki ya bu pencerenin konumunu değiştirmek istersek ne yapacağız? İsterseniz onu da bir sonraki bölümde anlatalım...

2.4 Pencere Konumunu Ayarlamak

Pencere boyutunu ayarlamayı bir önceki bölümde öğrenmiştik. Gelelim penceremizin konumunu ayarlamaya... Bu işi `move()` metodunu kullanarak gerçekleştireceğiz:

```

#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title("Bu bir başlıktır!")
        print self.pencere.get_title()
        self.pencere.resize(500, 500)
        self.pencere.move(700, 100)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada parantez içindeki ilk rakam pencerenin x düzlemindeki konumunu (soldan sağa), ikinci rakam ise y düzlemindeki konumunu gösteriyor (yukarıdan aşağıya).

Eğer pencremizi ekranın tam ortasında konumlandırmak istersek `set_position()` metodundan yararlanacağız:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title("Bu bir başlıktır!")
        print self.pencere.get_title()
        self.pencere.resize(500, 500)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Yukarıdaki kodlar, oluşturduğumuz pencerenin, ekranın ortasında açılmasını sağlayacaktır. Burada `gtk.WIN_POS_CENTER` parametresi yerine şunları da kullanabiliriz:

- gtk.WIN_POS_NONE:** Pencere konumu üzerinde herhangi bir değişiklik yapılmayacağını belirtir.
- gtk.WIN_POS_CENTER:** Pencereyi ekranın ortasına yerleştirir.
- gtk.WIN_POS_MOUSE:** Pencere, farenin o anda bulunduğu konumda açılır.
- gtk.WIN_POS_CENTER_ALWAYS:** Pencerenin boyutu değişse de, pencereyi hep ekranın ortasında tutar.
- gtk.WIN_POS_CENTER_ON_PARENT:** Eğer ikinci bir pencere varsa bu pencerenin ana pencereyi ortalamasını sağlar.

Pygtk'da pencere konumunu belirlemek için kullandığımız iki farklı metod olduğuna dikkat edin. Bunlardan birincisi `move()`, ikincisi ise `set_position()` metodudur.. `move()` metodunu, pencereyi ekran üzerinde özel bir konuma getirmek istediğimizde kullanıyoruz. `set_position()` metodu ise pencereyi belli konumlara yerleştirmemizi sağlıyor. Örneğin bir pencereyi soldan 300, üstten ise 500 piksele yerleştirmek istersek `move()` metodunu kullanacağız. Ama eğer pencereyi mesela ekranın ortasına yerleştireceksek, `set_position()` metodundan yararlanacağız.

2.5 Pencereyi Sağlıklı bir Şekilde Kapatmak

Önceki bölümlerde, Pygtk'yi kullanarak boş bir pencere oluşturmayı, pencereye bir başlık eklemeyi, oluşturduğumuz pencereyi konumlandırmayı ve boyutlandırmayı öğrenmiştik. Ama farkettik ki oluşturduğumuz bir pencereyi kapatmaya çalıştığımızda bir sorunla karşı karşıyayız. Sorun şu ki, aslında oluşturduğumuz pencereyi kapattığımızda programımız kapanmıyor. Programdan çıkmak için CTRL+C veya CTRL+Z tuşlarına basarak programı kapanmaya zorlamamız gerekiyor.. Bu bölümde, bir Pygtk uygulamasını sağlıklı bir şekilde nasıl kapatabileceğimizi göreceğiz. İsterseniz öncelikle tekrar boş bir pencere oluşturalım:

```
import pygtk
pygtk.require20()
import gtk

class Pencere(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Pencere()
uyg.main()
```

Yazdığımız bu kodların başarılı bir şekilde boş pencere oluşturacağını biliyoruz, ama aynı başarıyla kapanmayacağını da biliyoruz... Şimdi bu kodlara şöyle bir ekleme yapalım:

```
import pygtk
pygtk.require20()
import gtk

class Pencere(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Pencere()
uyg.main()
```

Buraya `self.pencere.connect("delete_event", gtk.main_quit)` diye yeni bir satır ekledik. Bu kodları çalıştırdığımızda, pencerenin çarpı düğmesine bastığımız zaman programımız düzgün bir şekilde kapanacaktır. Bu kodların getirdiği değişikliği konsoldaki imlecin durumundan da görebiliriz. Artık CTRL+C veya CTRL+Z tuşlarına basmak zorunda kalmadan programımızı sonlandırabiliyoruz.

Burada yaptığımız şey, penceremizin çarpı tuşuna bir görev atamaktan ibaret. Pencere üzerindeki çarpı tuşuna bastığımız anda, "gtk" modülünün `main_quit()` adlı metodu devreye girip programımızın kapanmasını sağlayacak. Pygtk'da "pencerenin çarpı tuşuna basma hareketi", "delete_event" olarak adlandırılır. Dolayısıyla yukarıdaki yeni satırı Türkçe olarak şöyle ifade edebiliriz:

```
pencere.bağla("çarpıya_basma_hareketi", gtk.programı_kapat)
```

Yukarıdaki Türkçe ifadeden anladığımız gibi, "bağla" kelimesine karşılık gelen "connect" ifadesi, temelde pencere aracını bir göreve "bağlıyor"... Bu görevi de parantez içinde tanımlıyoruz. Buna göre, pencere aracına bağladığımız görev, "üzerindeki çarpıya basıldığında" (`delete_event`), "programdan tamamen çıkmak" (`gtk.main_quit`)

Yukarıdaki kodlarda biz `gtk.main_quit()` metodunu doğrudan satır içine yazdık. Kolay yöntem budur. Ama eğer istersek bunu ayrı bir fonksiyon olarak da tanımlayabiliriz:

```

import pygtk
pygtk.require20()
import gtk

class Pencere(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)
        self.pencere.show()

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Pencere()
uyg.main()

```

Burada “kapat” adlı yeni bir fonksiyon tanımlayıp, connect() metodu içinde doğrudan bu fonksiyona gönderme yapıyoruz. Ayrıca “kapat” fonksiyonu içinde birtakım parametreler belirlediğimize de dikkat edin. Bir “olay” (event) tanımlarken (mesela “delete_event” bir olaydır), olaya ait fonksiyonun en az iki parametre alması gerekir. Bu parametrelerden biri “pencere aracına”, öteki ise “olayın kendisine” gönderme yapar. Bizim örneğimizde pencere aracına gönderme yapan parametreye “penar”, olaya gönderme yapan parametreye ise “event” adını verdik. Siz bu parametreleri istediğiniz şekilde adlandırabilirsiniz. Ama özellikle “event” parametresinin adını değiştirmemenizi öneririm. Çünkü bu da tıpkı “self” ve “main” gibi yerleşmiş bir ifadedir...

Böylelikle Pygtk kullanılarak yazılmış en basit uygulamanın nasıl olması gerektiğini öğrenmiş olduk. Basit bir Pygtk uygulamasında yer alan temel aşamaları şöyle bir gözden geçirelim:

Öncelikle gerekli modülleri içe aktarıyoruz:

```

import pygtk
pygtk.require20()
import gtk

```

Burada içe aktardığımız pygtk modülünü, sürüm kontrolü için kullandık. “pygtk.require20()” satırı, sistemimizde birden fazla pygtk sürümü olması durumunda, programımızı bu sürümlerden hangisiyle çalıştırmak istediğimizi belirlememizi sağlıyor. Modern Pygtk uygulamaları “2.x” sürümleri kullanılarak yazılır. Dolayısıyla biz de bu satır yardımıyla, kullanılacak sürümü “2.x” olarak belirliyoruz. Son olarak da gtk modülünü içe aktardık. “gtk”, programımızda asıl işi yapan modüldür.

Ardından program kodlarımızı yerleştirmek için bir sınıf tanımlıyoruz. Bu sınıf, Python’daki yeni tipte sınıfların gerektirdiği şekilde mutlaka bir başka sınıftan miras almak durumunda... Bizim burada özel olarak miras almak istediğimiz bir sınıf bulunmadığı için, “object” sınıfını miras alarak bu aşamayı geçiyoruz:

```

class Pencere(object):

```

Tanımladığımız sınıfın bir “__init__” fonksiyonu olması gerekiyor. Programımız ilk çalışmaya başladığı anda sahip olacağı özellikleri bu __init__ fonksiyonu içinde belirliyoruz. Mesela pencereyi ve öteki araçları bu fonksiyon içinde tanımlayabiliriz. Henüz pencerenin kendisi

dışında bir araç belirlemediğimiz için buraya şimdilik sadece boş pencereyi oluşturacak kodları yazıyoruz:

```
def __init__(self):
    self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
    self.pencere.connect("delete_event", self.kapat)
    self.pencere.show()
```

Burada pencere oluşturmak için gtk modülünün Window() fonksiyonundan yararlandığımızı dikkat edin. Ayrıca oluşturduğumuz pencerenin, bir pencerenin sahip olması gereken bütün özellikleri barındıracağından emin olmak için yine gtk modülünün WINDOW_TOPLEVEL niteliğini de parametre olarak gtk.Window fonksiyonu içinde belirtiyoruz.

Ardından, yazdığımız programın düzgün kapanmasını sağlamak için pencerenin çarpı tuşuna bir görev atıyoruz. Burada connect() adlı fonksiyondan faydalanarak, çarpı işaretini, daha sonra tanımlayacağımız kapat() adlı fonksiyona bağlıyoruz. Pygtk'da bu "çarpı tuşuna basma hareketi" "delete_event" olarak adlandırılır... Ayrıca bu "delete_event", teknik dille ifade etmek gerekirse, bir "olay"dır (event).

Sonraki satırda gördüğümüz "self.pencere.show()" ifadesi de oldukça önemlidir. Eğer bu satırı yazmazsak, oluşturduğumuz pencere ekranda görünmez... show() metodunu kullanarak penceremizi ekranda gösteriyoruz.

Sıra geldi programı kapatma fonksiyonumuzu tanımlamaya... Bu işlemi şu kodlar yardımıyla yapıyoruz:

```
def kapat(self, penar, event):
    gtk.main_quit()
```

Burada önemli nokta, kapat fonksiyonunun en az iki parametre alması... Biz birinci parametreyi "penar", ikinci parametreyi ise "event" olarak adlandırdık. Siz istediğiniz adları verebilirsiniz bu parametrelere. Ama en azından "event" kelimesini değiştirmemenizi tavsiye ederim. Burada "penar" parametresi, çarpı tuşuna basma hareketinin ait olduğu pencere aracına işaret ediyor (yani pencerenin kendisine...). "event" parametresi ise olayın kendisine atıfta bulunuyor (yani "delete_event" adlı olaya...). Programımızı sonlandırmak için ise "gtk" modülünün main_quit() adlı metodundan yararlanıyoruz.

Eğer "penar" ve "event" parametrelerinin tam olarak neye işaret ettiğini daha net bir şekilde görmek isterseniz fonksiyonunuzu şöyle yazabilirsiniz:

```
def kapat(self, penar, event):
    print type(penar)
    print type(event)
    gtk.main_quit()
```

Fonksiyonunuzu bu şekilde yazıp programı çalıştırdığınızda ve pencerenin çarpı simgesine bastığınızda komut satırında şu çıktıları göreceksiniz:

```
<type 'gtk.Window'>
<type 'gtk.gdk.Event'>
```

Demek ki gerçekten de "penar" parametresi "gtk.Window()" aracını (yani penceremizi); "event" parametresi ise "gtk.gdk.Event" adlı olayı (bizim örneğimizde "delete_event" adlı olayı) gösteriyor...

Programımızın asıl itici gücü olan fonksiyonu ise şöyle yazdık:

```
def main(self):
    gtk.main()
```

Bu fonksiyon, programımızın çalışmaya başladığını gösteriyor. Bir başka ifadeyle, bu fonksiyon yardımıyla programımıza hayat öpücüğü veriyoruz...

Son olarak, yazdığımız sınıfı örnekliyoruz:

```
uyg = Pencere()
```

Eğer sınıf içinde belirttiğimiz `main()` fonksiyonunu bir motor olarak düşünürsek, bu motorun pervanesini de şu kodla döndürüyoruz:

```
uyg.main()
```

2.6 Pencereleme Simge Ekleme

Bildiğiniz gibi, pek çok programda pencerenin sol üst köşesinde programa ait ufak bir simge yer alır. Eğer programcı, yazdığı bir program için herhangi bir simge belirlemezse pencerenin sol üst köşesi boş kalacaktır. İşte şimdi biz o alanı nasıl kendi zevkimize göre doldurabileceğimizi öğreneceğiz.

Yazdığınız bir programın simgesini pencerenin sol üst köşesine yerleştirebilmek için Pygtk'de `set_icon_from_file()` adlı bir metottan yararlanabilirsiniz. Örneğin:

```
#!/usr/bin/env python
#-*-coding: utf-8-*-
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(500, 500)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_title("PROGRAM")
        self.pencere.set_icon_from_file("falanca.png")
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

`set_icon_from_file()` adlı metot bilgisayarınızda kayıtlı olan bir resmi alıp uygulama simgesi haline getirecektir. Eğer yukarıdaki kodları çalıştırdığınız halde pencerenin sol üst köşesinde herhangi bir simge görmüyorsanız, bunun sorumlusu kullandığınız sistem temasıdır. Mesela Ubuntu'da "Human-Clearlooks" adlı tema uygulama simgelerini göstermez. Ama eğer uygulamanızı simge durumuna küçültecek olursanız, kullandığınız resmi panel üzerinde görebilirsiniz... Ayrıca kullandığınız temayı değiştirerek de uygulamaların simgelerini görünür hale getirebilirsiniz.

2.7 Boyutlandırılmayan Pencere Oluşturmak

Bir Pygtk penceresi normal şartlar altında kullanıcı tarafından boyutlandırılabilir. Yani bir pencerenin sağını solunu çekerek bu pencereyi büyütebilir veya küçültebilirsiniz. Ama siz yazdığınız bazı programlarda kullanıcının pencereyi boyutlandırmasına müsaade etmek istemiyor olabilirsiniz. Mesela bir hesap makinesi uygulamasında programcı sabit bir pencere boyutu belirleyip bu pencere boyutunun değiştirilmesini engellemek isteyebilir... Eğer siz de yazdığınız programların boyutlandırılmasını istemiyorsanız Pygtk'deki `set_resizable()` adlı metottan yararlanabilirsiniz:

```
#!/usr/bin/env python
#-*-coding: utf-8-*-
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_size_request(500, 500)
        self.pencere.set_resizable(False)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_title("BOYUTLANDIRILAMAZ")
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Burada dikkatimizi çeken iki nokta var. Birincisi, daha önce pencereleri boyutlandırmak için kullandığımız `resize()` metodu yerine burada `set_size_request()` adlı başka bir metod kullandık. İkincisi, pencerenin elle boyutlandırılmasını engellemek için `set_resizable()` adlı bir metoda "False" parametresi verdik.

Bu kodlarda gördüğümüz `set_size_request()` metodu Pygtk'de pencere boyutlandırmanın başka bir yoludur. Burada bu metod yardımıyla penceremizin boyutunu 500x500 olarak ayarladık. `resize()` metodu ile `set_size_request()` metodu arasındaki farkı [Pencere Araçlarının Yerleşimi](#) adlı bölümde inceleyeceğiz.

Burada bizi asıl ilgilendiren, `set_resizable()` metodudur. Bu metodun varsayılan değeri "True"dir ve bu değere sahip olduğunda pencerelerimiz elle rahatlıkla boyutlandırılabilir. Ama eğer biz bu metodun değerini "False" yapacak olursak, kullanıcılarımız artık bu pencereyi boyutlandıramaz.

2.8 Saydam/Şeffaf Pencere Oluşturmak

Pygtk'de, oluşturduğunuz pencereleri saydam/şeffaf hale getirmek de mümkündür. Ancak bu işlev her işletim sisteminde aynı şekilde çalışmaz. Pygtk'de pencereleri şeffaflaştırmak için `set_opacity()` adlı bir metottan yararlanıyoruz. Mesela:

```
#!/usr/bin/env python
#-*-coding: utf-8-*-
import pygtk
```

```

pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_size_request(500, 500)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_title("ŞEFFAF")
        self.pencere.set_opacity(0.5)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada şeffaflaştırma işini yerine getiren satır “self.pencere.set_opacity(0.5)”tir. set_opacity() metodu “0” ve “1” arası bir değer alır. Bu metodun değeri 0’a yaklaştıkça pencere daha da şeffaflaşacak, 1’e yaklaştıkça ise şeffaflığı azalacaktır.

Dediğimiz gibi, bu metodun platformlara göre bazı kısıtlamaları vardır. Örneğin Windows üzerinde bu metod her zaman çalışır. Ancak GNU/Linux’ta bu metodun çalışabilmesi için masaüstü efektlerinin veya “Compiz Fusion”un açık olması gerekir. Eğer Ubuntu kullanıyorsanız, bu metodu çalıştırabilmek için *Sistem > Tercihler > Görünüm* yolunu takip ederek *Görsel efektler* sekmesinde “extra” ayarını seçmeniz gerekiyor...

Bununla ilgili ufak bir örnek daha yapalım:

```

import pygtk
pygtk.require20()
import gtk
import time

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title("DENEME")
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("destroy", gtk.main_quit)
        for i in range(11):
            while gtk.events_pending():
                gtk.main_iteration()

            self.pencere.set_opacity(i*0.1)
            self.pencere.show()
            time.sleep(0.1)

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Bu örnekte Pygtk penceresi yavaş yavaş şeffaftan görünüme geçecektir. Ancak set_opacity() metoduna ilişkin olarak yukarıda anlattıklarımızdan yola çıkarak, bu metoda çok fazla itibar

edilmemesi gerektiğini söyleyebiliriz.

Böylece bir konuyu daha bitirmiş olduk. Dilerseniz şimdi bu noktaya kadar öğrendiğimiz bütün metotları ve bu metotların görevlerini kısaca listeleyelim:

gtk.Window() Ana pencereyi oluşturmamızı sağlar. Şu parametreleri alır:

“gtk.WINDOW_TOPLEVEL”

“gtk.WINDOW_POPUP”

set_title() Pencere başlık ekler.

get_title() Pencere başlığının ne olduğunu söyler.

resize() Pencere boyutunu ayarlar.

show() Pencere araçlarını gösterir.

maximize() Pencere tam ekran haline getirir.

unmaximize() Pencere tam ekran konumundan çıkarır.

move() Pencere özel bir konuma taşımamızı sağlar.

set_position() Pencere belirli bir konuma taşımamızı sağlar. Bu metot şu parametrelerden birini alır:

“gtk.WIN_POS_NONE”,

“gtk.WIN_POS_CENTER”,

“gtk.WIN_POS_MOUSE”,

“gtk.WIN_POS_CENTER_ALWAYS”,

“gtk.WIN_POS_CENTER_ON_PARENT”

connect() Pencere ve pencere araçlarına görev atamamızı sağlar.

main() Programın çalışmasını sağlar.

main_quit() Programdan çıkılmasını sağlar.

set_icon_from_file() Pencereye uygulama simgesi eklememizi sağlar.

set_size_request() Pencere boyutlandırmanın başka bir yoludur.

set_resizable() Pencereyi elle boyutlandırılıp boyutlandırılmayacağını belirler.

set_opacity() Pencereyi şeffaflaştırmasını sağlar.

Gördüğümüz gibi, Pygtk ile ilgili epey şey öğrendik. Öğrendiğimiz bu bilgiler sayesinde artık Pygtk’yi kullanarak boş bir pencere oluşturabiliyoruz ve bu boş pencerenin bazı niteliklerini değiştirebiliyoruz. Ama boş bir pencere oluşturmanın pek keyifli bir iş olmadığı ortada... O yüzden isterseniz, oluşturduğumuz pencereye başka öğeleri nasıl ekleyeceğimizi de öğrenelim.

Pygtk’de Pencere Araçları (1. Bölüm)

Bu bölümde Pygtk’deki pencere araçlarını anlatmaya başlayacağız. Peki pencere aracı nedir? Hemen söyleyelim:

Bir programın arayüzü üzerinde görünen, düğme, etiket, kutucuk, menü vb. araçlara, “pencere aracı” (Window Gadget – Widget) adı verilir.

Pygtk; pencere araçlarının sayısı bakımından oldukça zengin bir arayüz takımıdır. Bir programda ihtiyaç duyabileceğimiz bütün pencere araçlarını Pygtk’de bulabiliriz...

Bu bölümde inceleyeceğimiz ilk pencere aracı “Label” olacak. O halde yola koyulalım...

3.1 “Label” Pencere Aracı

“Label”, kelime olarak “etiket” anlamına gelir. Oluşturduğumuz arayüz üzerinde kullanıcıya bir not veya mesaj göstermek istediğimizde “Label” adlı bu pencere aracından yararlanabiliriz. Pygtk’de bir “Label” oluşturmak istediğimizde şuna benzer bir kod yazıyoruz:

```
etiket = gtk.Label()
```

Elbette yukarıdaki satır, “Label” aracının iskeletidir yalnızca. Gerçek hayatta Label’i kullanırken, bu pencere aracına bazı başka şeyler de eklememiz gerekir. İsterseniz bu araçla ilgili basit bir örnek vererek başlayalım işe:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)
        self.pencere.show()

        self.etiket = gtk.Label("Merhaba")
```

```
self.pencere.add(self.etiket)
self.etiket.show()

def kapat(self, penar, event):
    gtk.main_quit()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğünüz gibi, Pygtk ile bir etiket oluşturmak hiç de zor değil. Burada gördüğünüz gibi, Pygtk'de bir etiket oluşturmak için üç aşamadan geçiyoruz:

```
self.etiket = gtk.Label("Merhaba")
self.pencere.add(self.etiket)
self.etiket.show()
```

Dilerseniz bu aşamaları tek tek inceleyelim:

Öncelikle şu kodu kullanarak bir etiket meydana getiriyoruz:

```
self.etiket = gtk.Label("Merhaba")
```

Burada etiketi oluşturan fonksiyon, "gtk.Label()". Bu fonksiyon içinde kullandığımız karakter dizisi ise bu etiketin içeriğini gösteriyor. Biz ilk etiketimiz için "Merhaba" karakter dizisini kullandık.

Etiketimizi oluşturduktan sonra yapmamız gereken bir şey daha var. Pygtk'de pencere araçlarını görünür hale getirebilmek için bu araçları pencere üzerine "eklememiz" lazım. Dolayısıyla yukarıdaki formül aracılığıyla etiketimizi oluşturduktan sonra yapmamız gereken ilk iş bu etiketi pencereye eklemek olacaktır. Eğer etiketimizi pencereye eklemezsek, programımızı çalıştırmak istediğimizde etiketimiz ekranda görünmeyecektir.

Etiketi pencereye ekleme işlemini şu kodla yapıyoruz:

```
self.pencere.add(self.etiket)
```

Oluşturduğumuz etiketi pencereye eklemek için add() adlı bir metottan yararlandığımızı dikkat edin. "add" kelimesi Türkçe'de "eklemek" anlamına gelir. Dolayısıyla yukarıdaki satır ile "self.etiket"i, "self.pencere"ye eklemiş (add) olduk...

Artık etiketimizi ekranda gösterebiliriz:

```
self.etiket.show()
```

Yukarıdaki kodların geri kalanının ne işe yaradığını biliyorsunuz. Bunları önceki bölümlerde görmüştük.

Dikkat ederseniz, yukarıdaki kodları kullanarak oluşturduğumuz pencere yine "200x200" piksel boyutunda. Yani pencerenin kendisi, pencere üzerine yerleştirdiğimiz etiketten büyük. Aslında tahminimce siz de görüntünün böyle olmasını isteyeceksiniz... Ama eğer oluşan pencerenin etiketin kapladığı alanla sınırlı olmasını isterseniz kodlarınızı şu şekilde yazabilirsiniz:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)

        self.etiket = gtk.Label("Merhaba")
        self.pencere.add(self.etiket)
        self.etiket.show()

        self.pencere.show()

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada yaptığımız tek değişiklik, “self.pencere.show()” satırının yerini değiştirmek oldu... Ana penceremizi etiketten sonra göstererek, penceremizin etiket boyutuyla sınırlı olmasını sağladık. Peki bu nasıl oluyor?

Pygtk ile boş bir pencere oluşturduğumuzda, oluşan boş pencerenin boyutu varsayılan olarak “200x200” pikseldir. Yukarıda verdiğimiz ilk kodlarda ana pencereyi etiketten önce gösterdiğimiz için, pencere oluştuğu sırada henüz ortada bir etiket olmadığından, oluşan pencere “200x200” piksel boyutunda oluyor. Ama ana pencereyi etiketten sonra gösterirsek, pencere oluştuğu sırada kendisinden önce oluşmuş bir etiket olacağı için, ana penceremiz etiketin sahip olduğu boyuta göre kendini ayarlıyor. Elbette önceki bölümlerde öğrendiğimiz şekilde, `resize()` veya `get_size_request()` metodunu kullanarak pencere boyutunu kendimiz de ayarlayabiliriz:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(400, 400)
        #dilerseniz resize() yerine set_size_request() de
        #kullanabilirsiniz:
        #self.pencere.set_size_request(400, 400)
        self.pencere.connect("delete_event", self.kapat)

        self.etiket = gtk.Label("Merhaba")
        self.pencere.add(self.etiket)
        self.etiket.show()

        self.pencere.show()

    def kapat(self, penar, event):

```

```
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğümüz gibi, `self.pencere`'yi kodların neresinde göstermiş olursak olalım, `resize()` (veya `get_size_request()`) metodu yardımıyla pencerenin sahip olması gereken boyutu belirlediğimiz için, penceremiz bizim istediğimiz boyutlara göre açılıyor...

Bu arada, dikkat ederseniz, yukarıdaki kodlarda herhangi bir Türkçe karakter kullanmamaya özen gösterdik. Elbette istersek Pygtk uygulamalarında Türkçe karakterleri de rahatlıkla gösterebiliriz. Ama bunun için öncelikle betiğimizin dil kodlamasını belirlememiz gerekir. Bunu nasıl yapacağımızı biliyorsunuz. Kodlarımızın en başına şu satırı ekleyeceğiz:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-
```

Dolayısıyla kodlarımızın son hali şöyle olacak:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(400, 400)
        self.pencere.connect("delete_event", self.kapat)

        self.etiket = gtk.Label("Merhaba Türkçe karakterler: şçöğüışçögüi")
        self.pencere.add(self.etiket)
        self.etiket.show()

        self.pencere.show()

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Pygtk ile bir etiket oluşturmayı öğrendik. Şimdi bu etiketi nasıl düzenleyebileceğimize bakalım. Örneğin bu etiketin yazıtipi, renk, boyut, vb. özelliklerini nasıl değiştirebileceğimizi inceleyeceğiz...

3.1.1 Etiket Metnini Değiştirmek

"Label" pencere aracının metnini belirledikten sonra, programın işleyişi sırasında bazı noktalarda bu metni değiştirmek gerekebilir. Bu pencere aracının metnini değiştirmek için

set_text() adlı metottan yararlanacağız:

```
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(400, 400)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.etiket = gtk.Label()
        self.pencere.add(self.etiket)
        self.etiket.show()
        self.pencere.show()

        self.etiket.set_text("İşlem Tamamlandı!")

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğümüz gibi, etiketi ilk oluşturduğumuzda bu etiket herhangi bir metin içermiyor. Etiketin metnini, set_text() adlı metottan yararlanarak, sonradan oluşturduk...

Elbette, yukarıdaki örnek pek işlevli değil... Eğer biz etiket metnini program çalıştığı sırada değiştirebilirsek tabii ki örneğimiz çok daha mantıklı olacaktır. Mesela, programımız ilk açıldığında "Lütfen bekleyiniz..." gibi bir yazı görsek... Ardından da bu yazı belli bir süre sonra "İşlem tamamlandı..." şekline dönüşse çok daha iyi bir örnek vermiş olacağız... Önce şöyle bir deneme yapalım:

```
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk
import time

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.resize(500, 500)
        self.pencere.set_title("deneme 1, 2, 3...")
        self.pencere.show()

        self.etiket = gtk.Label("Lütfen bekleyiniz...")
        self.pencere.add(self.etiket)
        self.etiket.show()

        time.sleep(1)
        self.etiket.set_text("İşlem tamamlandı!")
```

```
def main(self):
    gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Gördüğümüz gibi, bu kodlar istediğimiz işlemi yerine getiremedi. Programımız “1” saniye bekledikten sonra açıldı ve sadece “İşlem tamamlandı!” etiketini gösterdi. Halbuki bizim istediğimiz bu değil...

Pygtk’de bu tür işlemleri yapabilmek için kodlarımız arasına şu satırları eklememiz gerekir:

```
while gtk.events_pending():
    gtk.main_iteration()
```

Bu kodların görevi, arayüzün güncellenmesini sağlamaktır. Eğer arayüz üzerinde gerçekleşmeyi bekleyen olaylar varsa, bu kodlar sayesinde arayüz her olaydan sonra güncellenir. Böylelikle arayüz üzerinde belli bir süre zarfında gerçekleşen farklı olaylar, arayüzün güncellenmesi sayesinde kullanıcıya gösterilebilir. İsterseniz bunu bir örnekle somutlaştırmaya çalışalım:

```
##-*-coding:utf-8-*-
```

```
import pygtk
pygtk.require20()
import gtk
import time
```

```
class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("destroy", gtk.main_quit)
        self.pencere.resize(300, 300)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_title("deneme 1, 2, 3...")
        self.pencere.show()

        self.etiket = gtk.Label("Lütfen bekleyiniz...")
        self.pencere.add(self.etiket)
        self.etiket.show()

        while gtk.events_pending():
            gtk.main_iteration()

        time.sleep(1)
        self.etiket.set_text("İşlem tamamlandı!")

    def main(self):
        gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Bu defa, programımız istediğimiz işlemi gerçekleştirdi. Tam istediğimiz şekilde, programımız “Lütfen bekleyiniz...” etiketiyle açıldı ve “1” saniye sonra bu etiket “İşlem tamamlandı!” şekline dönüştü. Burada “while gtk.events...” kodlarının görevi, arayüzü takibe almak... Bu kodlar,

arayüz üzerinde gerçekleşmeyi bekleyen herhangi bir olay olup olmadığını sürekli olarak kontrol ediyor. Bizim örneğimizde, “1” saniye sonra değişmeyi bekleyen bir etiket olduğu için, `while gtk.events_pending(): gtk.main_iteration()` kodları arayüzün donup kalmasını engelliyor ve tıpkı olması gerektiği gibi, “1” saniye sonra etiketi değiştiriyor...

Yukarıdakine benzer bir başka örnek daha verelim. Mesela bu kez bir “sayaç” örneği hazırlayalım. Yazacağımız programda 1’den 10’a kadar aralıksız olarak sayalım:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import pygtk
pygtk.require(2.0)
import gtk
import time

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)

        self.etiket = gtk.Label("başlıyor...")
        self.pencere.add(self.etiket)
        self.pencere.show()
        self.etiket.show()

        for i in range(11):
            while gtk.events_pending():
                gtk.main_iteration()

            time.sleep(1)
            self.etiket.set_text("%s" %i)

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Eğer burada “`while gtk.events_pending(): gtk.main_iteration()`” kodlarını yazmazsak, programımızı çalıştırdığımızda arayüzümüz bir süre bekleyecek, alttan alta 10’a kadar saymayı bitirdikten sonra kendini gösterecektir...

3.1.2 Etiketlerin Rengini Değiştirmek

Bir “Label” aracı üzerindeki metnin rengini değiştirmek konusunda Pygtk bize çok büyük olanaklar sağlar. Metin rengini (ve metnin başka pek çok özelliğini) değiştirmek için kullanabileceğimiz, “Pango Markup Language” (Pango İşaretleme Dili) adlı ufak bir dil yapısı sunar bize Pygtk... “Pango İşaretleme Dili”, HTML’ye oldukça benzer bir dildir. Dolayısıyla temel HTML bilgisi, biraz sonra göstereceğimiz örnekleri anlamak bakımından epey yararlı olacaktır.

Bu bölümde, bir etiketteki metnin yazıtipi rengini nasıl değiştireceğimizi öğreneceğiz. İsterseniz bir önceki bölümde gördüğümüz örnek üzerinden gidelim. Örneğimiz şuydu:

```

#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk
import time

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)

        self.etiket = gtk.Label("başlıyor...")
        self.pencere.add(self.etiket)
        self.pencere.show()
        self.etiket.show()

        for i in range(11):
            while gtk.events_pending():
                gtk.main_iteration()

            time.sleep(1)
            self.etiket.set_text("%s" %i)

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Şimdi burada, arayüz üzerinde tek tek görünen sayıların rengini değiştireceğiz. Normalde bu sayılar siyah renkte, ama biz bu sayıları kırmızıya döndüreceğiz:

```

#-*-coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk
import time

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)

        self.etiket = gtk.Label("başlıyor...")
        self.pencere.add(self.etiket)
        self.pencere.show()
        self.etiket.show()

        for i in range(11):
            while gtk.events_pending():
                gtk.main_iteration()

```

```

        time.sleep(1)
        self.etiket.set_markup("<span foreground='red'>%s</span>"%i)

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Gördüğünüz gibi, programı çalıştırdığımızda, arayüz üzerindeki sayılar kırmızı renkte görünüyor. Burada kullandığımız kodun formülü şöyledir:

```
pencere_aracı.set_markup("<span foreground='renk'>metin</span>")
```

Bu arada, set_markup() metodunun, etiketin metnini ayarlama görevi gördüğüne de dikkat edin. Burada ayrıca set_text() gibi bir metot kullanmamıza gerek kalmadı.

İsterseniz bununla ilgili daha basit bir örnek yapalım:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.show()

        self.etiket = gtk.Label("Merhaba")
        self.etiket.set_markup("<span foreground='blue'>Merhaba</span>")
        self.pencere.add(self.etiket)
        self.etiket.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada, İngilizce'deki temel renk adlarını kullanabiliyoruz. Bu renk adlarından bazılarını vereyim:

```

red      = kırmızı
blue     = mavi
white    = beyaz
black    = siyah
green    = yeşil
brown    = kahverengi
orange   = kavuniçi
purple   = eflatun

```

Elbette sadece bu renklerle sınırlı değiliz. Eğer isterseniz <http://html-color-codes.info/> adresini ziyaret ederek, oradaki renk paletinden bir renk seçip programınız içinde kullanabilirsiniz. O adresteki renkleri şöyle kullanıyoruz:

Renk paletinden istediğiniz bir rengin üzerine tıklayın. Aşağıda “selected color code is:” yazan kutucuktaki kodun tamamını kopyalayın ve programınız içinde ilgili yere yapıştırın. Bir örnek verelim... Diyelim ki o renk paletinden bir renk seçtik ve “#610B5E” kodunu elde ettik. Şimdi bu kodun tamamını kopyalıyoruz ve gerekli yere yapıştırıyoruz:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.show()

        self.etiket = gtk.Label("Merhaba")
        self.etiket.set_markup("<span foreground='#610B5E'>Merhaba</span>")
        self.pencere.add(self.etiket)
        self.etiket.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğünüz gibi, seçtiğimiz renk başarıyla etiket rengi olarak ayarlandı...

Eğer etiket metnini yukarıda yaptığımız gibi üst üste iki kez yazmak istemiyorsanız, `get_text()` metodu yardımıyla otomatik olarak etiket metnini alabilirsiniz:

```
self.etiket.set_markup("<span foreground=<'#610B5E'>%s</span>%self.etiket.get_text())
```

Eğer istersek, bu yöntemi kullanarak daha karmaşık işler de yapabiliriz. Mesela bir etiketi farklı renklere boyamak gibi...

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.show()

        self.etiket = gtk.Label("Merhaba")
        self.renkli1 = self.etiket.get_text()[4:]
        self.renkli2 = self.etiket.get_text()[4:]

        self.pml = "\
<span foreground='#610B5E'>%s</span>\
<span foreground='#FF0000'>%s</span>"
```

```
self.etiket.set_markup(self.pml %(self.renkli1, self.renkli2))
self.pencere.add(self.etiket)
self.etiket.show()
```

```
def main(self):
    gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Burada yaptığımız şey, karakter dizisini dilimlemekten ibaret... Karakter dizisinin ilk dört harfi için bir renk, geri kalan harfleri için ise başka bir renk belirledik.

Bu konuyla ilgili son bir örnek daha vererek yolumuza devam edelim. Bu defa, birkaç örnek önce gösterdiğimiz “sayaç” programında tek sayıları bir renkle çift sayıları başka bir renkle göstermeyi deneyelim:

```
##-*-coding: utf-8 -*-
```

```
import pygtk
pygtk.require20()
import gtk
import time
```

```
class Uygulama(object):
```

```
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)
```

```
        self.etiket = gtk.Label("başlıyor...")
        self.pencere.add(self.etiket)
        self.pencere.show()
        self.etiket.show()
```

```
        for i in range(11):
            while gtk.events_pending():
                gtk.main_iteration()

                time.sleep(1)
                if i % 2 == 0:
                    self.etiket.set_markup("<span foreground='red'>%s</span>%i")
                else:
                    self.etiket.set_markup("<span foreground='blue'>%s</span>%i")
```

```
    def kapat(self, penar, event):
        gtk.main_quit()
```

```
    def main(self):
        gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Burada çok basit bir mantık kullandık. Eğer sayı 2'ye bölündüğünde kalan 0 ise (if i % 2 == 0) bu sayı çifttir. Eğer kalan 0 değilse o sayı tektir... Bu mantığa göre, 2'ye tam bölünen sayılar için “red”, yani kırmızı renk belirledik. Bunu şu kod yardımıyla yaptık:

```
if i % 2 == 0:
    self.etiket.set_markup("<span foreground='red'>%s</span>%i")
```

Ama eğer sayı 2'ye tam bölünmüyorsa, renk olarak "blue" yani maviyi seçtik:

```
else:
    self.etiket.set_markup("<span foreground='blue'>%s</span>%i")
```

Böylece arayüz üzerindeki sayı değiştikçe, bu sayının çift veya tek olmasına göre, her bir sayıyı iki farklı renkte gösterdik.

3.1.3 Etiketleri Yaslamak

Pygtk, oluşturduğumuz etiketleri sağa-sola yaslamamıza izin verir. Bunun için `set_justify()` adlı bir metottan yararlanabiliriz. Bu metodu şöyle kullanıyoruz:

```
etiket.set_justify(gtk.JUSTIFY_LEFT)
```

`set_justify()` metodu dört farklı parametre alabilir:

gtk.JUSTIFY_LEFT Metodun varsayılan değeri budur. Etiket metnini sola yaslar.

gtk.JUSTIFY_RIGHT Etiket metnini sağa yaslar.

gtk.JUSTIFY_CENTER Etiket metnini ortalar.

gtk.JUSTIFY_FILL Etiket metnini pencere üzerine düzgün olarak dağıtır.

Bu metot tek satırlık etiket metinleri üzerinde hiçbir etkiye sahip değildir. Dolayısıyla `set_justify()` metodunu kullanabilmek için elimizde birden fazla satırdan oluşan bir etiket olmalı. Buna ufak bir örnek verelim:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.metin = """
Adana, Türkiye'nin güneyinde,
Doğu Akdeniz'in kuzeyinde yer alan ildir.
İl merkezinin adı da Adana olup,
Seyhan, Yüreğir, Çukurova, Sarıçam ve
Karaisalı ilçelerinin birleşimiyle oluşur.
"""

        self.etiket = gtk.Label(self.metin)
        self.etiket.set_justify(gtk.JUSTIFY_CENTER)
        self.pencere.add(self.etiket)
        self.pencere.show_all()
```

```
def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()
```

3.1.4 Etiketleri Seçilebilir Hale Getirmek

Pygtk'de "Label" pencere aracını kullanarak bir etiket oluşturduğumuzda, varsayılan olan etiket metnini fare ile seçemeyiz. Dolayısıyla, yazdığımız programdaki etiket metnini kopyalama imkanımız da yoktur. Ancak Pygtk'deki `set_selectable()` adlı bir metodun değerini değiştirerek bu durumu tersine çevirebiliriz:

```
#!/usr/bin/env python
## -*- coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.metin = """
        Adana, Türkiye'nin güneyinde,
        Doğu Akdeniz'in kuzeyinde yer alan ildir.
        İl merkezinin adı da Adana olup,
        Seyhan, Yüreğir, Çukurova, Sarıçam ve
        Karaisalı ilçelerinin birleşimiyle oluşur.
        """

        self.etiket = gtk.Label(self.metin)
        self.etiket.set_selectable(True)
        self.pencere.add(self.etiket)
        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Burada `set_selectable()` metodunun değerini "True" yaptığımız için, etiket metni fare ile seçilebilir. Ayrıca bu etiket üzerinde farenin sağ tuşu da çalışacak, böylece etiket metnini kopyalayabileceksiniz.

3.2 "Button" Pencere Aracı

`gtk.Label()` adlı pencere aracıyla yeterince çalıştık. Öğrenme sürecimizin sıkıcı bir hal alması için derseniz yeni bir pencere aracı daha görelim. Hem böylece Pygtk'deki olanaklarımız da artmış olur...

Şimdi öğreneceğimiz pencere aracının adı “gtk.Button()”. “Button” kelimesi “düğme” anlamına gelir. Dolayısıyla bu pencere aracı programlarımıza düğme ekleme imkanı sağlayacak bize. “Button” pencere aracının kullanımı az çok “Label” pencere aracıninkine benzer. Bir Pygtk uygulamasında düğme oluşturmak için şu kodu kullanıyoruz:

```
dugme = gtk.Button("düğme")
```

Burada “gtk.Button()” fonksiyonu düğmenin kendisidir. Fonksiyona parametre olarak atadığımız “düğme” kelimesi ise, oluşturduğumuz düğme üzerinde görünecek yazıdır. İsterseniz bunu bir örnek içinde kullanalım:

```
##-coding:utf-8-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.dugme = gtk.Button("düğme")
        self.dugme.show()
        self.pencere.add(self.dugme)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Bu şekilde, üzerinde “düğme” yazan bir düğme oluşturmuş olduk... Yalnız burada düğmeyi oluşturma adımlarına çok dikkat ediyoruz. Düğme oluştururken şu adımları takip ettik:

Öncelikle “self.dugme = gtk.Button(“düğme”)” satırı yardımıyla pencere üzerinde bir düğme meydana getiriyoruz. Burada “gtk” modülünün Button() fonksiyonunu kullandık ve bu fonksiyona “self.dugme” adını verdik. Oluşturacağımız düğmenin üzerinde ne yazacağını ise Button() fonksiyonunun içinde bir parametre olarak belirliyoruz. Bizim örneğimizde oluşturduğumuz düğmenin üzerinde “düğme” kelimesi yazacak.

İkinci adım olarak, düğmemizi “gösteriyoruz”. Bu işlemi de “self.dugme.show()” satırı yardımıyla gerçekleştiriyoruz. Eğer bu satırı yazmazsak düğmemiz yine de oluşur, ama ekranda görünmez!

Atmamız gereken son adım ise bu düğmeyi ana pencereye “eklemek” olacaktır. İşte “self.pencere.add(self.dugme)” satırı bu işe yarar...

Dikkat ederseniz oluşturduğumuz düğme, pencerenin tamamını kaplıyor. Yani düğmemiz ana pencerenin içine, etrafında hiç boşluk bırakmayacak şekilde yerleşmiş... Eğer düğmemizin etrafında biraz boşluk bırakmak istersek set_border_width() adlı metottan yararlanacağız:

```
##-coding:utf-8-

import pygtk
pygtk.require20()
import gtk
```

```

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.dugme = gtk.Button("düğme")
        self.dugme.set_border_width(30)
        self.dugme.show()
        self.pencere.add(self.dugme)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada yazdığımız “self.dugme.set_border_width(30)” satırı, düğmenin dört tarafında 30 piksellik boş bir alan oluşturuyor. Aynı metodu, istersek ana pencereye de uygulayabiliriz. Bu metodu ana pencereye uyguladığımızda, pencere içinde belirlediğimiz piksel sayısı kadar bir alan oluşacaktır:

```

##-*-coding:utf-8-*

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_border_width(30)
        self.dugme = gtk.Button("düğme")
        self.dugme.show()
        self.pencere.add(self.dugme)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Temel olarak Pygtk’de bir düğmeyi nasıl oluşturacağımızı öğrendiğimize göre artık yolumuza devam edebiliriz. Bir düğme elbette üzerine tıklandığında bir işlevi yerine getirebilirse işe yarar bir hal alacaktır. Bir sonraki bölümde, düğmelerimize nasıl görev atayacağımızı öğreneceğiz.

3.2.1 Düğmelere Görev Atamak

Daha önceki derslerimizde ana pencereye nasıl görev atayacağımızı öğrenmiştik. Hatırlarsanız, Pygtk’yi kendi haline bıraktığımızda, ana penceredeki çarpı düğmesine tıklamamız aslında programın kapanmasını sağlamıyordu. Programımızı başarıyla kapatabilmek için ana pencerenin çarpı düğmesine bir görev atamamız gerekiyordu. Bu işlemi şöyle yapıyorduk:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Kodlarımızı bu şekilde yazdığımız zaman, pencerenin çarpı düğmesine tıkladığımızda programımız tamamen kapanacaktır. Biz yukarıdaki örnekte işimizi tek satırda hallettik. Yani `gtk.main_quit` kodunu doğrudan `self.pencere.connect()` fonksiyonuna bir parametre olarak atadık. Ama eğer istersek bu parametreyi ayrı bir fonksiyon olarak da yazabileceğimizi biliyoruz:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", self.kapat)
        self.pencere.show()

    def kapat(self, penar, event):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada `kapat()` adlı fonksiyonumuzun en az iki parametre aldığına dikkat ediyoruz (`penar` ve `event`). `penar` adlı parametre `kapat()` adlı fonksiyonun bağlanacağı `“pencere aracını”`; `event` adlı parametre ise çarpı işaretine tıkladığında gerçekleşecek olayı temsil ediyor. Bizim örneğimizde `penar` ana pencerenin kendisi oluyor. `event` ise `“delete_event”`...

Düğmelere görev atamak istediğimizde de buna benzer bir yöntem kullanıyoruz. Şimdi mesela bir düğme oluşturalım ve bu düğmeye tıkladığımızda programımız kapansın:

```

#-*-coding: utf-8-*-

import pygtk

```

```

pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_border_width(10)
        self.pencere.show()

        self.dugme = gtk.Button("Kapat")
        self.dugme.connect("clicked", gtk.main_quit)
        self.dugme.show()
        self.pencere.add(self.dugme)

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada basitçe "clicked" adlı sinyali "gtk.main_quit" adlı fonksiyona bağladık. "clicked", Pygtk'de fare ile tıklama hareketini ifade eder. Burada düğme üzerine fare ile tıklanıldığında gtk.main_quit() adlı fonksiyon faaliyete geçecektir. Bu fonksiyonun pencereyi kapatma vazifesi gördüğünü biliyoruz. Dolayısıyla ana pencere üzerindeki "Kapat" adlı düğmeye fare ile tıkladığımızda penceremiz (ve tabii ki programımız) kapanacaktır.

Burada gtk.main_quit() fonksiyonunu doğrudan connect() fonksiyonu içinde belirttik. Düğmeye tıklanıldığında daha karmaşık işlemler yapabilmemiz için çoğunlukla ayrı bir fonksiyon yazmamız gerekecektir. İsterseniz hemen buna bir örnek verelim:

```

#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_border_width(10)
        self.pencere.show()

        self.dugme = gtk.Button("Kapat")
        self.dugme.connect("clicked", self.kapat)
        self.dugme.show()

        self.pencere.add(self.dugme)

    def kapat(self, penar):
        gtk.main_quit()

    def main(self):
        gtk.main()

uyg = Uygulama()

```

```
uyg.main()
```

Burada dikkat etmemiz gereken birkaç nokta var: Kodlarımız içinde tanımladığımız `kapat()` adlı fonksiyon “self” dışında en az “1” parametre daha almak zorunda... `kapat()` fonksiyonu için belirlediğimiz “penar” adlı parametre, `kapat()` fonksiyonunun hangi pencere aracına bağlanacağını gösteriyor. Bizim örneğimizde bu fonksiyon “self.dugme” adlı pencere aracına bağlanıyor. Bu arada, buradaki “kapat” fonksiyonunu “self.pencere.connect(“delete_event”, self.kapat)” şeklinde ana pencereye atayamayacağımıza dikkat edin... Neden? Çünkü “delete_event” adlı olay (event), bağlı olduğu fonksiyonun en az “2” parametreye sahip olmasını gerektirir (“self” hariç). “clicked” adlı sinyal (signal) ise bağlı olduğu fonksiyonun en az “1” parametre almasını gerektirir... Dolayısıyla olayların (event) ve sinyallerin (signals) gerektirdiği parametre sayısı farklı olduğu için aynı fonksiyonu hem “olaylar” hem de “sinyaller” için kullanamıyoruz. Ama ümitsizliğe kapılmayın! “Kullanamıyoruz,” dediysek “hiç kullanamıyoruz,” demedik!... Bu sorunun üstesinden gelmenin de bir yolu yordamı var. “kapat” adlı fonksiyona üçüncü bir parametre daha ekleyip buna da varsayılan olarak “None” değeri verirsek sorunumuzu halletmiş oluruz...

```
##*-coding:utf-8*-  
  
import pygtk  
pygtk.require20()  
import gtk  
  
class Uygulama(object):  
    def __init__(self):  
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)  
        self.pencere.connect("delete_event", self.kapat)  
        self.pencere.set_border_width(10)  
        self.pencere.show()  
  
        self.dugme = gtk.Button("Kapat")  
        self.dugme.connect("clicked", self.kapat)  
        self.dugme.show()  
  
        self.pencere.add(self.dugme)  
  
    def kapat(self, penar, event=None):  
        gtk.main_quit()  
  
    def main(self):  
        gtk.main()  
  
uyg = Uygulama()  
uyg.main()
```

Böylece aynı `kapat()` fonksiyonunu hem pencereye hem de pencere aracına atamış olduk...

Burada özel olarak dikkat etmemiz gereken öge “clicked” adlı sinyaldir. İşin asıl kısmını gerçekleştirmemizi bu sinyal sağlar. Pygtk’da düğmeler temel olarak şu sinyalleri üretir:

clicked fare ile düğme üzerine tıklanıp bırakılma anı

pressed fare ile düğme üzerine basılma anı

released farenin tuşunun serbest bırakılma anı

enter imlecin düğmenin üzerine gelme anı

leave imlecini düğmeyi terketme anı

Bu sinyaller içinde "clicked" ve "released" adlı sinyaller, davranışları bakımından birbirine çok benzer. İki sinyal de, fare tuşuna basıldığında değil, farenin tuşu bırakıldığında üretilecektir. Ancak bu iki sinyal arasında önemli bir fark vardır. "clicked" sinyalini kullandığımızda, farenin tuşuna basıp, tuşu bırakmadan fare imlecini düğmeden uzaklaştırırsak ve bundan sonra farenin tuşunu serbest bırakırsak üzerine tıkladığımız düğme işlevini yerine getirmeyecektir. Ama "released" sinyalini kullandığımızda, farenin tuşuna basıp, tuşu bırakmadan fare imlecini düğmeden uzaklaştırırsak dahi, farenin tuşunu serbest bırakır bırakmaz ana pencere üzerindeki düğme işlevini yerine getirecektir... Bu ikisi arasındaki farkı daha net anlamak için iki sinyalin ayrı ayrı kullanıldığı kodlar yazıp, sonucu incelemenizi öneririm.

Yukarıda verdiğimiz beş farklı sinyal içinde işimize en çok yarayacak olan, "clicked" adlı sinyaldir. Öteki sinyallere, yazdığımız programlarda nispeten daha az ihtiyaç duyacağız...

Gelin şimdi bu öğrendiklerimizi pratiğe döken basit bir uygulama yazalım:

```
#!/usr/bin/env python3
#-*-coding:utf-8-*-

import pygtk
pygtk.require(2.0)
import gtk

class Uygulama(object):
    def __init__(self):
        #Her düğmeye tıklayışımızda farklı bir dil
        #adı gösteren bir uygulama yazıyoruz... Bunun için öncelikle dilleri
        #içeren bir liste oluşturuyoruz.
        self.diller = ["Türkçe", "İngilizce", "Almanca", "Fransızca", "İtalyanca"]

        #self.diller adlı listedeki öğelere erişmek için aşağıdaki
        #index değerini kullanacağız.
        self.index = 0

        #Uygulamamızın ana penceresini oluşturuyoruz.
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)

        #Uygulamamızın düzgün kapanması için "delete_event"i
        #"gtk.main_quit()" adlı fonksiyona bağlıyoruz.
        self.pencere.connect("delete_event", gtk.main_quit)

        #Penceremizin başlığını belirliyoruz.
        self.pencere.set_title("Örnek bir uygulama...")

        #Penceremizi ekranın ortasında konumlandırıyoruz.
        self.pencere.set_position(gtk.WIN_POS_CENTER)

        #Penceremizin boyutunu belirliyoruz.
        self.pencere.resize(100, 100)

        #Penceremizi gösteriyoruz.
        self.pencere.show()

        #Üzerinde "Başla!" yazan bir düğme...
        self.dugme = gtk.Button("Başla!")

        #Düğmemizin etrafında biraz boşluk bırakalım.
        self.dugme.set_border_width(25)
```

```

#Düğme üzerine basıldığında "self.say" adlı fonksiyon
#çalışmaya başlasın.
self.dugme.connect("pressed", self.say)

#Düğmemizi ana pencereye ekliyoruz.
self.pencere.add(self.dugme)

#Düğmemizi gösteriyoruz.
self.dugme.show()

def say(self, penar):
#düğmemizin etiketini değiştiriyoruz. index değeri her
#defasında bir sayı artarak ilerleyeceği için, bu index
#değerini self.diller adlı listenin öğelerine tek tek
#erişmek amacıyla kullanabiliriz.
self.dugme.set_label(self.diller[self.index])

#Düğmeye her basışta index değerini 1 sayı artıralım.
self.index += 1

#index değeri listenin sonuncu öğesine ulaşıncaya
#programımızın hata vermemesi için index değerini
#sıfırlıyoruz. Böylece liste öğelerini almaya
#tekrar en baştan başlayabiliyoruz.
if self.index == len(self.diller):
    self.index = 0

def main(self):
#Uygulamamızın dinamosu...
gtk.main()

uyg = Uygulama()
uyg.main()

```

Daha önce de söylediğimiz gibi, bir düğmeye görev atayacağımız zaman, atanacak işlevi içeren fonksiyonu yazarken bu fonksiyonun "self" ile birlikte en az 2 parametre içermesine dikkat ediyoruz. Esasında, yazacağımız bu fonksiyon üçüncü bir parametre daha içerebilir. Bu üçüncü parametreyi şöyle kullanabiliriz:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.sayac = 0

        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_border_width(10)
        self.pencere.set_position(gtk.WIN_POS_CENTER)

        self.dugme = gtk.Button("Tamam")
        self.pencere.add(self.dugme)
        self.dugme.connect("clicked", self.bas, "Tamam")

```

```

        self.dugme.show()
        self.pencere.show()

    def bas(self, penar, veri):
        self.sayac += 1
        print "%s düğmesine %s kez bastınız!" %(veri, self.sayac)

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada bas() adlı fonksiyona üçüncü bir parametre ekledik ve bu parametrenin adını “veri” koyduk. Elbette siz isterseniz buna başka bir ad da verebilirsiniz... Bu yeni parametreyi temsil eden veriyi de connect() fonksiyonu içinde belirttik (“Tamam”). Burada düğmenin adı olan “Tamam”ı doğrudan yazmak yerine şöyle bir şey de yapabilirsiniz tabii ki:

```
self.dugme.connect("clicked", self.bas, self.dugme.get_label())
```

Bu şekilde kodlarımızı daha esnek bir hale getirmiş oluyoruz...

Bu arada eğer komut satırında Türkçe karakterleri gösteremiyorsanız bas() fonksiyonu içindeki karakter dizisini şöyle yazdırabilirsiniz:

```
print u"%s düğmesine %s kez bastınız!" %(veri, self.sayac)
```

Burada karakter dizisini Unicode olarak belirlediğimize dikkat edin...

3.2.2 Düğmelerin Rengini Değiştirmek

Normal şartlar altında Pygtk uygulamalarında kullanacağınız düğmelerin rengini değiştirmeye ihtiyaç duyacağınızı zannetmiyorum. Ama eğer olur da farklı renkte düğmeler kullanmak istersiniz diye Pygtk size bu isteğinizi gerçekleştirmenizi sağlayacak araçlar da sunar...

Pygtk’de bir düğmenin rengini değiştirmek için modify_bg() adlı bir metottan yararlanacağız. Ancak bir düğmenin rengini değiştirebilmek için bundan fazlasını bilmeye ihtiyacımız var. Çünkü Pygtk’de düğme rengini belirlerken renk adını dümdüz kullanamazsınız. İsterseniz lafi uzatmadan bir örnek verelim:

```

#-*-coding: utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("destroy", gtk.main_quit)
        self.pencere.set_border_width(30)

        self.btn = gtk.Button("deneme")
        #şimdi düğmenin rengini değiştiriyoruz...
        self.btn.modify_bg(gtk.STATE_NORMAL, gtk.gdk.Color("red"))
        self.pencere.add(self.btn)

```

```
self.pencere.show_all()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğünüz gibi kırmızı renkte bir düğme oluşturduk... Burada ilk olarak `modify_bg()` adlı bir metod görüyoruz. Ancak bu metodun düğme rengini değiştirebilmesi için bazı parametrelere ihtiyacı var. Bu metodun aldığı ilk parametre `gtk.STATE_NORMAL`. Bu ifade bir pencere aracının durumunu gösteriyor. Pygtk'de pencere araçları beş farklı durumda bulunabilir:

gtk.STATE_NORMAL

Bir pencere aracının herhangi bir işlem yapılmıyorkenki durumu. Mesela bir program başlatıldığında, arayüz üzerindeki düğmeler `gtk.STATE_NORMAL` denen durumdadır. Yukarıda verdiğimiz örnekte düğmenin rengini, bu `gtk.STATE_NORMAL` denen durum için değiştiriyoruz. Eğer düğmenin üzerine fare ile gelecek olursanız düğme renginin eski haline döndüğünü göreceksiniz. Çünkü fare ile üzerine geldiğinizde düğmemiz artık `gtk.STATE_NORMAL` denen durumda olmayacaktır...

gtk.STATE_PRELIGHT

Bir pencere aracının üzerine fare ile gelindiğindeki durumu. Yukarıdaki örnekte düğme üzerine fare ile geldiğinizde düğmenin renginin eski haline döndüğünü gördünüz. İşte düğmenin üzerine fare ile geldiğiniz duruma `gtk.STATE_PRELIGHT` adı verilir. Yukarıdaki örnekte ilgili satırı şöyle değiştirin:

```
self.btn.modify_bg(gtk.STATE_PRELIGHT, gtk.gdk.Color("red"))
```

Şimdi düğmenin üzerine fare ile geldiğinizde düğmenin kırmızıya döndüğünü göreceksiniz. Çünkü bu defa düğme rengini `gtk.STATE_NORMAL` için değil, `gtk.STATE_PRELIGHT` için değiştirdik...

gtk.STATE_ACTIVE

Bir pencere aracının aktif olduğu duruma `gtk.STATE_ACTIVE` adı verilir. Mesela bir düğmeye bastığınız anda o düğme `gtk.STATE_ACTIVE` denen duruma geçer. Yukarıdaki örneğin ilgili satırını şöyle değiştirin:

```
self.btn.modify_bg(gtk.STATE_ACTIVE, gtk.gdk.Color("red"))
```

Şimdi düğmeye tıkladığınızda düğme kırmızıya dönecek, bıraktığınızda ise eski rengini alacaktır.

gtk.STATE_SELECTED

Bir pencere aracının seçili olduğu duruma `gtk.STATE_SELECTED` adı verilir. Henüz bir buna örnek olarak gösterebileceğimiz bir pencere aracı öğrenmedik, ama mesela bir liste kutusu içindeki seçili satır `gtk.STATE_SELECTED` durumuna örnek olarak verilebilir...

gtk.STATE_INSENSITIVE

Bir pencere aracının meşgul olduğu duruma `gtk.STATE_INSENSITIVE` adı verilir. Henüz bunun için de verecek bir örneğimiz yok. Ama zamanla olacak...

Düğmenin rengini değiştirmek için yazdığımız koddaki `modify_bg()` ve `gtk.STATE_ACTIVE` kısımlarını anladık. İncelememiz gereken sadece `gtk.gdk.Color("red"))` kısmı kaldı...

gtk.gdk.Color(), renkleri tutma görevi gören bir sınıftır. Temel renkler için İngilizce renk adlarından yararlanabiliriz. Veya daha önce de gördüğümüz gibi <http://html-color-codes.info/> adresindeki onaltılık renk kodlarını kullanabiliriz. Eğer isterseniz RGB renk kodlarını da gtk.gdk.Color() sınıfı içinde kullanabilirsiniz. Bu sınıf bu haliyle GNU/Linux üzerinde herhangi bir sorun çıkarmadan çalışacaktır. Ancak Windows üzerinde bu sınıfı çalıştırmak istediğinizde "TypeError: an integer is required" gibi bir hata alıyorsanız gtk.gdk.Color() sınıfı yerine gtk.gdk.color_parse() sınıfından da yararlanabilirsiniz:

```
self.btn.modify_bg(gtk.STATE_NORMAL, gtk.gdk.color_parse("red"))
```

Eğer Windows üzerinde bu kodlar da beklediğiniz sonucu vermiyorsa sorun kullandığınız temadadır. *Başlat > Programlar > GTK+ > Theme Selector* yolunu takip ederek mevcut GTK temasını değiştirmeyi deneyin. Eğer "Theme Selector" uygulaması Başlat menüsünde görünmüyorsa, "C\GTK\bin" klasörü içinde "gtkthemeselector.exe" adlı uygulamayı bulup buna çift tıklayarak da tema seçici programı başlatabilirsiniz...

Pencere Araçlarının Yerleşimi

Şimdiye kadar yazdığımız Pygtk uygulamalarında hep ana pencere üzerine tek bir pencere aracı yerleştirdik. Dikkat ettiyseniz hiç bir uygulamamızda aynı anda birden fazla pencere aracı kullanmadık. Çünkü şimdiye kadar öğrendiğimiz bilgiler tek bir pencere üzerine aynı anda birden fazla pencere aracı yerleştirmemize müsaade etmiyordu. Ama artık bu işi nasıl yapacağımızı öğrenmemizin vakti geldi... İşte biz de bu bölümde, Pygtk uygulamalarına nasıl aynı anda birden fazla pencere ekleyebileceğimizi öğreneceğiz.

Bu dediğimiz işlemi gerçekleştirebilmek için, Pygtk bize birkaç farklı seçenek sunuyor. Bu bölümde bize sunulan bu yöntemleri tek tek incelemeye çalışacağız.

4.1 Fixed() Haznesi

Yukarıdaki “giriş” bölümünde bahsettiğimiz gibi, Pygtk’de pencere araçlarını yerleştirmek için elimizde birkaç farklı seçenek vardır. Fixed() haznesini kullanmak da bu seçenekler biridir. Öncelikle “hazne”nin ne demek olduğunu anlamaya çalışalım... Programlama dillerinde “hazne”; “başka nesnelere içinde barındıran bir araç” olarak tarif edilebilir. “Hazne” kelimesinin İngilizce’deki karşılığı “container”...

Bu bölümde göreceğimiz “Fixed()” adlı araç da bir haznedir. Fixed() adlı bu hazne, Pygtk ile oluşturacağımız pencere araçlarını içinde barındırma vazifesi görür. İsterseniz hemen bunun nasıl kullanılacağına ilişkin bir örnek verelim:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_border_width(10)

        self.dugme1 = gtk.Button("Birinci")
```

```
self.dugme2 = gtk.Button("İkinci")
self.dugme3 = gtk.Button("Üçüncü")

self.hazne = gtk.Fixed()
self.pencere.add(self.hazne)

self.hazne.put(self.dugme1, 0, 0)
self.hazne.put(self.dugme2, 65, 0)
self.hazne.put(self.dugme3, 120, 0)

self.dugme1.show()
self.dugme2.show()
self.dugme3.show()
self.hazne.show()
self.pencere.show()

def main(self):
    gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Burada öncelikle “gtk.Fixed()” adlı sınıfı örnekliyoruz ve buna “self.hazne” adını veriyoruz. Zira “gtk.Fixed()” bir hazne olmasının yanısıra, GTK içindeki sınıflardan biridir aynı zamanda... Dolayısıyla bu sınıfı programımız içinde kullanabilmek için öncelikle bunu örneklememiz gerekiyor.

Örnekleme işlemini tamamladıktan sonra haznemizi ana pencereye ekliyoruz. Bu işlemi yapmak için “self.pencere.add(self.hazne)” kodunu kullandık. Haznemizi ana pencereye eklediğimize göre, pencere araçlarımızı hazne içine koymaya başlayabiliriz... Gördüğümüz gibi, önce pencereyi oluşturuyoruz, ardından haznemizi oluşturup bunu pencereye ekliyoruz, son olarak da pencere araçlarını hazneye yerleştiriyoruz. Pencere araçlarımızı hazneye yerleştirebilmek için put () adlı fonksiyondan yararlanacağız. Bu fonksiyonun formülü şöyledir:

```
hazne.put(pencere_aracı, x_düzlemindeki_konum, y_düzlemindeki_konum)
```

Bu formüle göre, “self.hazne.put(self.dugme1, 0, 0)” ifadesi şu anlama geliyor:

düğme1’i, x düzleminde (yani soldan sağa) 0. noktaya; y düzleminde ise (yani yukarıdan aşağıya) 0. noktaya gelecek şekilde hazne içine yerleştir!

Aynı mantığı kullanarak öteki düğmelerimizi de sırasıyla x düzleminde 65. ve 120. noktalara; y düzleminde ise 0. ve 0. noktalara gelecek şekilde hazne içine yerleştiriyoruz.

Sıra geldi bütün araçlarımızı göstermeye... Araçlarımızı gösterirken içten dışa doğru ilerlemenizi öneririm.

Buna göre ilk önce düğmeleri gösteriyoruz:

```
self.dugme1.show()
self.dugme2.show()
self.dugme3.show()
```

Daha sonra haznemizi gösteriyoruz:

```
self.hazne.show()
```

Son olarak da en dışta yer alan penceremizi gösteriyoruz:

```
self.pencere.show()
```

Bu arada size bir ipucu vereyim. Gördüğümüz gibi her aracı tek tek göstermek biraz yorucu ve sıkıcı oluyor. Pygtk bu konuda da bizi büyük bir dertten kurtaracak ve daha az benzin yakmamızı sağlayacak bir metot sunuyor bize. Bu şekilde bütün araçları tek tek göstermek yerine şöyle bir şey yazabiliriz:

```
self.pencere.show_all()
```

Yani kodlarımız şöyle görünecek:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_border_width(10)

        self.dugme1 = gtk.Button("Birinci")
        self.dugme2 = gtk.Button("İkinci")
        self.dugme3 = gtk.Button("Üçüncü")

        self.hazne = gtk.Fixed()
        self.pencere.add(self.hazne)

        self.hazne.put(self.dugme1, 0, 0)
        self.hazne.put(self.dugme2, 65, 0)
        self.hazne.put(self.dugme3, 120, 0)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

show_all() metodu pencere üzerindeki bütün araçları gösterecektir. Dolayısıyla bu metodu ana pencereye bağlamaya dikkat ediyoruz...

“Fixed()” adlı haznenin put() adlı fonksiyonundan başka bir de move() adlı bir fonksiyonu vardır. “move” kelimesinin Türkçe’de “hareket etmek” anlamına geldiğini söylersek, herhalde bu fonksiyonun işlevi az çok ortaya çıkacaktır... move() fonksiyonu, Fixed() haznesi içindeki pencere araçlarını hareket ettirmemizi, yani bunların yerlerini değiştirmemizi sağlar.

Bununla ilgili basit bir örnek verelim:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
```

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_border_width(10)

        self.dugme1 = gtk.Button("Tıkla!")
        self.dugme1.connect("clicked", self.oynat)

        self.dugme2 = gtk.Button("İkinci")
        self.dugme3 = gtk.Button("Üçüncü")

        self.hazne = gtk.Fixed()
        self.pencere.add(self.hazne)

        self.hazne.put(self.dugme1, 0, 0)
        self.hazne.put(self.dugme2, 65, 0)
        self.hazne.put(self.dugme3, 120, 0)

        self.pencere.show_all()

    def oynat(self, penar):
        self.hazne.move(self.dugme2, 65, 50)
        self.hazne.move(self.dugme3, 120, 100)

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

move() fonksiyonunun nasıl kullanıldığını görüyorsunuz... Bu fonksiyon gtk.Fixed() sınıfına ait olduğu ve biz de bu sınıfı daha önceden “hazne” adıyla örneklediğimiz için, bu fonksiyonu “self.hazne.move()” şeklinde kullanıyoruz. Bu fonksiyonun birinci parametresi, hareket ettirilecek pencere aracıdır. İkinci ve üçüncü parametreler ise pencere aracının x ve y düzlemlerine göre hangi konumlara taşınacağını gösteren iki adet tamsayıdır. Bu programı çalıştırdığımızda, “Tıkla!” etiketli düğmeye bastığımız zaman “İkinci” ve “Üçüncü” etiketli düğmeler yer değiştirecektir...

Bu yeni bilgiyi kullanarak biraz daha “ilginç” şeyler de yapabiliriz. Mesela fareyle üzerine gelindiğinde etrafa kaçışan düğmeler yapabiliriz bu move() fonksiyonunu kullanarak...

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk
import random

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)

```

```

self.pencere.connect("delete_event", gtk.main_quit)
self.pencere.set_position(gtk.WIN_POS_CENTER)
self.pencere.resize(500, 500)

self.dugme1 = gtk.Button("Birinci")
self.dugme1.connect("enter", self.salla)

self.dugme2 = gtk.Button("İkinci")
self.dugme2.connect("enter", self.salla)

self.dugme3 = gtk.Button("Üçüncü")
self.dugme3.connect("enter", self.salla)

self.hazne = gtk.Fixed()
self.pencere.add(self.hazne)

self.hazne.put(self.dugme1, 0, 0)
self.hazne.put(self.dugme2, 65, 0)
self.hazne.put(self.dugme3, 120, 0)

self.pencere.show_all()

def salla(self, penar):
    self.hazne.move(penar, random.randint(0,450), random.randint(0, 450))

def main(self):
    gtk.main()

```

```

uyg = Uygulama()
uyg.main()

```

Buradaki `salla()` fonksiyonunu biraz açıklayalım:

`move()` fonksiyonunun hangi parametrelere ihtiyaç duyduğunu biliyoruz. Bu fonksiyonu kullanabilmek için ilk parametre olarak hareket ettirilecek pencere aracını yazmamız gerekiyor. İkinci ve üçüncü parametrelerde ise konum bilgilerini veriyoruz... Daha önceden, Pygtk'de sinyalleri bağladığımız fonksiyonların `self` hariç en az bir parametreye sahip olması gerektiğini söylemiştik. Bu parametre, sinyalin bağlanacağı pencere aracı olacak. Dolayısıyla `salla()` fonksiyonunun parametresinden ötürü zaten elimizde pencere aracının adı var... İşte biz bu pencere aracını `move()` fonksiyonu içinde de kullanacağız. Böylelikle her bir düğme için ayrı fonksiyon yazmamıza gerek kalmıyor... Elimizdeki bilgilere göre `move()` fonksiyonunu rahatlıkla oluşturabiliyoruz. `random` modülünün ürettiği rastgele sayılar sayesinde de düğmelerin üzerine fare ile her gelişimde düğmelerin her biri farklı konumlara kaçışıyor. Farenin düğme üzerine gelme anını "enter" ile gösterdiğimiz önceki bölümde söylemiştik. Dolayısıyla düğmelere görev atarken "clicked" yerine "enter" sinyalini yazmayı unutmuyoruz. Peki bu kaçışan düğmeleri yakalama ihtimalimiz var mı? Evet, çok düşük bir ihtimal de olsa var... Eğer `random` modülünün art arda ürettiği sayılar birbirine çok yakın olursa düğmeleri yakalayabiliriz...

Gördüğünüz gibi, `Fixed()` haznesi içine yerleştirdiğimiz düğmeler, üzerinde yazan etiketin boyutunu alıyor. Dolayısıyla "Tamam" ve "git" gibi iki farklı etikete sahip iki düğmenin boyutu birbirinden farklı olacaktır. Bir örnek verelim:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()

```

```

import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_border_width(10)

        self.etiket1 = gtk.Label("e.posta al!")
        self.etiket2 = gtk.Label("e.posta gönder!")
        self.etiket3 = gtk.Label("e.posta sil!")

        self.dugme1 = gtk.Button("Al!")
        self.dugme2 = gtk.Button("Gönder!")
        self.dugme3 = gtk.Button("Sil")

        self.hazne = gtk.Fixed()
        self.pencere.add(self.hazne)

        self.hazne.put(self.etiket1, 0, 10)
        self.hazne.put(self.etiket2, 0, 50)
        self.hazne.put(self.etiket3, 0, 90)

        self.hazne.put(self.dugme1, 110, 7)
        self.hazne.put(self.dugme2, 110, 47)
        self.hazne.put(self.dugme3, 110, 87)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Gördüğünüz gibi, buradaki düğmelerin her birinin boyutu birbirinden farklı... Eğer istediğiniz buysa sorun yok, ama eğer bütün düğmelerin aynı boyutta olmasını isterseniz `set_size_request()` adlı metodu kullanmanız gerekecek:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_border_width(10)

        self.etiket1 = gtk.Label("e.posta al!")
        self.etiket2 = gtk.Label("e.posta gönder!")
        self.etiket3 = gtk.Label("e.posta sil!")

```

```

self.dugme1 = gtk.Button("Al!")
self.dugme1.set_size_request(65, 25)

self.dugme2 = gtk.Button("Gönder!")
self.dugme2.set_size_request(65, 25)

self.dugme3 = gtk.Button("Sil")
self.dugme3.set_size_request(65, 25)

self.hazne = gtk.Fixed()
self.pencere.add(self.hazne)

self.hazne.put(self.etiket1, 0, 10)
self.hazne.put(self.etiket2, 0, 50)
self.hazne.put(self.etiket3, 0, 90)

self.hazne.put(self.dugme1, 110, 7)
self.hazne.put(self.dugme2, 110, 47)
self.hazne.put(self.dugme3, 110, 87)

self.pencere.show_all()

def main(self):
    gtk.main()

```

```

uyg = Uygulama()
uyg.main()

```

Burada `set_size_request()` metodunu kullanarak bütün düğmelerin boyutunu “65x25” piksel olarak ayarladık. Bu `set_size_request()` metodu yalnızca düğmelerle kullanılan bir metot değildir. Mesela bu metodu pencereleri boyutlandırmak için de kullanabiliriz:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_size_request(100, 100)

        self.pencere.show()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Hatırlarsanız, daha önceki derslerimizde pencereleri boyutlandırmak için `resize()` adlı başka bir metot kullanmıştık. Hem `resize()`, hem de `set_size_request()` yaygın olarak pencere boyutlandırma işlemleri için kullanılır. Ama aslında bu iki metodun birbirinden önemli bir farkı vardır. `resize()` metodunu kullanarak boyutlandırdığımız bir pencereyi daha

sonradan kenarlarından sürükleyip istediğimiz boyuta kadar küçültüp büyütebiliriz. Ama `set_size_request()` metodunu kullandığımızda, oluşan pencere ancak `set_size_request()` metodu içinde belirtilen boyutlar kadar küçültülebilir. Bu dediğimi tam olarak anlamak için hem `resize()` hem de `set_size_request()` metoduyla ayrı ayrı iki pencere oluşturup elle boyutlandırmayı denemeniz gerekiyor... Bunu yaptığınızda göreceksiniz ki, aslında `set_size_request()` metodu, bir pencerenin sahip olabileceği minimum boyutu belirlememizi sağlıyor...

Böylece Pygtk'de pencere araçlarını arayüz üzerine yerleştirmemizi sağlayan yöntemlerden birini görmüş olduk. Ancak bununla ilgili size bir iyi, bir de kötü haberim var... Önce iyi haberle başlayayım:

İyi haber, `Fixed()` haznesinin kullanımının karmaşık olmamasıdır. Bu hazneyi kullanarak işlerinizi rahatlıkla halledebilirsiniz. `Fixed()` haznesinin fazla bir özelliği olmadığı için, öteki araçlara kıyasla öğrenmesi daha kolaydır... Şimdi gelelim kötü habere:

Pygtk programlarında `Fixed()` haznesi hemen hemen hiç kullanılmaz!... Çünkü bu aracın zayıflıkları güçlü yanlarından çok daha fazladır ve kimi yerlerde programınızın arayüzünün berbat olmasına yol açabilir. `Fixed()` haznesi, pencere araçlarını arayüz üzerinde sabit noktalara yerleştirir. Dolayısıyla kullanıcılarınız sistemlerindeki temayı değiştirirlerse programınızın arayüzü üzerinde kaymalar meydana gelebilir. Ayrıca sizin programı yazdığınız sistemdeki yazıtiplerinin boyutu ile, programı kullanan kişilerin sistemdeki yazıtipi boyutları arasındaki farklılıklar düğme ve etiketlerin taşmasına neden olabilir. Mesela yukarıda verdiğimiz örneklerin Windows ve GNU/Linux'ta aynı şekilde görüneceğinin garantisi yoktur... Hatta aynı işletim sistemleri içinde dahi pencere araçlarının görünüşünde büyük farklılıklar ortaya çıkabilir. Bunun dışında, özellikle çok-dilli uygulamalar geliştiriyorsanız bu hazneyi kullanmaktan kaçınmanız gerekecektir. Çünkü bir etiket başka bir dile çevrildiğinde büyük ihtimalle kapladığı alan da değişecek ve pencere araçlarının üst üste binmesine yol açabilecektir. Aynı şekilde, eğer tasarladığınız bir uygulama sağdan sola doğru yazılan bir dile çevrilmek istenirse, kullandığınız `Fixed()` haznesinden dolayı bu gerçekleşemeyecektir. Çünkü bu haznenin "bidi" (çift-yönlü yazım) desteği bulunmuyor... Bunun dışında, bu hazne ile yerleştirilmiş bir öğede değişiklik yaptığınızda öteki öğelerin yerlerini de en baştan ayarlamamız gerekecektir.

4.2 Table() Pencere Aracı

Bu bölümde, `Fixed()` haznesinden çok daha yetenekli bir araçtan söz edeceğiz. Hatırlarsanız, sahip olduğu dezavantajlardan ötürü, yazdığımız uygulamalarda mümkün olduğunca `Fixed()` haznesini kullanmaktan kaçınmamız gerektiğini söylemiştik. `Fixed()` haznesinin tersine, bu bölümde öğreneceğimiz `Table()` adlı pencere aracı bir hayli esnek ve güçlüdür.

"Table" Türkçe'de "tablo" anlamına gelir. İsminden de anlaşılacağı gibi, bu araç bize düğmelerimizi, etiketlerimizi ve başka pencere araçlarımızı arayüz üstüne bir tablo şeklinde dizme imkanı sağlayacak. Burada "tablo" derken, satır ve sütünlara sahip bir düzenden söz ediyoruz. Oluşturmak istediğimiz pencere araçlarını, `Table()` aracının satır ve/veya sütunlarına yerleştireceğiz.

`Table()` pencere aracı şekil olarak ekran üzerinde şuna benzer bir yapı oluşturur:

0	1	2
0+-----+-----+		
1+-----+-----+		
2+-----+-----+		

Yukarıdaki çizim, 2 satır ve 2 sütundan oluşan, "2x2" boyutlarındaki bir tabloyu gösteriyor. İşte biz de oluşturacağımız pencere araçlarını bu satır ve sütunlara yerleştireceğiz. İsterseniz hemen bunun kullanımıyla ilgili ufak bir örnek verelim:

```
tablo = gtk.Table(2, 2)
```

Bu şekilde, 2 satır ve 2 sütundan oluşan bir tablo oluşturmuş olduk. Yani yukarıdaki çizime benzer bir tablo elde ettik.

Tabloya pencere aracı eklemek için `attach()` adlı metottan faydalanacağız. Bu metodu şöyle kullanıyoruz:

```
tablo.attach(pencere, sütun_başlangıcı, sütun_sonu, satır_başlangıcı, satır_sonu)
```

Burada "pencere", tabloya ekleyeceğimiz pencere aracının kendisi oluyor. Öteki parametreler ise pencere aracının yerleştirileceği tablo koordinatlarını gösteriyor. Dolayısıyla bir tabloya pencere aracı ekleyebilmek için, oluşturduğumuz tablonun satır ve sütun numaralarını göz önünde bulundurmamız gerekiyor. Gelin isterseniz bununla ilgili basit bir örnek yapalım.

Öncelikle şu kodları yazıyoruz:

```
##-coding:utf-8-  
  
import pygtk  
pygtk.require20()  
import gtk
```

Şimdi bir sınıf oluşturup temel bilgileri bu sınıf içine yerleştirelim:

```
class Uygulama(object):  
    def __init__(self):  
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)  
        self.pencere.connect("delete_event", gtk.main_quit)
```

Şimdi de tabloya yerleştireceğimiz düğmeleri oluşturalım:

```
self.dgm1 = gtk.Button("düğme1")  
self.dgm2 = gtk.Button("düğme2")  
self.dgm3 = gtk.Button("düğme3")  
self.dgm4 = gtk.Button("düğme4")
```

Buraya kadar olan kısmı topluca görelim:

```
##-coding:utf-8-  
  
import pygtk  
pygtk.require20()  
import gtk  
  
class Uygulama(object):  
    def __init__(self):  
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)  
        self.pencere.connect("delete_event", gtk.main_quit)  
  
        self.dgm1 = gtk.Button("düğme1")  
        self.dgm2 = gtk.Button("düğme2")
```

```
self.dgm3 = gtk.Button("düğme3")
self.dgm4 = gtk.Button("düğme4")
```

Bu kısımda bilmediğimiz herhangi bir şey yok. Burada gördüğümüz her şeyi önceki konulardan biliyoruz. Şimdi yolumuza devam edelim. Öncelikle 2x2 boyutlarında bir tablo oluşturuyoruz:

```
self.tablo = gtk.Table(2, 2)
```

İlk düğmemizi bu tablonun sol üst köşesine yerleştirelim:

```
self.tablo.attach(self.dgm1, 0, 1, 0, 1)
```

Gördüğünüz gibi, attach() adlı fonksiyona parametre olarak ilk önce tabloya yerleştireceğimiz pencere aracını yazdık (self.dgm1). Öteki sayılar ise düğmenin tam koordinatlarını gösteriyor. Amacımız bu düğmeyi tablonun sol üst köşesine yerleştirmek. Koordinatları yazarken yukarıda verdiğimiz çizimi göz önünde bulundurmanız işinizi epey kolaylaştıracaktır. O çizime göre düğmemiz tam olarak şöyle görünecek:

```
  0           1           2
0+-----+-----+
| DÜĞME 1 |           |
1+-----+-----+
|           |           |
2+-----+-----+
```

Gördüğünüz gibi, düğmemizin bulunduğu konum şöyle:

```
sütun_başlangıcı=0
sütun_sonu=1
satır_başlangıcı=0
satır_sonu=1
```

Şimdi ikinci düğmemizi yerleştirelim. İkinci düğmeyi tablonun sol alt köşesine yerleştireceğiz. Düğmemiz şöyle görünecek:

```
  0           1           2
0+-----+-----+
|           |           |
1+-----+-----+
| DÜĞME 2 |           |
2+-----+-----+
```

Düğmemizin konumu şöyle:

```
sütun_başlangıcı=0
sütun_sonu=1
satır_başlangıcı=1
satır_sonu=2
```

Buna göre attach() fonksiyonunu şu şekilde yazıyoruz:

```
self.tablo.attach(self.dgm2, 0, 1, 1, 2)
```

Sıra geldi üçüncü düğmemizi yerleştirmeye... Bunu tablonun sağ üst köşesine yerleştireceğiz:

```
  0          1          2
0+-----+-----+
|          | DÜĞME 3 |
1+-----+-----+
|          |          |
2+-----+-----+
```

Bu düğmemizin konumu da şöyle:

```
sütun_başlangıcı=1
sütun_sonu=2
satır_başlangıcı=0
satır_sonu=1
```

Buna göre attach() fonksiyonu şöyle olacak:

```
self.tablo.attach(self.dgm3, 1, 2, 0, 1)
```

Sıra geldi son düğmemizi yerleştirmeye... Bunu da tablonun sağ alt köşesine yerleştireceğiz:

```
  0          1          2
0+-----+-----+
|          |          |
1+-----+-----+
|          | DÜĞME 4 |
2+-----+-----+
```

Buna göre bu düğmemizin konumu da şöyle olmalı:

```
sütun_başlangıcı=1
sütun_sonu=2
satır_başlangıcı=1
satır_sonu=2
```

attach() fonksiyonumuzu yazıyoruz:

```
self.tablo.attach(self.dgm4, 1, 2, 1, 2)
```

Düğmelerimizi tablonun uygun yerlerine yerleştirdiğimize göre kodlarımızın son haline şöyle bir topluca bakalım:

```
#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)
```

```
self.dgm1 = gtk.Button("düğme1")
self.dgm2 = gtk.Button("düğme2")
self.dgm3 = gtk.Button("düğme3")
self.dgm4 = gtk.Button("düğme4")
self.tablo = gtk.Table(2, 2)

self.tablo.attach(self.dgm1, 0, 1, 0, 1)
self.tablo.attach(self.dgm2, 0, 1, 1, 2)
self.tablo.attach(self.dgm3, 1, 2, 0, 1)
self.tablo.attach(self.dgm4, 1, 2, 1, 2)
```

Bu noktadan sonra artık tablomuzu ana pencereye eklememiz gerekiyor. Bunu da şu komutla yapıyoruz:

```
self.pencere.add(self.tablo)
```

Pencere üzerindeki bütün öğeleri gösteriyoruz:

```
self.pencere.show_all()
```

main() fonksiyonumuzu yazıyoruz:

```
def main(self):
    gtk.main()
```

Son olarak da sınıfımızı örnekleyip main() fonksiyonunu çağırıyoruz:

```
uyg = Uygulama()
uyg.main()
```

Böylece şu kodları elde etmiş oluyoruz:

```
#!/usr/bin/env python3
coding: utf-8

import pygtk
pygtk.require(2.0)
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("düğme1")
        self.dgm2 = gtk.Button("düğme2")
        self.dgm3 = gtk.Button("düğme3")
        self.dgm4 = gtk.Button("düğme4")

        self.tablo = gtk.Table(2, 2)
        self.tablo.attach(self.dgm1, 0, 1, 0, 1)
        self.tablo.attach(self.dgm2, 0, 1, 1, 2)
        self.tablo.attach(self.dgm3, 1, 2, 0, 1)
        self.tablo.attach(self.dgm4, 1, 2, 1, 2)

        self.pencere.add(self.tablo)
```

```
self.pencere.show_all()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğümüz gibi, pencere araçlarını tabloya yerleştirmek o kadar da zor değil. Tek yapmamız gereken, araçları hangi koordinatlara yerleştirdiğimize dikkat etmek...

Bu kodlar yardımıyla arayüz üzerine yerleştirdiğimiz düğmeler, dikkat ederseniz, birbirlerine bitişik duruyorlar. Ama biz her zaman düğmelerimizin arayüz üzerinde tıkiş tıkiş durmasını istemeyebiliriz. Eğer düğmeler arasına mesafe koymak istersek, satırlar ve sütunlar için ayrı ayrı olmak üzere iki farklı metottan yararlanacağız. İsterseniz durumu sözle değil, örnekler üzerinden açıklamaya çalışalım:

```
#-*- coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("düğme1")
        self.dgm2 = gtk.Button("düğme2")
        self.dgm3 = gtk.Button("düğme3")
        self.dgm4 = gtk.Button("düğme4")

        self.tablo = gtk.Table(2, 2)
        self.tablo.set_row_spacings(10) #satırlar arasında boşluk bırakıyoruz
        self.tablo.set_col_spacings(10) #sütunlar arasında boşluk bırakıyoruz
        self.tablo.attach(self.dgm1, 0, 1, 0, 1)
        self.tablo.attach(self.dgm2, 0, 1, 1, 2)
        self.tablo.attach(self.dgm3, 1, 2, 0, 1)
        self.tablo.attach(self.dgm4, 1, 2, 1, 2)

        self.pencere.add(self.tablo)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Koyu harflerle gösterdiğimiz kodlar yardımıyla arayüz üzerinde bulunan tablomuzun satırları ve sütunları arasında 10'ar piksellik boşluklar oluşturduk. Böylece düğmelerimizi birbirinden ayırmış olduk...

Eğer bütün satırlar veya sütunlar üzerinde işlem yapmak yerine, sadece istediğimiz satırlar veya sütunlar üzerinde işlem yapmak istersek şu metotlardan yararlanacağız:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("düğme1")
        self.dgm2 = gtk.Button("düğme2")
        self.dgm3 = gtk.Button("düğme3")
        self.dgm4 = gtk.Button("düğme4")

        self.tablo = gtk.Table(2, 2)
        self.tablo.set_row_spacing(0, 10) #0. satıra 10 piksellik bir
                                                #boşluk yerleştiriyoruz.
        self.tablo.set_col_spacing(0, 10) #0. sütuna 10 piksellik bir
                                                #boşluk yerleştiriyoruz.

        self.tablo.attach(self.dgm1, 0, 1, 0, 1)
        self.tablo.attach(self.dgm2, 0, 1, 1, 2)
        self.tablo.attach(self.dgm3, 1, 2, 0, 1)
        self.tablo.attach(self.dgm4, 1, 2, 1, 2)

        self.pencere.add(self.tablo)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Bu kodların tam olarak nasıl çalıştığı ve ne işe yaradığı, 4x4'ten daha büyük tablolarda daha net olarak görünecektir...

Table() adlı aracı tanıtırken, bu aracı şu şekilde kullandığımızı söylemiştik:

```
tablo = gtk.Table(2, 2)
```

Daha önce de söylediğimiz gibi, burada "2x2" boyutunda bir tablo oluşturuyoruz. Yani oluşturduğumuz tabloda 2 adet satır, 2 adet de sütun bulunuyor... Esasında "gtk.Table()" fonksiyonunun tam formülü şöyledir:

```
gtk.Table(rows, columns, homogeneous)
```

Gördüğümüz gibi, bu araç aslında toplam üç parametre alıyor. "rows" parametresine satır sayısını, "columns" parametresine ise sütun sayısını yazıyoruz. Sadece "gtk.Table(2, 2)" yazmak yerine, "gtk.Table(rows=2, columns=2)" yazmak, parantez içindeki sayıların hangisinin satır, hangisinin sütun olduğunu akılda tutmak açısından daha kolay olacaktır. Dolayısıyla siz de gtk.Table() fonksiyonu yardımıyla bir tablo oluştururken, parantez içine yazdığınız sayıların neyin karşılığı olduğunu da belirtmeyi tercih edebilirsiniz...

Yukarıdaki formülde dikkatimizi çeken başka bir şey de elbette parametre sayısının üç olması... Şimdiye kadar “homogeneous” adlı bu üçüncü parametreyi görmemiştik. Ama şimdi “homogeneous” parametresinin ne olduğunu bize öğretecek bir örnek yapacağız:

“homogeneous” parametresinin temel görevi, birbirinden farklı boyutlardaki düğmelerin hepsini aynı boyuta getirmektir... Hemen şöyle bir örnek verelim:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("tamam")
        self.dgm2 = gtk.Button("çık")

        self.tablo = gtk.Table(rows=1, columns=2)
        self.tablo.attach(self.dgm1, 0, 1, 0, 1)
        self.tablo.attach(self.dgm2, 1, 2, 0, 1)

        self.pencere.add(self.tablo)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Burada “1” satır ve “2” sütundan oluşan bir tablo meydana getirdik. Bu tablonun sol üst köşesine bir düğmemizi, sağ üst köşesine ise öteki düğmemizi yerleştirdik. Yalnız dikkat ederseniz, düğme etiketlerinin uzunlukları birbirinden farklı olduğu için, düğmelerin boyutları da birbirinden farklı oluyor. Eğer düğmenin üzerindeki etiketin uzunluğu ne olursa olsun bütün düğmelerinizin aynı boyda olmasını isterseniz, “homogeneous” parametresi imdadınıza yetişecektir:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("tamam")
        self.dgm2 = gtk.Button("çık")

        self.tablo = gtk.Table(rows=1, columns=2, homogeneous=True)
```

```

self.tablo.attach(self.dgm1, 0, 1, 0, 1)
self.tablo.attach(self.dgm2, 1, 2, 0, 1)

self.pencere.add(self.tablo)

self.pencere.show_all()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()

```

Buradaki “homogeneous” parametresine “True” değerini verirsek tablo üzerindeki bütün düğmeler aynı boyda olacak; bu parametrenin değerini “False” yaptığımızda ise her bir düğme farklı boyutta olacaktır.. Eğer bu parametreyi hiç yazmazsak, Pygtk sanki bu parametrenin değeri “False” olarak belirlenmiş gibi davranacaktır..

Aslında bu parametre doğrudan düğmenin boyutuna müdahale etmiyor. Bu parametrenin müdahale ettiği şey tablonun satır ve sütunlarının boyutu. “homogeneous” parametresini “True” olarak ayarladığımızda, tablo üzerindeki satır ve sütunlar, tablo üzerindeki en geniş düğmenin boyutunu alıyor. Yani satır ve sütunlar, boyutları açısından tablo üzerine “homojen” olarak dağılıyor. Satır ve sütunlar tablo üzerine “homojen” olarak dağılıncaya da düğmelerimiz, içinde buldukları satır veya sütunu kaplayacak şekilde genişliyor. Bu parametrenin değerini “False” olarak ayarladığımızda veya hiç yazmadığımızda ise tablo üzerindeki satır ve sütunlar, içinde barındırdıkları düğmelerin boyutunu alıyor. Eğer düğmenin etiketi kısa ise düğmenin kendisi de kısa olacağı için, bu düğmenin bulunduğu satır veya sütun da dar oluyor.. Bu açıklama biraz karışık gelmiş olabilir. gtk.Table() aracının nerede nasıl davrandığını anlamamanın en iyi yolu kendi kendinize denemeler yapmaktır. Mesela biz şimdi şöyle bir örnek verelim:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("tamam")
        self.dgm2 = gtk.Button("çık")

        self.tablo = gtk.Table(rows=4, columns=4, homogeneous=True)
        self.tablo.attach(self.dgm1, 2, 3, 3, 4)
        self.tablo.attach(self.dgm2, 3, 4, 3, 4)

        self.pencere.add(self.tablo)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Bu örnek, “homogeneous” adlı parametrenin alttan alta neler karıştırdığını açık bir şekilde ortaya koyuyor. Burada “homogeneous” parametresinin değeri “True” olduğu için, tablo üzerindeki bütün satır ve sütunlar, en geniş düğmenin (yani “tamam” etiketli düğmenin) boyutunu aldı. “çık” etiketli düğme, “tamam” etiketli düğmeye kıyasla daha kısa olduğu için, “homogeneous=True” ifadesinin etkisiyle, içinde bulunduğu satırın tamamını kaplayacak şekilde genişledi. Dikkat ederseniz, tablomuz “4x4” boyutunda. Yani tablo üzerinde toplam 4 satır ve 4 sütun var. Ama arayüz üzerinde sadece iki adet düğme yer alıyor. Biz bu iki düğmeyi tablonun sağ alt köşelerine yerleştirdik. Bu arada, boş satır ve sütunların da, boş olmalarına rağmen, tablo üzerindeki en geniş düğmenin boyutunu aldığına dikkat edin... Burada “homogeneous” parametresinin değerini “False” olarak ayarlarsanız, boş satır ve sütunların yokluk derecesine kadar daraldığını göreceksiniz...

Dediğimiz gibi, bir tablodaki satır ve sütunları “homojen” olarak dağıttığımızda, düğmeler, içinde yer aldıkları satır ve sütunların içini tamamen kaplayacak şekilde genişleyecektir. “homojen” olarak dağıtılmamış bir tabloda ise düğmeler farklı boyutlarda olacaktır. Bir örnek vermek gerekirse:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("a")
        self.dgm2 = gtk.Button("abcd")
        self.dgm3 = gtk.Button("123456")
        self.dgm4 = gtk.Button("fdsfdfsdfsd")

        self.tablo = gtk.Table(rows=2, columns=3)
        self.tablo.attach(self.dgm1, 0, 1, 0, 1)
        self.tablo.attach(self.dgm2, 1, 2, 0, 1)
        self.tablo.attach(self.dgm3, 2, 3, 0, 1)
        self.tablo.attach(self.dgm4, 0, 3, 1, 2)

        self.pencere.add(self.tablo)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Gördüğümüz gibi bu kodlarda “homogeneous=True” gibi bir ifade kullanmadığımız için düğmeler birbirinden farklı boyutlara sahip. Bir de şuna bakalım:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk
```

```

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("a")
        self.dgm2 = gtk.Button("abcd")
        self.dgm3 = gtk.Button("123456")
        self.dgm4 = gtk.Button("fdsfdfsdfd")

        self.tablo = gtk.Table(rows=2, columns=3, homogeneous=True)
        self.tablo.attach(self.dgm1, 0, 1, 0, 1)
        self.tablo.attach(self.dgm2, 1, 2, 0, 1)
        self.tablo.attach(self.dgm3, 2, 3, 0, 1)
        self.tablo.attach(self.dgm4, 0, 3, 1, 2)

        self.pencere.add(self.tablo)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada “homogeneous” parametresinin değerini “True” olarak ayarladığımız için bütün düğmeler aynı boyuta geldi. Tablonun ikinci satırındaki “fdsfdfsdfd” etiketli düğmenin boyutunun öteki bütün düğmelerden farklı olması sizi şaşırtmasın. Zira o düğmenin boyutunu elimizle biz kendimiz ayarladık.. Dikkat ettiyseniz, o düğmeyi tabloya eklerken kullandığımız “self.tablo.attach(self.dgm4, 0, 3, 1, 2)” kodu o düğmeyi ikinci satırın tamamını kaplayacak şekilde ayarlıyor. (satur_başlangıcı=0, satur_sonu=3)

Şimdi bir deneme yapalım. Yukarıdaki kodlarla oluşturduğunuz pencereyi kenarından fare yardımıyla çekip büyütün. Gördüğünüz gibi, düğmeler de pencereyle birlikte büyüyüp küçülüyor. Bu durumun “homogeneous” parametresiyle ilgisi yoktur. Bu parametreyi “True” da yapsanız “False” da yapsanız, pencereyi fare ile büyütüp küçülttüğünüzde düğmeler de pencereyle beraber büyüyüp küçülecektir. Neden? Çünkü bu davranışı belirleyen parametre “homogeneous” değil, henüz öğrenmediğimiz başka bir parametredir. Şimdi bu davranışı tetikleyen parametrenin ne olduğuna bakalım.

Yukarıdaki şekilde oluşturduğumuz pencerelerde düğmeler pencereyle birlikte genişleyip daralır. Bunun nedeni, tabloya eklenen düğmelerin, varsayılan olarak “gtk.EXPAND” ve “gtk.FILL” adlı iki parametreye sahip olmasıdır.. Şimdi bu parametrelerin ne işe yaradığını göreceğiz. Şu kod parçasına bakalım:

```

self.tablo.attach(self.dgm1, 0, 1, 0, 1)
self.tablo.attach(self.dgm2, 1, 2, 0, 1)
self.tablo.attach(self.dgm3, 2, 3, 0, 1)
self.tablo.attach(self.dgm4, 0, 3, 1, 2)

```

Dört adet düğmeyi tabloya bu şekilde yerleştirdiğimizde aslında Pygtk bu kodları şu şekilde algılayacaktır:

```

self.tablo.attach(self.dgm1, 0, 1, 0, 1, gtk.EXPAND|gtk.FILL, gtk.EXPAND|gtk.FILL)
self.tablo.attach(self.dgm2, 1, 2, 0, 1, gtk.EXPAND|gtk.FILL, gtk.EXPAND|gtk.FILL)

```

```
self.tablo.attach(self.dgm3, 2, 3, 0, 1, gtk.EXPAND|gtk.FILL, gtk.EXPAND|gtk.FILL)
self.tablo.attach(self.dgm4, 0, 3, 1, 2, gtk.EXPAND|gtk.FILL, gtk.EXPAND|gtk.FILL)
```

Yukarıda verdiğimiz bu kodlardaki ilk satırı ele alalım:

```
self.tablo.attach(self.dgm1, 0, 1, 0, 1, gtk.EXPAND|gtk.FILL, gtk.EXPAND|gtk.FILL)
```

Buradaki ilk "gtk.EXPAND|gtk.FILL" komutu "xoptions" olarak anılır.. ikinci "gtk.EXPAND|gtk.FILL" ise "yoptions" olarak anılır.. "xoptions" kelimesi "x seçenekleri"; "yoptions" kelimesi ise "y seçenekleri" anlamına gelir. Burada "x seçenekleri" ifadesiyle kastedilen, bir düğmenin "x düzleminde kapsayacağı alana ilişkin seçenekler"dir. Aynı şekilde "y seçenekleri" ifadesiyle kastedilen şey de bir düğmenin "y düzleminde kapsayacağı alana ilişkin seçenekler"dir.. Bu bilgiye göre, yukarıdaki formülde yer alan ilk "gtk.EXPAND|gtk.FILL" ifadesi şu anlama geliyor:

pencere büyüyüp küçüldüğünde düğmemiz x düzleminde (yani soldan sağa doğru) hem genişlesin (expand), hem de içinde bulunduğu satır veya sütunu kapsasın (fill)

Aynı formüldeki ikinci "gtk.EXPAND|gtk.FILL" ifadesi ise şu anlama geliyor:

pencere büyüyüp küçüldüğünde düğmemiz y düzleminde (yani yukarıdan aşağıda doğru) hem genişlesin (expand), hem de içinde bulunduğu satırı veya sütunu kapsasın (fill)

Şimdi bu parametrelerin işlevini daha net anlayabilmek için ilk "gtk.EXPAND|gtk.FILL" ifadesindeki "gtk.FILL" kısmını silin. Yani kodunuz şöyle olsun:

```
self.tablo.attach(self.dgm1, 0, 1, 0, 1, gtk.EXPAND, gtk.EXPAND|gtk.FILL)
self.tablo.attach(self.dgm2, 1, 2, 0, 1, gtk.EXPAND, gtk.EXPAND|gtk.FILL)
self.tablo.attach(self.dgm3, 2, 3, 0, 1, gtk.EXPAND, gtk.EXPAND|gtk.FILL)
self.tablo.attach(self.dgm4, 0, 3, 1, 2, gtk.EXPAND, gtk.EXPAND|gtk.FILL)
```

Gördüğünüz gibi, "gtk.FILL" ifadesini kaldırıncaya arayüzün görüntüsü değişti. Artık düğmelerimiz, içinde buldukları satır ve sütunları kaplamıyor. Yalnız burada "xoptions" parametresinin değerinde yer alan "gtk.FILL" ifadesini sildiğimiz için bu değişiklik yalnızca "x düzlemini" etkiliyor. Bu parametrelerin tam olarak ne işe yaradığını anlamak için kendi kendinize çeşitli kombinasyonlar denemenizi öneririm...

"gtk.FILL" ve "gtk.EXPAND" dışında Pygtk'de "gtk.SHRIK" adlı bir parametre daha bulunur. Bu parametre ise düğmelerin "büzüşmesini", yani kesinlikle büyüyüp küçülmemesini sağlar. Şu örnek ne demek istediğimi açıklayacaktır:

```
##-*-coding:utf-8-*
```

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere= gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dgm1 = gtk.Button("a")
        self.dgm2 = gtk.Button("abcd")
        self.dgm3 = gtk.Button("123456")
        self.dgm4 = gtk.Button("fdsfdfsdfsd")
```

```
self.tablo = gtk.Table(rows=2, columns=3, homogeneous=True)
self.tablo.attach(self.dgm1, 0, 1, 0, 1, gtk.SHRINK, gtk.SHRINK)
self.tablo.attach(self.dgm2, 1, 2, 0, 1, gtk.SHRINK, gtk.SHRINK)
self.tablo.attach(self.dgm3, 2, 3, 0, 1, gtk.SHRINK, gtk.SHRINK)
self.tablo.attach(self.dgm4, 0, 3, 1, 2, gtk.SHRINK, gtk.SHRINK)

self.pencere.add(self.tablo)

self.pencere.show_all()

def main(self):
    gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Bu kodlarla oluşturduğumuz pencereyi ne şekilde boyutlandırırız boyutlandırılmaz, düğmeler oldukları yere çakılı vaziyette kalacaktır.

4.3 Box() Sınıfı

Pygtk'de pencere araçlarını pencere üzerine yerleştirebilmek için şimdiye kadar iki adet araç öğrendik. Bunlar pygtk içindeki Fixed() ve Table() adlı araçlardı. Bu iki araç arasında en çok kullanılanı Table()'dir. Fixed() haznesi her ne kadar oldukça kolay bir kullanıma sahip olsa da, Pygtk uygulamalarında hemen hemen hiç kullanılmaz. Bunun sebeplerini Fixed() haznesini işlerken anlatmıştık... Table() adlı aracın yanısıra, Pygtk'de pencere araçlarını pencere üzerine yerleştirmek için yaygın olarak kullanılan başka bir araç daha var. Bu bölümde işleyeceğimiz bu aracın genel adı "Box()". Ancak biz bu Box() sınıfını doğrudan kullanmayacağız. Bunun yerine, Box() adlı sınıftan türetilmiş "Hbox()" ve "Vbox()" adlı alt sınıflardan yararlanacağız. Ayrıca HBox() ve VBox()'tan farklı olarak ButtonBox() sınıfından türetilen "HButtonBox()" ve "VButtonBox()" adlı sınıflara da, aralarındaki benzerliklerden ötürü bu bölümde değineceğiz.

Bu bölümde "Hbox()", "Vbox()", "HButtonBox()" ve "VButtonBox()" sınıflarını sırayla inceleyeceğiz. O halde gelin hemen işe koyulalım...

4.4 HBox() Haznesi

HBox() da tıpkı Table() gibi bir haznedir. Yani bu araç, başka pencere araçlarını üzerinde taşıma vazifesi görür. İngilizce'de bu haznenin açılımı "Horizontal Box"tur. Bu ifade Türkçe'de "Yatay Kutu" anlamına gelir. Dolayısıyla bu hazne, içine yerleştirdiğimiz pencere araçlarını pencerenin yatay düzlemine yerleştirir. Bunun tam olarak ne demek olduğunu biraz sonra göreceğiz.

Pygtk'de bir yatay kutu oluşturabilmek için şu kodları kullanıyoruz:

```
kutu = gtk.HBox()
```

Gelin isterseniz bu kodu doğal ortamında görelim:

```
import pygtk
pygtk.require20()
import gtk
```

```

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = gtk.HBox()

        self.pencere.add(self.kutu)
        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Bu kodları çalıştırdığımızda pencere üzerinde pek bir değişiklik görmeyeceğiz. Yani sanki boş bir pencere oluşturmuş gibi olduk bu kodlar yardımıyla... Bu kodlar arasına yazdığımız "self.kutu = gtk.HBox()" satırının hiç bir etkisinin olmaması (aslında olmamış gibi görünmesi), oluşturduğumuz kutunun henüz boş olmasından kaynaklanıyor. Yani henüz oluşturduğumuz kutuya herhangi bir pencere aracı eklemediğimiz için, doğal olarak kutu boş görünüyor. Bu arada oluşturduğumuz kutuyu add() metodu yardımıyla ana penceremize eklemeyi unutmayoruz...

Şimdi bu kutuyu nasıl dolduracağımızı görelim:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dugme = gtk.Button("DENEME")

        self.kutu = gtk.HBox()
        self.kutu.pack_start(self.dugme)
        self.pencere.add(self.kutu)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada hayatımıza pack_start() adlı yeni bir metot giriyor... Bu metot, HBox() kutusu içine yerleştirdiğimiz pencere araçlarını yatay düzlem üzerinde soldan sağa doğru dizmemizi sağlar. İsterseniz birden fazla pencere aracını kullanarak daha net bir örnek vermeye çalışalım:

```

import pygtk
pygtk.require20()
import gtk

```

```

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dugme = gtk.Button("DENEME")
        self.dugme1 = gtk.Button("DENEME1")
        self.dugme2 = gtk.Button("DENEME2")
        self.dugme3 = gtk.Button("DENEME3")
        self.dugme4 = gtk.Button("DENEME4")

        self.kutu = gtk.HBox()

        self.kutu.pack_start(self.dugme)
        self.kutu.pack_start(self.dugme1)
        self.kutu.pack_start(self.dugme2)
        self.kutu.pack_start(self.dugme3)
        self.kutu.pack_start(self.dugme4)

        self.pencere.add(self.kutu)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Gördüğünüz gibi, her bir düğmemizi `pack_start()` metodu yardımıyla "HBox()" üzerine tek tek yerleştirdik. Burada düğmelerin soldan sağa doğru sıralandığına dikkat edin. Eğer bu düğmeleri sağdan sola doğru sıralamak istersek `pack_end()` metodundan yararlanacağız:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dugme = gtk.Button("DENEME")
        self.dugme1 = gtk.Button("DENEME1")
        self.dugme2 = gtk.Button("DENEME2")
        self.dugme3 = gtk.Button("DENEME3")
        self.dugme4 = gtk.Button("DENEME4")

        self.kutu = gtk.HBox()

        self.kutu.pack_end(self.dugme)
        self.kutu.pack_end(self.dugme1)
        self.kutu.pack_end(self.dugme2)
        self.kutu.pack_end(self.dugme3)
        self.kutu.pack_end(self.dugme4)

        self.pencere.add(self.kutu)

```

```

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada “DENEME” etiketli düğmenin en sağda yer aldığına, öteki düğmelerin de buna uygun şekilde sağdan sola doğru dizilmiş olduğuna dikkat edin...

HBox() adlı hazne iki parametre alır. Bunlardan biri “homogeneous”, ikincisi ise “spacing”dir. “homogeneous” parametresini Table() pencere aracını işlerken de görmüştük. Bu parametre burada da aynı görevde kullanılır. “spacing” parametresi ise pencere araçları arasında boşluk bırakabilmemizi sağlar:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dugme = gtk.Button("D")
        self.dugme2 = gtk.Button("DEN")
        self.dugme3 = gtk.Button("DENEM")
        self.dugme4 = gtk.Button("DENEME")

        self.kutu = gtk.HBox(homogeneous=False, spacing=10)

        self.kutu.pack_end(self.dugme)
        self.kutu.pack_end(self.dugme2)
        self.kutu.pack_end(self.dugme3)
        self.kutu.pack_end(self.dugme4)

        self.pencere.add(self.kutu)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada “homogeneous” parametresini “False” olarak ayarladığımız için, HBox() içindeki düğmelerin boyutları birbirinden farklı oldu. Eğer bu parametreyi “True” olarak ayarlarsak (varsayılan değer False’dır), HBox() içindeki bütün düğmeler, en uzun etiketli düğmenin boyutunu alacak, dolayısıyla bütün düğmeler aynı boyda olacaktır...

Aynı kodlar içinde gördüğümüz “spacing” adlı parametreye verdiğimiz değer sayesinde ise, düğmeler arasında 10 piksellik boşluklar oluşturduk. Böylece düğmelerimizi dip dibe yaşamaktan kurtarmış olduk...

Bunun dışında, pack_start() ve pack_end() metotları da bazı parametreler alabilir. Bu

metotların alabileceği parametre sayısı 3'tür:

expand: True veya False değeri alır. Düğmelerin genişleyip genişlemeyeceğini belirler.

fill: True veya False değeri alır. Düğmelerin, kutuyu kaplayıp kaplamayacaklarını belirler.

padding: Düğmeler arasındaki mesafeyi belirler.

Bununla ilgili bir örnek verelim:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dugme = gtk.Button("D")
        self.dugme2 = gtk.Button("DEN")
        self.dugme3 = gtk.Button("DENEM")
        self.dugme4 = gtk.Button("DENEME")

        self.kutu = gtk.HBox()

        self.kutu.pack_start(self.dugme, expand=False, fill=False, padding=10)
        self.kutu.pack_start(self.dugme2, expand=False, fill=False, padding=10)
        self.kutu.pack_start(self.dugme3, expand=False, fill=False, padding=10)
        self.kutu.pack_start(self.dugme4, expand=False, fill=False, padding=10)

        self.pencere.add(self.kutu)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Bu kodları çalıştırdığımızda, pencereyi ucundan çekip büyütünce düğmelerin ne tepki verdiğine dikkat edin...

Son bir not olarak şunu da söyleyelim. Eğer isterseniz `pack_start()` yerine sadece `add()` metodunu da kullanabilirsiniz:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.dugme = gtk.Button("D")
```

```

self.dugme2 = gtk.Button("DEN")
self.dugme3 = gtk.Button("DENEM")
self.dugme4 = gtk.Button("DENEME")

self.kutu = gtk.HBox()

self.kutu.add(self.dugme)
self.kutu.add(self.dugme2)
self.kutu.add(self.dugme3)
self.kutu.add(self.dugme4)

self.pencere.add(self.kutu)

self.pencere.show_all()

def main(self):
    gtk.main()

```

```

uyg = Uygulama()
uyg.main()

```

Ancak tabii ki `pack_start()` metodu, `add()` metoduna kıyasla daha gelişmiş bir araçtır. Eğer `add()` metodunun sağladığı özellikler bana yeter, diyorsanız yukarıdaki gibi bir yazım tarzı işlerinizi kolaylaştırabilir...

4.5 VBox() Haznesi

Bir önceki bölümde incelediğimiz `HBox()` haznesi, pencere araçlarını pencere üzerinde yatay olarak diziyordu. Şimdi öğreneceğimiz `VBox()` haznesi ise pencere araçlarını pencere üzerinde dikey olarak dizecek...

`VBox()` haznesini de tıpkı `HBox()` haznesine benzer şekilde oluşturuyoruz:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = gtk.VBox()

        self.pencere.add(self.kutu)
        self.pencere.show_all()

    def main(self):
        gtk.main()

```

```

uyg = Uygulama()
uyg.main()

```

Elbette, bu kodların işe yarar bir şey ortaya çıkarabilmesi için burada da `pack_start()` ve `pack_end()` metotlarından yararlanmamız gerekiyor:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = gtk.VBox()

        self.etiket1 = gtk.Button("Etiket 1")
        self.etiket2 = gtk.Button("Etiket 2")
        self.etiket3 = gtk.Button("Etiket 3")

        self.kutu.pack_start(self.etiket1)
        self.kutu.pack_start(self.etiket2)
        self.kutu.pack_start(self.etiket3)

        self.pencere.add(self.kutu)
        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Gördüğünüz gibi, “VBox()” haznesinin, “HBox()” haznesinden kullanım olarak hiçbir farkı yok. Bu ikisi arasındaki tek fark, HBox()’un düğmeleri soldan sağa (veya sağdan sola) dizmesi, VBox()’un ise düğmeleri yukarıdan aşağıya (veya aşağıdan yukarıya) dizmesidir... HBox() haznesini incelerken öğrendiğimiz her şeyi VBox() ile de kullanabilirsiniz...

Pygtk’de HBox() ve VBox() haznelerini, istediğiniz arayüz tasarımını oluşturabilmek için istediğiniz sayıda ve istediğiniz şekilde kullanabilirsiniz. Dilerseniz, bu konuyu kapatmadan önce bir örnek daha vererek HBox() ve VBox() haznelerinin esnekliğini görelim:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        #Normal bir şekilde penceremizi oluştup biçimlendiriyoruz...
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_title(u"Örnek bir Uygulama")
        self.pencere.connect("destroy", self.kapat)
        self.pencere.set_border_width(10)

        #Bir VBox() oluşturuyoruz. Bu, daha sonra içine düğmelerimizi
        #yerleştireceğimiz iki farklı HBox()’u barındıracak... Böylece
        #arayüz üzerinde HBox()’ların içindeki düğmeler alt alta görünecek
        self.anavbox = gtk.VBox(True, 0)
        self.pencere.add(self.anavbox)

```

```

#Üst sıradaki düğmeleri barındıracak olan HBox()'u oluşturuyoruz.
self.hbox1 = gtk.HBox(True, 0)

#Biraz önce oluşturduğumuz HBox()'u ana VBox() içine yerleştiriyoruz
self.anavbox.pack_start(self.hbox1)

#Şimdi de ikinci sıradaki düğmeleri barındıracak olan HBox()'u
#oluşturalım...
self.hbox2 = gtk.HBox(True, 0)

#Bunu da ana VBox() içine yerleştiriyoruz.
self.anavbox.pack_start(self.hbox2)

#Şimdi üst ve alt sırada yer alacak düğmeleri birer demet halinde
#tanımlayalım. Bu demetin her ögesi, kendi içinde iki öğeden oluşuyor.
#İlk öge düğmenin adı, ikinci öge ise düğmeye atanacak fonksiyonu temsil
#ediyor... Ben burada her sıradaki düğmeler için örnek olarak birer
#tane fonksiyon belirledim (self.kapat ve self.merhaba). Siz isterseniz
#her bir düğme için ayrı bir fonksiyon belirleyebilirsiniz...
ust_btnler = (("kapat", self.kapat),
              ("btn2", None),
              ("btn3", None),
              ("btn4", None))

alt_btnler = ("btn5", None),
             ("merhaba", self.merhaba),
             ("btn7", None),
             ("btn8", None))

#Üst ve alt sıradaki düğmeleri birer for döngüsü ile iki hamlede
#oluşturuyoruz. Burada "penar_olustur()" adlı bir fonksiyondan
#yardım alıyoruz. Bu fonksiyonu aşağıda tanımlayacağız.
for ust_oge in ust_btnler:
    self.penar_olustur(ust_oge[0], ust_oge[1], self.hbox1)

for alt_oge in alt_btnler:
    self.penar_olustur(alt_oge[0], alt_oge[1], self.hbox2)

#Bu da "penar_olustur()" adlı yardımcı fonksiyonumuz... Bu fonksiyon bizi
#aynı şeyleri tekrar tekrar yazmaktan kurtarıyor. Fonksiyonumuz "self"
#dışında üç parametre alıyor. İlk parametre olarak düğmenin adını veriyoruz.
#İkinci parametremiz düğmeye atanacak fonksiyonu gösteriyor. Üçüncü
#parametreyi ise düğmenin hangi HBox()'a yerleştirileceğini belirleyebilmek
#için kullanacağız...
def penar_olustur(self, metin, komut, penar):
    self.btn = gtk.Button(metin)
    #Yukarıda tanımladığımız demetler içinde ben "None" değerini
    #kullandığım için normalde Python bize TypeError hatası verecektir.
    #Bu hatanın programımızı çöktürmemesi için aşağıdaki try... except...
    #bloğunu yazıyoruz.
    try:
        self.btn.connect("clicked", komut)
    except TypeError:
        pass

    penar.pack_start(self.btn)
    self.pencere.show_all()

```

```

def kapat(self, penar):
    gtk.main_quit()

def merhaba(self, penar):
    print u"Merhaba Dünya"

def main(self):
    gtk.main()

```

```

uyg = Uygulama()
uyg.main()

```

Dediğimiz gibi, bu kodlarda üst ve alt satırda yer alacak düğmeleri iki farklı demet (tuple) içinde tanımladık. Biz burada demetler yerine sözlüklerden de yararlanabilirdik. Ama eğer sözlük kullanacak olursak, sözlükler sırasız veri tipleri oldukları için düğmeler ekranda bizim istediğimiz sırada görünmeyecektir... O yüzden yukarıdaki durum için en iyi yol, ikişer öğeli demetlerden oluşan iki ana demet tanımlamaktır.

4.6 HButtonBox() ve VButtonBox() Hazneleri

HButtonBox() ve VButtonBox() hazneleri HBox() ve VBox() haznelerine çok benzer. Bu yüzden "gtk.ButtonBox()" sınıfından türetilmiş bu hazneleri bu bölümde incelemeyi uygun bulduk. HButtonBox() ve VButtonBox() hazneleri HBox() ve VBox() haznelerine benzemekle birlikte, HButtonBox() veya VButtonBox() ile paketlenen düğmeler, pencere üzerine uygun bir şekilde dağılmalarını sağlayacak ayarları otomatik olarak elde ederler... Peki bu ne demek oluyor? Galiba en iyisi bu durumu bir örnekle açıklamak olacak:

```

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(300, 300)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = gtk.HButtonBox()

        self.etiket1 = gtk.Button("Etiket 1")
        self.etiket2 = gtk.Button("Etiket 2")

        self.kutu.add(self.etiket1)
        self.kutu.add(self.etiket2)

        self.pencere.add(self.kutu)
        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada gördüğünüz gibi, “HButtonBox()” haznesini oluşturma işlemi HBox()’u oluşturma işlemi aynısı. Ayrıca burada self.etiket1 ve self.etiket2’yi self.kutu’ya eklerken add() metodundan yararlanıyoruz. İsterseniz burada da pack_start() ve pack_end() metodlarını kullanabilirsiniz. Ama açıkçası HBoxButton() için add() gibi basit bir metod kullanmak yeterli olacaktır...

İsterseniz yukarıdaki kodlarda “self.kutu = gtk.HButtonBox()” satırını “self.kutu = gtk.HBox()” şeklinde değiştirip çalıştırın. HButtonBox() içine yerleştirilmiş düğmelerin HBox() içine yerleştirilmiş düğmelere göre farklı bir görünüme sahip olduğunu göreceksiniz. “gtk.HBox()” haznesini kullanarak bu şekilde bir görünüm elde etmek epey zordur. Dolayısıyla eğer istediğiniz görünüm buysa, gtk.HBox() ile cebelleşmek yerine doğrudan gtk.HButtonBox() haznesinden faydalanabilirsiniz...

Eğer HButtonBox() içine yerleştirdiğiniz düğmelerin konumuyla oynamak isterseniz, set_layout() adlı bir metod epey işinize yarayacaktır. Bu metodu şu şekilde kullanıyoruz:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(300, 300)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = gtk.HButtonBox()
        self.kutu.set_layout(gtk.BUTTONBOX_SPREAD)

        self.etiket1 = gtk.Button("Etiket 1")
        self.etiket2 = gtk.Button("Etiket 2")

        self.kutu.add(self.etiket1)
        self.kutu.add(self.etiket2)

        self.pencere.add(self.kutu)
        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Burada parantez içinde kullandığımız “gtk.BUTTONBOX_SPREAD” yerine kullanabileceğimiz birkaç farklı seçenek vardır elimizde:

- gtk.BUTTONBOX_SPREAD** : Düğmeleri ortalar
- gtk.BUTTONBOX_START** : Düğmeleri sola yaslar
- gtk.BUTTONBOX_END** : Düğmeleri sağa yaslar
- gtk.BUTTONBOX_EDGE** : Düğmeleri kenarlara yaslar

Bu parametreleri kullanarak, düğmeleri pencerenin farklı yerlerinde konumlandırabiliriz...

HButtonBox() dışında bir VButtonBox() diye bir şey vardır.. Tahmin edebileceğiniz gibi, bu hazne HButtonBox()’un aksine düğmeleri düşey düzleme yerleştiriyor. VButtonBox()’un kul-

lanımı ile HButtonBox()'un kullanımı aynıdır. Bunlar arasındaki tek fark bu iki haznenin düğmeleri pencere üzerinde konumlandığı yerdir.

Bununla ilgili küçük bir örnek yapalım:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.resize(300, 300)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = gtk.VButtonBox()
        self.kutu.set_layout(gtk.BUTTONBOX_END)

        self.etiket1 = gtk.Button("Etiket 1")
        self.etiket2 = gtk.Button("Etiket 2")

        self.kutu.add(self.etiket1)
        self.kutu.add(self.etiket2)

        self.pencere.add(self.kutu)
        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Bu kodlar, pencere üzerindeki düğmeleri pencerenin dip kısmına yerleştirecektir...

Buraya kadar Pygtk ile ilgili öğrendiklerimizi kullanarak şöyle bir örnek uygulama yazabiliriz:

```
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        #penceremizi oluşturalım
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)

        #penceremiz için bir başlık belirleyelim
        self.pencere.set_title("Emin misiniz?")

        #penceremizi boyutlandırılalım
        self.pencere.resize(150, 150)

        #penceremizin kenar çizgisi ile öğeler arasında
        #25 piksellik boşluk bırakalım.
        self.pencere.set_border_width(25)
```

```
#penceremizi ekranın tam ortasında konumlandıralım
self.pencere.set_position(gtk.WIN_POS_CENTER)

#penceremizin "x" düğmesine basıldığında penceremiz
#sağlıklı bir şekilde kapansın...
self.pencere.connect("delete_event", gtk.main_quit)

#etiketimizi belirliyoruz
self.soru = gtk.Label("Çıkmak istediğinizden emin misiniz?")

#"Vazgeç" düğmesini oluşturalım
self iptal = gtk.Button("Vazgeç")

#"Vazgeç" düğmesine tıklandığında "fonk" adlı
#fonksiyonumuz çalışsın. connect() metodunun üçüncü
#parametresine düğmemizin adını yazıyoruz.
self iptal.connect("clicked", self.fonk, "Vazgeç")

#"Çık" adlı düğmemiz...
self.terket = gtk.Button("Çık")

#Bu düğmeyi de "fonk" adlı fonksiyona bağlıyoruz.
#Yine üçüncü parametre olarak düğme adını yazıyoruz.
self.terket.connect("clicked", self.fonk, "Çık")

#dikey kutumuz... Öteki öğelerle arasına 10
#piksel boşluk bırakıyoruz.
self.vbox = gtk.VBox(spacing=10)

#dikey kutumuzu paketliyoruz. Bu kutu içinde "soru" adlı
#pencere aracımız yer alacak. Ayrıca "expand" ve "fill"
#parametrelerine "False" değerini vererek pencere aracımızın
#boyutlandırma işleminden etkilenmemesini sağlıyoruz.
self.vbox.pack_start(self.soru, expand=False, fill=False)

#Şimdi de bir adet HButtonBox() haznesi oluşturuyoruz...
self.hbut = gtk.HButtonBox()

#HButtonBox() haznesini doğruca VBox() haznesinin içine
#yerleştiriyoruz. Unutmayın, ana pencereye yalnızca tek
#bir pencere aracı eklenebilir!...
self.vbox.pack_start(self.hbut)

#HButtonBox() içine iptal ve terket düğmelerini yerleştirelim.
self.hbut.add(self iptal)
self.hbut.add(self.terket)

#HButtonBox() içindeki düğmeleri kutu içine ortalıyoruz
self.hbut.set_layout(gtk.BUTTONBOX_SPREAD)

#VBox() haznesini doğrudan pencere üzerine yerleştiriyoruz.
self.pencere.add(self.vbox)

#Son olarak bütün pencere araçlarını gösteriyoruz
self.pencere.show_all()

def fonk(self, penar, data):
    #Ana fonksiyonumuz... Burada kullanıcının hangi
```

```
#düğmeye bastığını komut satırında göstereceğiz
#Bunun için, connect() metoduna üçüncü parametre
#olarak eklediğimiz veriyi kullanacağız.
print "%s düğmesine basıldı"%data

#Eğer kullanıcı "Çık" adlı düğmeye basarsa
#komut satırına "Programdan çıkılıyor..."
#yazdıralım ve programı kapatalım
if data == "Çık":
    print "Programdan çıkılıyor..."
    gtk.main_quit()
#Eğer kullanıcının bastığı düğme "Çık" değilse
#"İşlem iptal edildi!..." çıktısı veriyoruz...
else:
    print "İşlem iptal edildi!..."

def main(self):
    gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Bu kodlar içinde özellikle dikkat etmemiz gereken kısım, iki farklı hazneyi (VBox ve HButtonBox) pencere üzerine nasıl yerleştirdiğimizdir. Gördüğümüz gibi, haznelerden birini ana pencere üzerine yerleştirirken (VBox), ötekini de öbür haznenin içine yerleştirdik (HButtonBox). Çünkü, ana pencere üzerine hem VBox'u, hem de HButtonBox'u aynı anda yerleştiremeyiz. Ana pencere sadece tek bir pencere aracı olabilir...

Bu arada kullandığınız konsol Türkçe karakterleri düzgün gösteremiyorsa yukarıdaki kodlarda yer alan karakter dizilerini unicode olarak tanımlamak işinize yarayabilir. Yani mesela:

```
print u"Programdan çıkılıyor..."
print u"İşlem iptal edildi!..."
```

...şeklinde karakter dizilerinin önüne "u" harfi getirebilirsiniz...

Pygtk'de Pencere Araçları (2. Bölüm)

Pygtk'de şimdiye kadar pencere araçları içinde `gtk.Label()` ve `gtk.Button()`'u gördük. Pygtk'de pencere araçlarından önce öğrenmemiz gereken konular olduğu için, pencere araçlarını incelemeye ara verip araya başka konular yerleştirmiştik.

Daha önce de dediğimiz gibi, Pygtk pencere araçları bakımından oldukça zengin bir arayüz takımıdır. Bu bölümde, Pygtk'deki pencere araçlarını incelemeye devam edeceğiz.

5.1 “Entry” Pencere Aracı

“Entry” pencere aracı, kullanıcıların veri girmesini sağlayan bir alandır. Bu pencere aracı dikdörtgen bir kutu şeklindedir. Python'da komut satırından çalışan programlar yazarken nasıl `raw_input()` ve `input()` fonksiyonlarından yararlanıyorsak, Pygtk'de de kullanıcıdan veri alabilmek için sıklıkla “Entry” pencere aracından faydalanacağız.

Pygtk'de Entry pencere aracı şöyle oluşturulur:

```
entry = gtk.Entry()
```

İsterseniz bunu bir de bağlamında görelim:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.entry = gtk.Entry()

        self.pencere.add(self.entry)
        self.pencere.show_all()
```

```
def main(self):  
    gtk.main()
```

```
uyg = Uygulama()  
uyg.main()
```

Gördüğümüz gibi, Entry() pencere aracını oluşturmak öteki pencere araçlarını oluşturmaktan hiç farklı değil. Bu kodlarda, ana pencere üzerinde tek bir pencere aracı olduğu için, bu pencere aracını (yani Entry'yi) doğrudan pencerenin kendisine ekledik. Ama uygulamada Entry aracını doğrudan ana pencere üzerine eklemeyeceğiz. Çünkü muhtemelen pencere üzerinde Entry ile birlikte başka pencere araçları da olacak. O yüzden daha önceki bölümlerde öğrendiğimiz konumlandırma araçlarını (gtk.Table, gtk.VBox, gtk.HBox gibi...) kullanmamız gerekecek.

Entry() pencere aracı, kullanıcının tek satırlık bir veri girmesine müsaade eder. Eğer kullanıcının girdiği veri kutunun uzunluğunu aşıyorsa ilk yazılanlar sola doğru kayacaktır. Entry() pencere aracı üzerinde farenin sağ tuşu da otomatik olarak çalışır. Dolayısıyla herhangi bir şey yapmamıza gerek kalmadan, "kes, kopyala, yapıştır ve sil" gibi özellikler emrimize amade olacaktır...

İsterseniz Entry() pencere aracının da kullanıldığı basit bir arayüz tasarlayarak kendi kendimize pratik yapalım:

```
#-*- coding: utf-8 -*
```

```
import pygtk  
pygtk.require20()  
import gtk
```

```
class Uygulama(object):  
    def __init__(self):  
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)  
        self.pencere.set_border_width(10)  
        self.pencere.set_position(gtk.WIN_POS_CENTER)  
        self.pencere.connect("delete_event", gtk.main_quit)  
  
        self.kull = gtk.Label("K. Adı: ")  
        self.paro = gtk.Label("Parola: ")  
        self.kentry = gtk.Entry()  
        self.pentry = gtk.Entry()  
  
        self.tablo = gtk.Table(4, 4)  
        self.tablo.attach(self.kull, 0, 1, 0, 1)  
        self.tablo.attach(self.paro, 0, 1, 1, 2)  
        self.tablo.attach(self.kentry, 1, 2, 0, 1)  
        self.tablo.attach(self.pentry, 1, 2, 1, 2)  
  
        self.pencere.add(self.tablo)  
  
        self.kutu = gtk.HButtonBox()  
        self.kutu.set_border_width(10)  
  
        self.tablo.attach(self.kutu, 0, 2, 2, 3)  
  
        self.btn = gtk.Button("Tamam")  
        self.kutu.pack_start(self.btn)  
  
        self.pencere.show_all()
```

```
def main(self):  
    gtk.main()
```

```
uyg = Uygulama()  
uyg.main()
```

Burada "gtk.Table()" ve gtk.HButtonBox()" araçlarını kullanarak, ana penceremiz üzerine beş farklı pencere aracı yerleştirdik. Pencere araçlarını içinde barındıran haznelerin yerleşimine özellikle dikkat edin. Bildiğiniz gibi, ana pencere üzerine yalnızca tek bir pencere aracı ekleyebiliyoruz. Bu yüzden ana pencere üzerine "Table()" pencere aracını yerleştirdik. Table()'nin üzerine ise "HButtonBox()" haznesini koyduk.

Elbette yukarıdaki örneği farklı araçları kullanarak ve farklı şekillerde de tasarlayabilirsiniz. Örneğin sadece Table() aracını kullanarak dahi yukarıdaki arayüzü elde edebilirsiniz:

```
#-*-coding:utf-8-*-
```

```
import pygtk  
pygtk.require20()  
import gtk
```

```
class Uygulama(object):  
    def __init__(self):  
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)  
        self.pencere.set_border_width(10)  
        self.pencere.set_position(gtk.WIN_POS_CENTER)  
        self.pencere.connect("delete_event", gtk.main_quit)
```

```
        self.kull = gtk.Label("K. Adı: ")  
        self.paro = gtk.Label("Parola: ")  
        self.kentry = gtk.Entry()  
        self.pentry = gtk.Entry()  
        self.btn = gtk.Button("Tamam")
```

```
        self.tablo = gtk.Table(4, 4)
```

```
#Burada "Tamam" düğmesinin denk geldiği satırı  
#öteki satırlardan biraz ayırıyoruz.  
        self.tablo.set_row_spacing(1, 10)
```

```
        self.tablo.attach(self.kull, 0, 1, 0, 1)  
        self.tablo.attach(self.paro, 0, 1, 1, 2)  
        self.tablo.attach(self.kentry, 1, 2, 0, 1)  
        self.tablo.attach(self.pentry, 1, 2, 1, 2)
```

```
#Burada ise "Tamam" düğmesinin bütün sütunu kaplamaması için  
#gtk.SHRINK parametresinden yararlanıyoruz.  
        self.tablo.attach(self.btn, 1, 2, 2, 3, gtk.SHRINK)
```

```
        self.pencere.add(self.tablo)
```

```
        self.pencere.show_all()
```

```
def main(self):  
    gtk.main()
```

```
uyg = Uygulama()
```

```
uyg.main()
```

Siz de bambaşka yöntemler kullanarak bu beş pencere aracını arayüz üzerine yerleştirebilirsiniz... Farklı yöntemler arasında size en kolay geleni benimsemekte özgürsünüz...

5.1.1 “Entry” Pencere Aracından Veri Almak

Entry() pencere aracının ne olduğunu ve ne işe yaradığını anlatırken, bu pencere aracını kullanıcıdan veri almak için kullanabileceğimizi söylemiştik. Kullanıcı, kendisinden istediğimiz verileri Entry() ile oluşturduğumuz kutuların içine yazıyor. Burada soru şu: Kullanıcı verileri kutulara giriyor girmesine, ama biz girilen bu bilgileri nasıl alıp kullanacağız? İşte bu bölümde bu sorunun cevabını bulmaya çalışacağız.

Pygtk’de, Entry() aracına kullanıcı tarafından girilen verileri alabilmek için “get_text()” adlı metottan yararlanacağız. Ufak bir örnekle başlayalım:

```
#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class Arayuz(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(20)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.giris = gtk.Entry()
        self.dgm = gtk.Button("Göster")
        self.dgm.connect("clicked", self.goster)

        self.hbox = gtk.HBox()
        self.pencere.add(self.hbox)
        self.hbox.pack_start(self.giris)
        self.hbox.pack_start(self.dgm)

        self.pencere.show_all()

    def goster(self, penar):
        self.veri = self.giris.get_text()
        print self.veri

    def main(self):
        gtk.main()

gui = Arayuz()
gui.main()
```

Bu uygulamada, Entry() pencere aracına yazdığımız herhangi bir veri, “Göster” adlı düğmeye basılarak komut satırında gösterilebiliyor... Burada Entry() aracındaki veriyi almamızı sağlayan, get_text() adlı metottur. Elbette bu metodu çok daha karmaşık işler yapmak için de kullanabilirsiniz. Mesela elimizi alıştırma için, bir sayının karekökünü hesaplayan bir uygulama yazmayı deneyelim:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk
from math import sqrt

class Arayuz(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(20)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.giris = gtk.Entry()
        self.hesapla = gtk.Button("k.kök hesapla")
        self.hesapla.connect("clicked", self.Karekok)
        self.etiket = gtk.Label("Karekök: ")
        self.s_etiket = gtk.Label("0")

        self.tablo= gtk.Table(2, 2, homogeneous=False)
        self.tablo.set_row_spacings(10)
        self.tablo.set_col_spacings(10)
        self.pencere.add(self.tablo)

        self.tablo.attach(self.giris, 0, 1, 0, 1)
        self.tablo.attach(self.hesapla, 1, 2, 0, 1)
        self.tablo.attach(self.etiket, 0, 1, 1, 2)
        self.tablo.attach(self.s_etiket, 1, 2, 1, 2)

        self.pencere.show_all()

    def Karekok(self, penar):
        try:
            self.entry = int(self.giris.get_text())
        except ValueError:
            pass

        self.karekok = sqrt(self.entry)
        self.s_etiket.set_label(str(self.karekok))

    def main(self):
        gtk.main()

gui = Arayuz()
gui.main()

```

Burada kutucuğa girilen sayıların karekötünü hesaplayan bir uygulama yazdık... Bu kodlar içinde özellikle "Karekok" adlı fonksiyonun içindeki işlemlere dikkat ediyoruz. Burada çoğunlukla Python bilgimizden yararlandık, ama bu kodları yazarken Pygtk'nin bazı kaprislerini de göz önünde bulundurmamız gerekiyor. Örneğin set_label() metoduna vereceğimiz değer "karakter dizisi (string)" olmak zorunda olduğu için, "self.karekok" değerini str() fonksiyonu yardımıyla karakter dizisine çeviriyoruz. Ayrıca bu kodlarda "self.entry" değişkenini de int() fonksiyonu yardımıyla tamsayıya çevirdiğimize dikkat edin. Çünkü normalde Entry() aracından alınacak verinin tipi karakter dizisidir. Ama biz Entry() aracından aldığımız değerle bazı aritmetik işlemler yapacağımız için, öncelikle bu değeri tamsayıya çevirmemiz gerekiyor.

Bu kodlarda, dikkat ederseniz, tek bir Table() aracı kullanarak bütün pencere araçlarımızı ana pencere üzerine yerleştirebildik. Özellikle attach() metodu içinde kullandığımız koordinat bilgilerini doğru bir şekilde yazmak ilk zamanlarda epey zor olabilir. Ama bu koordinat bilgilerini yazarken, Table() pencere aracını anlattığımız bölümdeki koordinat tablosunu gözünüzün önüne getirirseniz bu bilgileri girmek biraz daha kolaylaşabilir.

Bu arada, dikkat ettiyseniz, arayüz üzerindeki “karekök” ve “0” etiketleri birbirinden çok uzak duruyor. Bunların birbirine yakın durması görünüş açısından daha şık olurdu. Şimdi bu etiketleri istediğimiz şekilde nasıl hizalayacağımıza bakalım.

Pygtk’de kullandığımız etiketleri hizalamak için set_alignment() adlı bir metottan yararlanacağız. Bu metodu şu şekilde kullanıyoruz:

```
etiket.set_alignment(xalign, yalign)
```

Gördüğümüz gibi, bu metot iki farklı parametre alıyor. “xalign” parametresi etiketin soldan sağa doğru olan hizalamasını kontrol ederken, “yalign” parametresi ise aynı etiketin yukarıdan aşağıya doğru olan hizalamasını kontrol ediyor. Bu parametreler, “0.0”, “0.5” ve “1.0” değerlerini alabilir. “xalign” parametresine vereceğimiz “0.0” değeri etiketin en solda olmasını sağlar. “0.5” değeri etiketi ortalayacak, “1.0” değeri ise etiketi en sağa yaslayacaktır. Aynı şekilde “yalign” parametresine vereceğimiz “0.0” değeri etiketi en üste hizalayacak, “0.5” değeri ortalayacak, “1.0” değeri ise en alta hizalayacaktır... (Aslında set_alignment() metodu içinde “0.0” ile “1.0” arasındaki bütün değerleri kullanabilirsiniz. Dolayısıyla hizalama konusunda ince ayara ihtiyacınız olursa 0.1, 0.2, 0.4 vb. değerler vererek istediğiniz hizalamayı elde edebilirsiniz. Ancak pek çok durumda “0.0”, “0.5” ve “1.0” değerleri bizim için yeterli olacaktır.)

Gelin isterseniz set_alignment() metodunu yukarıda verdiğimiz örneğe uygulayalım:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import pygtk
pygtk.require(2.0)
import gtk
from math import sqrt

class Arayuz(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(20)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.giris = gtk.Entry()
        self.hesapla = gtk.Button("k.kök hesapla")
        self.hesapla.connect("clicked", self.Karekok)

        self.etiket = gtk.Label("Karekök: ")
        self.etiket.set_alignment(1.0, 0.5)

        self.s_etiket = gtk.Label("0")
        self.s_etiket.set_alignment(0.0, 0.5)

        self.tablo = gtk.Table(2, 2, homogeneous=False)
        self.tablo.set_row_spacings(10)
        self.tablo.set_col_spacings(10)
        self.pencere.add(self.tablo)
```

```

self.tablo.attach(self.giris, 0, 1, 0, 1)
self.tablo.attach(self.hesapla, 1, 2, 0, 1)
self.tablo.attach(self.etiket, 0, 1, 1, 2)
self.tablo.attach(self.s_etiket, 1, 2, 1, 2)

self.pencere.show_all()

def Karekok(self, penar):
    try:
        self.entry = int(self.giris.get_text())
    except ValueError:
        pass

    self.karekok = sqrt(self.entry)
    self.s_etiket.set_label(str(self.karekok))

def main(self):
    gtk.main()

gui = Arayuz()
gui.main()

```

Burada “self.etiket”i x düzleminde en sağa (1.0), y düzleminde ise ortaya hizaladık (0.5). Aynı şekilde “self.s_etiket”i ise x düzleminde en sola (0.0), y düzleminde de ortaya hizaladık (0.5). Y düzlemindeki hizalama ilk bakışta belli olmayabilir. Ortaya çıkan pencereyi elinizle büyütüp küçültürseniz, “yalign” parametresine verilen değerin nasıl bir etki oluşturduğunu daha açık bir şekilde görebilirsiniz.

5.1.2 “Entry”ye Girilen Veriyi Görünmez Hale Getirmek

Özellikle kullanıcı adı ve parola işlemleri için hazırlayacağımız arayüzlerde güvenlik açısından parolanın gizlenmesi gerekebilir. Örneğin, kullanıcı parolasını yazarken ekranda parola yerine “*” gibi bir karakterin görünmesini isteyebiliriz... İşte bu bölüme Entry() aracına girilen verileri bu anlamda nasıl görünmez hale getirebileceğimizi tartışacağız.

Yukarıda bahsedilen işlemi gerçekleştirmek için Pygtk’te set_visibility() adlı bir metottan yararlanacağız.

Hatırlarsanız, Entry() aracını anlatmaya başladığımız ilk bölümde şöyle bir örnek vermiştik:

```

#-*-coding:utf-8-*

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kull = gtk.Label("K. Adı: ")
        self.paro = gtk.Label("Parola: ")
        self.kentry = gtk.Entry()

```

```

self.pentry = gtk.Entry()

self.tablo = gtk.Table(4, 4)
self.tablo.attach(self.kull, 0, 1, 0, 1)
self.tablo.attach(self.paro, 0, 1, 1, 2)
self.tablo.attach(self.kentry, 1, 2, 0, 1)
self.tablo.attach(self.pentry, 1, 2, 1, 2)

self.pencere.add(self.tablo)

self.kutu = gtk.HButtonBox()
self.kutu.set_border_width(10)

self.tablo.attach(self.kutu, 0, 2, 2, 3)

self.btn = gtk.Button("Tamam")
self.kutu.pack_start(self.btn)

self.pencere.show_all()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()

```

Burada kullanıcı parolasını yazarken ekranda parolanın ne olduğu görünüyor. Ancak gerçek hayatta karşılaştığımız çoğu uygulama, kullanıcının girdiği parolayı doğrudan ekrana basmaz. Onun yerine kullanıcının girdiği her karakter için bir "*" işareti yerleştirilir. Bunu nasıl yapacağımızı görelim şimdi:

```

#-*-coding:utf-8-*

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kull = gtk.Label("K. Adı: ")
        self.paro = gtk.Label("Parola: ")
        self.kentry = gtk.Entry()
        self.pentry = gtk.Entry()
        self.pentry.set_visibility(False)

        self.tablo = gtk.Table(4, 4)
        self.tablo.attach(self.kull, 0, 1, 0, 1)
        self.tablo.attach(self.paro, 0, 1, 1, 2)
        self.tablo.attach(self.kentry, 1, 2, 0, 1)
        self.tablo.attach(self.pentry, 1, 2, 1, 2)

        self.pencere.add(self.tablo)

```

```

self.kutu = gtk.HButtonBox()
self.kutu.set_border_width(10)

self.tablo.attach(self.kutu, 0, 2, 2, 3)

self.btn = gtk.Button("Tamam")
self.kutu.pack_start(self.btn)

self.pencere.show_all()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()

```

Artık kullanıcımız parolasını yazarken parola ekranda görünmeyecek...

Gördüğümüz gibi, parola ekrana yazılırken parola yerine "*" işaretini görüyoruz. Pygtk bize bu işaretin kendisi değiştirme imkanı da sunar. Yani illa "*" işaretini kullanmak zorunda değiliz. Bunu değiştirebiliriz. Bu işlem için `set_invisible_char()` adlı başka bir metottan yararlanacağız:

```

#-*-coding:utf-8-*

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kull = gtk.Label("K. Adı: ")
        self.paro = gtk.Label("Parola: ")
        self.kentry = gtk.Entry()
        self.pentry = gtk.Entry()
        self.pentry.set_visibility(False)
        self.pentry.set_invisible_char("?")

        self.tablo = gtk.Table(4, 4)
        self.tablo.attach(self.kull, 0, 1, 0, 1)
        self.tablo.attach(self.paro, 0, 1, 1, 2)
        self.tablo.attach(self.kentry, 1, 2, 0, 1)
        self.tablo.attach(self.pentry, 1, 2, 1, 2)

        self.pencere.add(self.tablo)

        self.kutu = gtk.HButtonBox()
        self.kutu.set_border_width(10)

        self.tablo.attach(self.kutu, 0, 2, 2, 3)

        self.btn = gtk.Button("Tamam")

```

```
        self.kutu.pack_start(self.btn)

        self.pencere.show_all()

    def main(self):
        gtk.main()

uyg = Uygulama()
uyg.main()
```

Böylece parola yazılırken parolanın kendisi yerine ekranda “?” işareti görünecek...

5.1.3 “Entry”ye Girilen Verinin Azami Uzunluğunu Belirlemek

Bazı uygulamalarda Entry() aracına girilen verinin en fazla kaç karakter uzunluğunda olabileceğini önceden belirlemek gerekebilir. Örneğin bir kullanıcının girebileceği parola uzunluğunu 10 karakter ile sınırlandırmak istiyor olabilirsiniz. İşte bu işlem için Pygtk’de set_max_length() adlı bir metottan yararlanacağız.

Bir önceki bölümde verdiğimiz örnek üzerinden gidelim:

```
#!/usr/bin/env python
## coding: utf-8

import pygtk
pygtk.require(2.0)
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kull = gtk.Label("K. Adı: ")
        self.paro = gtk.Label("Parola: ")
        self.kentry = gtk.Entry()
        self.pentry = gtk.Entry()
        self.pentry.set_visibility(False)
        self.pentry.set_max_length(10)

        self.tablo = gtk.Table(4, 4)
        self.tablo.attach(self.kull, 0, 1, 0, 1)
        self.tablo.attach(self.paro, 0, 1, 1, 2)
        self.tablo.attach(self.kentry, 1, 2, 0, 1)
        self.tablo.attach(self.pentry, 1, 2, 1, 2)

        self.pencere.add(self.tablo)

        self.kutu = gtk.HButtonBox()
        self.kutu.set_border_width(10)

        self.tablo.attach(self.kutu, 0, 2, 2, 3)

        self.btn = gtk.Button("Tamam")
        self.kutu.pack_start(self.btn)
```

```
self.pencere.show_all()

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()
```

Burada kullanıcımız en fazla 10 karakterlik bir parola yazabilir. Kodlar arasında görünen `set_max_length(10)` satırı, kullanıcıyı 10 karakter fazlasını yazmasına engel olacaktır...

5.1.4 “Entry” Metninin Rengini Değiştirmek

Hatırlarsanız “Button” adlı pencere aracını işlerken `modify_bg()` adlı bir metot yardımıyla düğme renklerini değiştirebildiğimizi görmüştük. Benzer bir metodu, Entry pencere aracındaki metnin rengini değiştirmek için de kullanabiliriz. Ancak bu kez `modify_bg` yerine `modify_text` adlı bir metottan yararlanacağız. Bu metodu şu şekilde kullanıyoruz:

```
pencere_araci.modify_text(gtk.STATE_NORMAL, gtk.gdk.color_parse(renk))
```

`modify_text()` metodu, metin tabanlı pencere araçlarındaki metnin rengini değiştirmemize yardımcı olur.

Bu yapı size “Button” konusundan tanıdık geliyor olmalı... İsterseniz bununla ilgili küçük bir örnek yapalım:

```
##-coding: utf-8-

import pygtk
pygtk.require20()
import gtk

class RenkliYazi(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.hbox = gtk.HBox()
        self.pencere.add(self.hbox)

        self.tb_red = gtk.Button("kırmızı")
        self.tb_red.connect("clicked", self.renklendir, "red")

        self.entry = gtk.Entry()
        self.hbox.pack_start(self.entry)

        self.hbox.pack_start(self.tb_red)

        self.pencere.show_all()

    def renklendir(self, penar, renk):
        self.entry.modify_text(gtk.STATE_NORMAL, gtk.gdk.Color(renk))

    def main(self):
        gtk.main()
```

```
uyg = RenkliYazi()
uyg.main()
```

Burada eğer `renklendir()` fonksiyonu içindeki satır Windows'ta çalışmazsa, bunun yerine `self.entry.modify_text(gtk.STATE_NORMAL, gtk.gdk.color_parse(renk))` satırını yazabileceğinizi biliyorsunuz...

5.1.5 “Entry” ile bir Örnek...

Bu bölümde `Entry()` pencere aracını kullanarak bir örnek uygulama yazacağız. Bu örnek uygulamayı yazarken aynı zamanda şimdiye kadar gördüğümüz konuları da tekrar etmiş olacağız.

Burada amacımız, kullanıcıdan iki adet sayı alan ve bu sayıları birbirleriyle toplayıp birbirlerinden çıkaran bir uygulama yazmak...

İlk kodlarımızı yazarak başlayalım işe:

```
##-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk
```

Bu kodlar uygulamamızın demirbaşları... Bu satırların ne işe yaradığını artık adımımız gibi biliyoruz.

Devam edelim...

```
class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_position(gtk.WIN_POS_CENTER)
        self.pencere.set_border_width(20)
        self.pencere.connect("delete_event", gtk.main_quit)
```

Burada “Uygulama” adlı bir sınıf tanımladık. Bu sınıfımız, Python’daki yeni tip sınıfların gerektirdiği şekilde “object” sınıfından miras alıyor.

Sınıfımız içinde bir “__init__” fonksiyonu tanımlıyoruz. Arayüz oluşur oluşmaz hayata geçmesi gereken özellikleri bu __init__ fonksiyonu içine yazacağız. Burada öncelikle penceremizi oluşturduk. Ardından da “set_position()” metodunu kullanarak penceremizi ekranın en ortasında konumlandırdık. Bir sonraki satırda kullandığımız “set_border_width()” metodu ise pencere araçlarının pencere kenarına göre ne kadar uzakta yer alacağını belirliyor. “connect()” metodu ise pencere aracına bir görev atamamızı sağlıyor. Biz burada, pencerenin çarpı düğmesine basıldığında programımızın sağlıklı bir şekilde kapanabilmesini sağlamak için kullandık bu metodu...

Devam ediyoruz:

```
self.etk_ilk = gtk.Label("ilk sayı:")
self.etk_ilk.set_alignment(0.0, 0.5)

self.etk_iki = gtk.Label("ikinci sayı:")
self.etk_iki.set_alignment(0.0, 0.5)
```

```
self.etk_sonuc = gtk.Label("sonuç:")
self.etk_sonuc.set_alignment(0.0, 0.5)
```

Bu satırlarda, arayüz üzerinde yer alacak Label() araçlarını, yani etiketleri oluşturuyoruz. Arayüzümüz üzerinde “ilk sayı:”, “ikinci sayı:” ve “üçüncü sayı:” şeklinde üç adet etiket bulunacak. Bir önceki bölümde öğrendiğimiz set_alignment() metodunu kullanarak bu etiketleri sola yaslıyoruz. Bildiğiniz gibi, set_alignment() metodu “xalign” ve “yalign” olmak üzere iki farklı parametre alıyor. Bu iki parametre, “0.0”, “0.5” ve “1.0” olmak üzere üç farklı değer alabilir. Yukarıdaki kodlarda “xalign” parametrelerine “0.0” değerini vererek etiketimizin sola yaslanmasını sağladık. Eğer etiketleri ortalamak isteseydik xalign parametresine 0.5 değeri vermemiz gerekecekti. Etiketleri en sağa yaslamak için ise “1.0” değerini kullanacaktık. Yazacağımız uygulamalarda “yalign” parametresine genellikle “0.5” değeri vermek mantıklı bir seçim olacaktır. Çünkü yazacağımız uygulamalarda etiketlerimizin y düzleminde (yukarıdan aşağıya doğru) genellikle orta noktaya denk gelmesini isteyeceğiz.

Şimdi yukarıda tanımladığımız her bir etiketin karşısına denk gelecek Entry() araçlarını oluşturacağız:

```
self.ent_ilk = gtk.Entry()
self.ent_ilk.set_alignment(0.5)

self.ent_iki = gtk.Entry()
self.ent_iki.set_alignment(0.5)

self.ent_sonuc = gtk.Entry()
self.ent_sonuc.set_alignment(0.5)
```

Burada yine set_alignment() metodunu kullandığımızı dikkat edin. Ancak bu metodun Entry() aracıyla birlikte kullanıldığında bazı farklılıklar gösterir. set_alignment() metodunu Label() aracıyla birlikte kullanırken bu metoda iki farklı parametre veriyorduk. Ancak bu metodu Entry() aracıyla kullandığımızda tek bir parametre vermemiz yeterli olacaktır. Yukarıda Entry() aracıyla birlikte kullandığımız set_alignment() metodu, Entry() aracına yazılan verilerin kutucuğun ortasında konumlanmasını sağlıyor.. Eğer bu set_alignment() satırlarını kaldıracak olursak, kullanıcının kutucuğa girdiği veriler sola yaslanacaktır. Kendi göz zevkinize göre set_alignment() satırlarını yazıp yazmamaya karar verebilirsiniz...

Bir sonraki kısımda ise toplama ve çıkarma işlemlerini yapmamızı sağlayacak düğmeleri oluşturuyoruz. “Toplama” ve “Çıkarma” işlemi için iki ayrı düğme meydana getiriyoruz:

```
self.btn_topla = gtk.Button("topla")
self.btn_topla.connect("clicked", self.islem, "topla")

self.btn_cikar = gtk.Button("çıkarma")
self.btn_cikar.connect("clicked", self.islem, "çıkarma")
```

Oluşturduğumuz bu düğmeleri, daha sonra tanımlayacağımız “islem” adlı bir fonksiyona bağlıyoruz. Burada, connect() metodunun üçüncü parametresini de yazdığımızı dikkat edin. “topla” ve “çıkarma” olarak yazdığımız bu üçüncü parametreler sayesinde bütün işlemleri tek bir fonksiyon yardımıyla halledebileceğiz:

```
self.anakutu = gtk.VBox()
self.pencere.add(self.anakutu)

self.tablo = gtk.Table(3, 3)
self.anakutu.pack_start(self.tablo)
```

```

self.tablo.attach(self.etk_ilk, 0, 1, 0, 1)
self.tablo.attach(self.etk_iki, 0, 1, 1, 2)
self.tablo.attach(self.etk_sonuc, 0, 1, 2, 3)
self.tablo.attach(self.ent_ilk, 1, 2, 0, 1)
self.tablo.attach(self.ent_iki, 1, 2, 1, 2)
self.tablo.attach(self.ent_sonuc, 1, 2, 2, 3)

self.hbut = gtk.HButtonBox()
self.hbut.set_layout(gtk.BUTTONBOX_SPREAD)
self.anakutu.pack_start(self.hbut, padding=10)

self.hbut.add(self.btn_topla)
self.hbut.add(self.btn_cikar)

self.pencere.show_all()

```

Yukarıdaki satırlarda bir adet “VBox()”, bir adet “Table()” bir adet de “HButtonBox()” kullandık... Buradaki “VBox()” haznesi öteki araçları içine yerleştireceğimiz ana kutumuz olma görevini üstleniyor. Table() ve HButtonBox() araçlarını ana kutumuzun üzerine yerleştireceğiz. Burada hangi öğeleri hangi haznelerin içine yerleştirdiğimize dikkat edin. Table() aracı içine etiketleri (Label) ve kutucukları (Entry) yerleştiriyoruz. “topla” ve “çıkara” düğmelerini HButtonBox() içine alıyoruz. Bütün bu öğelerin hepsini ise VBox() içine koyuyoruz... Son olarak VBox()’u da ana pencere üzerine yerleştiriyoruz...

```

def islem(self, penar, data):
    self.ent_sonuc.set_text("")
    self.ilk = self.ent_ilk.get_text()
    self.iki = self.ent_iki.get_text()
    try:
        if data == "topla":
            self.hesap = int(self.ilk) + int(self.iki)
        elif data == "çıkara":
            self.hesap = int(self.ilk) - int(self.iki)
        else:
            self.hesap = "İmkansız!"
    except ValueError:
        pass
    else:
        self.ent_sonuc.set_text(str(self.hesap))

def main(self):
    gtk.main()

uyg = Uygulama()
uyg.main()

```

Bu bölümde ise toplama ve çıkarma işlemlerini gerçekleştiren “islem” adlı fonksiyonu yazıyoruz. Öncelikle “self.ent_sonuc.set_text(“”)” satırı ile sonuç kutusunu boşaltıyoruz. Böylece her işlemden sonra sonuç kutusu içindeki değerler temizlenmiş olacak. Aksi halde art arda gelen işlemlerin sonucu üst üste binecektir...

Sonraki iki satırda işlem kutucuklarına kullanıcı tarafından girilen verileri alıyoruz. Bunun için get_text() adlı metottan yararlanıyoruz.

Daha sonra ise bir try... except... else... bloğu oluşturuyoruz. “try...” bloğu içinde, connect() metodunun üçüncü parametresinin hangi değere sahip olduğuna bağlı olarak toplama veya çıkarma işlemi yapıyoruz. Eğer üçüncü parametre “topla” ise, yani kullanıcı

toplama işlemini yapan düğmeye basmışsa, “self.hesap = int(self.ilk) + int(self.iki)” işlemi yapılacak, eğer üçüncü parametre “çıkart” ise, yani kullanıcı çıkarma işlemini yapan düğmeye basmışsa, “self.hesap = int(self.ilk) - int(self.iki)” işlemi yapılacaktır. Burada bir adet de “else” bloğu yazdık. Ama bu bloğu yazmasak da olurdu, çünkü yukarıdaki if ve elif blokları içinde belirtilen durumlar haricinde başka bir durumun gerçekleşmesi imkansızdır... Ama biz yine de işimizi sağlama almayı tercih ettik...

except... bloğunda “ValueError” hatalarını yakalıyoruz. Çünkü kullanıcı kutucuklara sayı yerine harf girerse programımız hata verecektir. Böyle bir durum olduğunda pass deyimi gereğince hiçbir şey yapmadan bir sonraki adıma geçiyoruz.

else... bloğunda ise, program hatasız bir şekilde çalışırsa ne yapılacağını belirliyoruz. Buna göre, eğer programdan herhangi bir hata alınmazsa, sonuç kutucuğu içine “hesap” değişkeninin değerini yazdırıyoruz. Burada “hesap” değişkenini str() fonksiyonu yardımıyla karakter dizisine çevirdiğimize dikkat edin. Çünkü Entry() araçlarına yalnızca karakter dizileri girebiliriz...

Sonraki kodları ise zaten tanıyorsunuz. “def main...” satırı ile main() fonksiyonunu tanımlıyoruz. Eğer bu fonksiyonu yazmak programımız çalışmayacaktır. Son olarak ise “Uygulama()” adlı sınıfımızı örnekliyoruz ve “uyg” örneği üzerinden main() fonksiyonunu çağırarak programımızı başlatıyoruz.

İsterseniz bu kodları bir bütün olarak görelim:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_position(gtk.WIN_POS_CENTER)

        self.pencere.set_border_width(20)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.etk_ilk = gtk.Label("ilk sayı:")
        self.etk_ilk.set_alignment(0.0, 0.5)

        self.etk_iki = gtk.Label("ikinci sayı:")
        self.etk_iki.set_alignment(0.0, 0.5)

        self.etk_sonuc = gtk.Label("sonuç:")
        self.etk_sonuc.set_alignment(0.0, 0.5)

        self.ent_ilk = gtk.Entry()
        self.ent_ilk.set_alignment(0.5)

        self.ent_iki = gtk.Entry()
        self.ent_iki.set_alignment(0.5)

        self.ent_sonuc = gtk.Entry()
        self.ent_sonuc.set_alignment(0.5)

        self.btn_topla = gtk.Button("topla")
        self.btn_topla.connect("clicked", self.islem, "topla")
```

```

self.btn_cikar = gtk.Button("çıkıkar")
self.btn_cikar.connect("clicked", self.islem, "çıkıkar")

self.anakutu = gtk.VBox()
self.pencere.add(self.anakutu)

self.tablo = gtk.Table(3, 3)
self.anakutu.pack_start(self.tablo)

self.tablo.attach(self.etk_ilk, 0, 1, 0, 1)
self.tablo.attach(self.etk_iki, 0, 1, 1, 2)
self.tablo.attach(self.etk_sonuc, 0, 1, 2, 3)
self.tablo.attach(self.ent_ilk, 1, 2, 0, 1)
self.tablo.attach(self.ent_iki, 1, 2, 1, 2)
self.tablo.attach(self.ent_sonuc, 1, 2, 2, 3)

self.hbut = gtk.HButtonBox()
self.hbut.set_layout(gtk.BUTTONBOX_SPREAD)
self.anakutu.pack_start(self.hbut, padding=10)

self.hbut.add(self.btn_topla)
self.hbut.add(self.btn_cikar)

self.pencere.show_all()

def islem(self, penar, data):
    self.ent_sonuc.set_text("")
    self.ilc = self.ent_ilk.get_text()
    self.iki = self.ent_iki.get_text()
    try:
        if data == "topla":
            self.hesap = int(self.ilc) + int(self.iki)
        elif data == "çıkıkar":
            self.hesap = int(self.ilc) - int(self.iki)
        else:
            self.hesap = "İmkansız!"
    except ValueError:
        pass
    else:
        self.ent_sonuc.set_text(str(self.hesap))

def main(self):
    gtk.main()

```

```

uyg = Uygulama()
uyg.main()

```

Bu arada, gördüğünüz gibi, klavye üzerindeki pek çok kısayol tuşu, yazdığımız uygulama üzerinde otomatik olarak çalışıyor. Örneğin, klavyedeki sekme (tab) tuşuna basarak pencere araçları üzerinde hareket edebiliyoruz. Aynı şekilde klavye üzerindeki ok tuşları da işlevli durumda... Örneğin "topla" düğmesi seçiliyken enter tuşuna basarsak birinci ve ikinci kutudaki değerler toplanacak; "çıkıkar" düğmesi seçiliyken enter tuşuna basarsak da birinci ve ikinci kutudaki değerler birbirinden çıkarılacaktır...

5.2 “ToggleButton” Pencere Aracı

Bu pencere aracı, daha önce gördüğümüz “Button” adlı pencere aracına çok benzer. Ama ToggleButton ile Button arasında önemli farklar bulunur. “Toggle Button” bir nevi “açma-kapama düğmesi”dir. ToggleButton’lar “açık” ve “kapalı” olmak üzere iki farklı konumda bulunabilir. “Button” adlı düğmelerin ise böyle bir özelliği yoktur.

Pygtk’de bir ToggleButton oluşturmak için şöyle bir yapı kullanıyoruz:

```
tb = gtk.ToggleButton("düğme adı")
```

ToggleButton’u oluşturduktan sonra bu düğmeye bir kez bastığınızda düğmenin basılı kaldığını, ikinci kez bastığınızda ise düğmenin eski haline döndüğünü göreceksiniz. Bu etkiyi birden fazla ToggleButton oluşturarak daha net bir şekilde görebilirsiniz. Dilerseniz ToggleButton ile ilgili küçük bir örnek yapalım:

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class ToggleButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(30)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.vbox = gtk.VBox(True, 0)
        self.vbox.set_spacing(10) #bu komutla satır aralıklarını ayarlıyoruz.
        self.pencere.add(self.vbox)

        #şimdi beş adet ToggleButton oluşturacağız...
        for i in range(5):
            self.tb = gtk.ToggleButton("düğme%s"%i)
            self.vbox.pack_start(self.tb)

        self.pencere.show_all()

    def main(self):
        gtk.main()

ackapa = ToggleButton()
ackapa.main()
```

Gördüğünüz gibi, ToggleButton oluşturmak Button oluşturmaktan çok farklı değil. Şimdi gelelim bu ToggleButton adlı pencere aracının özelliklerine...

5.2.1 ToggleButton’un Durumunu Öğrenmek

Pygtk’de ToggleButton adlı pencere aracının “açık” ve “kapalı” olmak üzere iki farklı konumda bulunduğunu söylemiştik. Bir ToggleButton’un basılı konumu “True”, basılı olmayan konumu ise “False” değerine sahiptir. ToggleButton’un basılı durumda olup olmadığı bilgisi get_active() adlı bir metot yardımıyla alabiliriz:

```

#-*-coding:utf-8-*-

import pygtk
pygtk.require20()
import gtk

class ToggleButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(30)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.vbox = gtk.VBox(True, 0)
        self.vbox.set_spacing(10)
        self.pencere.add(self.vbox)

        for i in range(5):
            self.tb = gtk.ToggleButton("düğme%s"%i)

            #burada "self.tb.get_label()" fonksiyonu bize düğmenin adını veriyor
            #Eğer isterseniz "clicked" yerine "toggled" sinyalini de kullanabilirsiniz.
            self.tb.connect("clicked", self.fonk, self.tb.get_label())
            self.vbox.pack_start(self.tb)

        self.pencere.show_all()

    def fonk(self, penar, data):
        # "get_active()" metodunu nasıl kullandığımıza dikkat edin.
        # Bu metod bize ToggleButton'un basılı olup olmadığı
        # bilgisini veriyor.
        print "%s: %s"%(data, penar.get_active())

    def main(self):
        gtk.main()

tb = ToggleButton()
tb.main()

```

Bu kodları çalıştırdıktan sonra arayüz üzerindeki düğmelere bastığımızda hem düğmenin adı hem de düğmenin basılı olup olmadığı bilgisi komut satırında görüntülenecektir. "True" değeri düğmenin basılı olduğunu, "False" değeri ise düğmenin basılı olmadığını gösterir. Eğer "True" ve "False" değerleri yerine daha anlamlı kelimeler kullanmak isterseniz fonk() adlı fonksiyonu şöyle yazabilirsiniz:

```

def fonk(self, penar, data):
    print "%s %s"%(data, ("kapalı", "açık")[penar.get_active()])

```

Böylece düğmeye ilk basışta komut satırında "açık" çıktısı, ikinci basışta ise "kapalı" çıktısı görüntülenecektir... "Kapalı" ve "açık" kelimeleri yerine istediğiniz kelimeleri koymakta özgürsünüz...

Dilerseniz ToggleButton'un kullanımını biraz daha netleştirmek için ufak bir örnek verelim:

```

#-*-coding:utf-8-*-

import pygtk

```

```

pygtk.require20()
import gtk

class ToggleButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(30)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.tablo = gtk.Table(4, 2, True)
        self.tablo.set_row_spacings(5)
        self.pencere.add(self.tablo)

        #beş tane farklı ToggleButton oluşturuyoruz...
        for i in range(5):
            self.btn = gtk.ToggleButton("düğme%s"%i)

            self.etk = gtk.Label()
            self.etk.set_text("")

            #oluşturduğumuz düğme ve etiketleri tabloya yerleştiriyoruz.
            self.tablo.attach(self.btn, 0, 1, i, i+1)
            self.tablo.attach(self.etk, 1, 2, i, i+1)

            self.btn.connect("clicked", self.fonk, self.etk)

        self.pencere.show_all()

        #düğmelere her basışta düğme durumunun etikete yazdırılması için
        #"fonk()" adlı bir fonksiyon yazıyoruz...
        def fonk(self, dugme, etiket):
            self.durum = ("kapalı", "açık")[dugme.get_active()]
            etiket.set_text(str(self.durum))

        def main(self):
            gtk.main()

tb = ToggleButton()
tb.main()

```

Burada ToggleButton'un açık mı yoksa kapalı mı olduğu bilgisi komut satırına değil, arayüz üzerindeki etiketlere yazdırılıyor...

5.2.2 ToggleButton'un Açılış Değerini Belirlemek

Normal şartlar altında bir ToggleButton ilk oluşturulduğunda kapalıdır. Eğer oluşturduğunuz ToggleButton'un ilk açıldığında basılı/açık olmasını isterseniz set_active() adlı metottan yararlanabilirsiniz:

```

tb = gtk.ToggleButton("isim")
tb.set_active(True)

```

Böylece ToggleButton ilk oluştuğunda "True" değerine sahip olacaktır.

5.3 “CheckBoxton” Pencere Aracı

“CheckBoxton” bildiğimiz onay kutusudur... Pygtk’de oluşturduğumuz arayüzlere “CheckBoxton” eklemek için şu yolu takip ediyoruz:

```
onay_kutusu = gtk.CheckBoxton("isim")
```

Yukarıda “ToggleButton” için verdiğimiz örneği aynen “CheckBoxton” için de verebiliriz. Tek fark, yukarıda “ToggleButton” geçen yerlere “CheckBoxton” yazıyoruz:

```
#-*- coding:utf-8 -*-

import pygtk
pygtk.require20()
import gtk

class CheckBoxton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(30)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.tablo = gtk.Table(4, 2, True)
        self.tablo.set_row_spacings(5)
        self.pencere.add(self.tablo)

        #beş tane farklı CheckBoxton oluşturuyoruz...
        for i in range(5):
            self.btn = gtk.CheckBoxton("düğme%s"%i)

            self.etk = gtk.Label()
            self.etk.set_text("")

            #oluşturduğumuz düğme ve etiketleri tabloya yerleştiriyoruz.
            self.tablo.attach(self.btn, 0, 1, i, i+1)
            self.tablo.attach(self.etk, 1, 2, i, i+1)

            self.btn.connect("clicked", self.fonk, self.etk)

        self.pencere.show_all()

        #düğmelere her basışta düğme durumunun etikete yazdırılması için
        #"fonk()" adlı bir fonksiyon yazıyoruz...
        def fonk(self, dugme, etiket):
            self.durum = ("seçili değil", "seçili")[dugme.get_active()]
            etiket.set_text(str(self.durum))

        def main(self):
            gtk.main()

tb = CheckBoxton()
tb.main()
```

5.4 “RadioButton” Pencere Aracı

ToggleButton ve CheckButton’a çok benzeyen başka bir pencere aracı da “RadioButton”dur. Bu pencere aracını şu şekilde oluşturuyoruz:

```
rb = gtk.Radiobutton(grup, "isim")
```

Bu formülde, öteki pencere araçlarında olmayan bir parametre göze çarpıyor. Pygtk’de bir “RadioButton” pencere aracı oluşturabilmek için formülde görünen bu “grup” adlı parametreyi mutlaka belirtmemiz gerekir. Eğer bu parametreyi belirtmeden şöyle bir örnek kod yazacak olursak Python bize bir hata mesajı gösterecektir:

```
##-coding:utf-8-
import pygtk
pygtk.require20()
import gtk

class RadioButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.rb = gtk.Radiobutton("rb1")
        self.pencere.add(self.rb)
        self.pencere.show_all()

    def main(self):
        gtk.main()

rb = RadioButton()
rb.main()
```

Bu kodları çalıştırdığımızda şu hata mesajını alırız:

```
TypeError: first argument must be a GtkRadioButton or None
```

Pygtk’de bir RadioButton oluşturacağımız zaman, bu pencere aracını mutlaka başka bir veya daha fazla RadioButton grubuna dahil etmemiz gerekir. Peki ya elimizde sadece tek bir RadioButton varsa ne olacak? Aslında mantık olarak arayüz üzerinde tek bir RadioButton kullanmak son derece anlamsızdır. Neticede arayüz üzerinde birden fazla RadioButton olmalı, ki kullanıcı bu RadioButton’lar arasında seçim yapabilsin...

Şimdi isterseniz daha mantıklı bir örnek verelim:

```
##-coding:utf-8-
import pygtk
pygtk.require20()
import gtk

class RadioButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.vbox = gtk.VBox()
```

```

self.etiket = gtk.Label("Python'un Hangi Sürümünü Kullanıyorsunuz?")

self.rb1 = gtk.RadioButton(None, "Python 2.5")
self.rb2 = gtk.RadioButton(self.rb1, "Python 2.6")
self.rb3 = gtk.RadioButton(self.rb1, "Python 3.0")
self.rb4 = gtk.RadioButton(self.rb1, "Python 3.1")

self.vbox.pack_start(self.etiket, True, True, 10)
self.vbox.pack_start(self.rb1)
self.vbox.pack_start(self.rb2)
self.vbox.pack_start(self.rb3)
self.vbox.pack_start(self.rb4)

self.pencere.add(self.vbox)

self.pencere.show_all()

def main(self):
    gtk.main()

rb = RadioButton()
rb.main()

```

Burada, "self.rb1" adlı ilk RadioButton aracı herhangi bir gruba üye olamayacağı için, bunun "grup" parametresine "None" değeri verdik. Ondan sonra oluşturduğumuz RadioButton'ları ise "self.rb1" ile grupladık. Böylece bütün RadioButton'lar birbirlerinden haberdar olmuş oldu. İsterseniz oluşturduğumuz dört RadioButton'a da ilk parametre olarak "None" değerini vermeyi deneyin. Bu şekilde RadioButton'ların birbirinden habersiz olduğunu göreceksiniz...

Hangi RadioButton'un seçili olduğunu, hangisini seçili olmadığını öğrenmek için de get_active() adlı metodu kullanabilirsiniz. Mesela yukarıdaki örneği şöyle yazabilirsiniz:

```

#-*-coding:utf-8-*-
import pygtk
pygtk.require20()
import gtk

class RadioButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.vbox = gtk.VBox()

        self.etiket = gtk.Label("Python'un Hangi Sürümünü Kullanıyorsunuz?")

        self.rb1 = gtk.RadioButton(None, "Python 2.5")
        self.rb1.connect("clicked", self.fonk)

        self.rb2 = gtk.RadioButton(self.rb1, "Python 2.6")
        self.rb2.connect("clicked", self.fonk)

        self.rb3 = gtk.RadioButton(self.rb1, "Python 3.0")
        self.rb3.connect("clicked", self.fonk)

        self.rb4 = gtk.RadioButton(self.rb1, "Python 3.1")

```

```

self.rb4.connect("clicked", self.fonk)

self.vbox.pack_start(self.etiket, True, True, 10)
self.vbox.pack_start(self.rb1)
self.vbox.pack_start(self.rb2)
self.vbox.pack_start(self.rb3)
self.vbox.pack_start(self.rb4)

self.pencere.add(self.vbox)

self.pencere.show_all()

def fonk(self, penar):
    durum = (u"seçili değil", u"seçili")[penar.get_active()]
    print "%s %s" % (penar.get_label(), durum)

def main(self):
    gtk.main()

rb = RadioButton()
rb.main()

```

fonk() adlı fonksiyon içindeki yapıyı, daha önce gördüğümüz CheckButton ve ToggleButton adlı pencere araçlarından da hatırlıyor olmalısınız.

RadioButton'lardaki bu "grup" parametresi, farklı RadioButton gruplarıyla çalışmak istediğinizde işinize çok yarayacaktır. Mesela buna şöyle bir örnek verebiliriz:

```

#-*-coding:utf-8-*-
import pygtk
pygtk.require20()
import gtk

class RadioButton(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.vbox = gtk.VBox()

        self.etiket_py = gtk.Label("Python'un Hangi Sürümünü Kullanıyorsunuz?")

        self.rb1_py = gtk.RadioButton(None, "Python 2.5")
        self.rb1_py.connect("clicked", self.fonk)

        self.rb2_py = gtk.RadioButton(self.rb1_py, "Python 2.6")
        self.rb2_py.connect("clicked", self.fonk)

        self.rb3_py = gtk.RadioButton(self.rb1_py, "Python 3.0")
        self.rb3_py.connect("clicked", self.fonk)

        self.rb4_py = gtk.RadioButton(self.rb1_py, "Python 3.1")
        self.rb4_py.connect("clicked", self.fonk)

        self.etiket_de = gtk.Label("Hangi Masaüstü Ortamını Kullanıyorsunuz?")

        self.rb1_de = gtk.RadioButton(None, "KDE 3.5")

```

```

self.rb1_de.connect("clicked", self.fonk)

self.rb2_de = gtk.RadioButton(self.rb1_de, "KDE 4")
self.rb2_de.connect("clicked", self.fonk)

self.rb3_de = gtk.RadioButton(self.rb1_de, "GNOME")
self.rb3_de.connect("clicked", self.fonk)

self.rb4_de = gtk.RadioButton(self.rb1_de, "XFCE")
self.rb4_de.connect("clicked", self.fonk)

self.vbox.pack_start(self.etiket_py, True, True, 10)
self.vbox.pack_start(self.rb1_py)
self.vbox.pack_start(self.rb2_py)
self.vbox.pack_start(self.rb3_py)
self.vbox.pack_start(self.rb4_py)

self.vbox.pack_start(self.etiket_de, True, True, 10)
self.vbox.pack_start(self.rb1_de)
self.vbox.pack_start(self.rb2_de)
self.vbox.pack_start(self.rb3_de)
self.vbox.pack_start(self.rb4_de)

self.pencere.add(self.vbox)

self.pencere.show_all()

def fonk(self, penar):
    durum = (u"seçili değil", u"seçili")[penar.get_active()]
    print "%s %s" % (penar.get_label(), durum)

def main(self):
    gtk.main()

rb = RadioButton()
rb.main()

```

Burada aynı arayüz üzerinde iki farklı RadioButton grubu kullanıyoruz. İlk grubumuz Python sorusuyla ilgili. İkinci grup ise Masaüstü sorusuyla... RadioButton aracındaki "grup" parametresi sayesinde iki farklı grup içinde yer alan pencere araçlarının birbiriyle karışmasını engelliyoruz.

Pygtk'de Resimlerle Çalışmak

Arayüz tasarlarken, programlarımızın görselliğini artırmak, bunları daha göze hitap eder bir hale getirmek için çoğu zaman arayüz üzerine ufak resimler eklemek isteriz. Resimler, abartılmadıkları sürece, programların kullanılabilirliği üzerinde olumlu etkide bulunan öğelerdir.

Pygtk, resimlerle çalışmamızı kolaylaştıracak pek çok araç barındırır. İşte biz de bu bölümde bu araçların neler olduğunu, Pygtk uygulamalarına nasıl resim ekleyebileceğimizi inceleyeceğiz.

6.1 Pencere Araçlarına Stoktan Resim Ekleme

Bildiğiniz gibi, Pygtk'de bir düğme oluşturmak için basitçe şu formülü kullanıyorduk:

```
btn = gtk.Button("düğme")
```

Burada parantez içinde belirttiğimiz karakter dizisi düğmemizin adı oluyor... "Button" adlı pencere aracı bunun dışında bir parametre daha alır. Bu parametrenin adı "stock"tur. Bunu şöyle kullanıyoruz:

```
btn = gtk.Button(stock = resim)
```

"stock" parametresinin yerine <http://www.pygtk.org/docs/pygtk/gtk-stock-items.html> adresinde gösterilen değerlerden birini yazabilirsiniz. İsterseniz hemen bununla ilgili bir örnek verelim:

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-
import pygtk
pygtk.require20()
import gtk

class ResimliDugme(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.set_border_width(10)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.vbox = gtk.VBox(False, 10)
```

```

self.etik = gtk.Label("Çıkmak istediğimize emin misiniz?")

self.vbox.pack_start(self.etik, False, False, 0)

self.hbut = gtk.HButtonBox()

self.vbox.pack_start(self.hbut)

self.btn_cik = gtk.Button(stock = gtk.STOCK_QUIT)
self.btn_cik.connect("clicked", gtk.main_quit)

self.hbut.pack_start(self.btn_cik, False, False, 0)

self.btn iptal = gtk.Button(stock = gtk.STOCK_CANCEL)

self.hbut.pack_start(self.btn iptal, False, False, 0)

self.pencere.add(self.vbox)

self.pencere.show_all()

def main(self):
    gtk.main()

rd = ResimliDugme()
rd.main()

```

Burada düğmeler için kendimiz ayrıca bir ad belirlemedik. "stock" parametresi içinde belirttiğimiz resim, düğme için uygun bir isim de belirler.

Eğer yukarıdaki örneği çalıştırdığınızda düğme üzerinde herhangi bir simge/resim görmüyorsanız bunun sebebi kullandığınız GTK temasıdır.. Windows'ta GTK ile birlikte gelen "gtk-themeselector" adlı uygulama yardımıyla kullandığınız temayı değiştirerek düğme üzerinde resim gösterebilen bir tema seçebilirsiniz. "gtkthemeselector" adlı uygulamaya *Başlat > Programlar > GTK* yolunu takip ederek ulaşabilirsiniz. Eğer bu programı menü içinde bulamazsanız "C:\GTK\bin" klasörü içindeki "gtkthemeselector.exe" adlı dosyaya çift tıklayarak da programı çalıştırabilirsiniz. Orada görünen temalar içinde mesela "Metal" adlı tema düğmeler üzerindeki resimlerin görünmesini sağlar. Eğer "Ben mutlaka varsayılan temayı kullanmak istiyorum. Ama düğme üzerindeki resimler de görünsün istiyorum," diyorsanız "C:\GTK\share\themes\MS-Windows\gtk-2.0" klasörü içindeki "gtkrc" adlı dosyayı düzenleyerek varsayılan temanın, düğme üzerindeki resimleri göstermesini sağlayabilirsiniz. O dosya içindeki "gtk-button-images" adlı seçeneğin değerini "1" yapıp dosyayı bu şekilde kaydetmeniz yeterli olacaktır.. Yani o seçenek şöyle görünmeli:

```
gtk-button-images = 1
```

GNU/Linux'ta Ubuntu dağıtımının geliştiricilerinin GNOME masaüstü ayarlarında yaptıkları bir değişiklik nedeniyle Ubuntu'da Pygtk uygulamalarındaki düğme simgeleri görünmez.. Üstelik kullandığınız GTK temasını değiştirmek de burada hiçbir işe yaramayacaktır. Ubuntu'da düğme simgelerini gösterebilmek için "gconf-editor" içindeki "/desktop/gnome/interface/" yolu içindeki "buttons_have_icons" ve "menus_have_icons" değerlerine birer tik atmamız gerekiyor...

6.2 Pencere Araçlarına Dosyadan Resim Ekleme

Yukarıda, pencere araçlarına nasıl stoktan resim ekleyeceğimizi gördük. Şimdi ise pencere araçlarına bilgisayarımızda bulunan başka resimleri de eklemeyi öğreneceğiz. Dilerseniz bir örnek üzerinden gidelim:

```
import pygtk
pygtk.require20()
import gtk

class Resim(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.resim = gtk.Image()
        self.resim.set_from_file("falanca.png")
        self.resim.show()

        self.btn = gtk.Button()
        self.btn.add(self.resim)
        self.btn.show()

        self.pencere.add(self.btn)

        self.pencere.show()

    def main(self):
        gtk.main()

r = Resim()
r.main()
```

Burada öncelikle `gtk.Image()` fonksiyonunu kullanarak bir resim nesnesi oluşturuyoruz. Daha sonra `set_from_file()` adlı metottan yararlanarak bilgisayarımızda bulunan "falanca.png" adlı dosyayı seçiyoruz. Böylece programımızda hangi resmi kullanacağımızı belirlemiş olduk. Tabii `show()` metodunu kullanarak resmimizi göstermeyi unutmuyoruz...

Resmimizi tanımladıktan sonra normal şekilde bir düğme oluşturuyoruz ve `add()` metodunu kullanarak resmi düğmeye ekliyoruz. Yine `show()` metodunu kullanarak düğmemizi göstermeyi ihmal etmiyoruz.

Son olarak düğmemizi de pencereye ekleyip gösterdikten sonra işimiz bitmiş oluyor...

6.3 Pencere Araçlarına Resim ve Etiket Ekleme

Yukarıda, bir pencere aracına nasıl resim ekleyeceğimizi gördük. Peki aynı pencere aracına hem resim, hem de etiket eklemek istersek ne yapacağız? Dilerseniz yine bir örnek üzerinden gidelim:

```
#-*-coding: utf-8-*-

import pygtk
pygtk.require20()
```

```
import gtk

class Uygulama(object):
    def __init__(self):
        #penceremizi oluşturalım...
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)

        #penceremizin düzgün kapanmasını sağlayalım...
        self.pencere.connect("delete_event", gtk.main_quit)

        #burada, biraz sonra tanımlayacağımız "kutu_olustur()" adlı fonksiyonu
        #çağırıp bunu bir değişkene atıyoruz.
        self.kutu = self.kutu_olustur("istihzacom.png", "www.istihza.com")

        #düğmemiz...
        self.btn = gtk.Button()

        #biraz önce tanımladığımız kutuyu düğmenin içine yerleştiriyoruz
        self.btn.add(self.kutu)

        #düğmemizi de pencerenin içine koyuyoruz.
        self.pencere.add(self.btn)

        #pencereyi ve pencere araçlarını gösteriyoruz.
        self.pencere.show_all()

    def kutu_olustur(self, rsm, etk):
        #biraz sonra tanımlayacağımız resim ve etiketi tutması için bir HBox()
        #haznesi oluşturuyoruz...
        self.hbox = gtk.HBox()

        #resmimizi tanımlıyoruz.
        self.resim = gtk.Image()

        #set_from_file() metodu ile bilgisayarımızda bulunan bir resmi alıyoruz.
        #yukarıda __init__ fonksiyonu içinde "kutu_olustur()" fonksiyonunu
        #çağırırken "rsm" değişkenini "istihzacom.png" olarak belirledik. Yani
        #bilgisayarımızdan aldığımız resmin adı "istihzacom.png" olacak.
        self.resim.set_from_file(rsm)

        #burada da etiketimizi tanımlıyoruz. Etiket metninin yerinde gördüğünüz
        #"etk" değişkenini yukarıdaki __init__ fonksiyonu içinde
        #"www.istihza.com" olarak belirlemiştik. Dolayısıyla etiket metnimiz
        #"www.istihza.com" olacak.
        self.etiket = gtk.Label(etk)

        #şimdi resim ve etiketi en başta oluşturduğumuz HBox() haznesi içine
        #tek tek yerleştiriyoruz.
        self.hbox.pack_start(self.resim)
        self.hbox.pack_start(self.etiket)

        #resmimizi ve etiketimizi gösteriyoruz.
        self.resim.show()
        self.etiket.show()

        #fonksiyonumuz HBox() haznesini döndürüyor... Böylece HBox() haznesini,
        #içine yerleştirdiğimiz resim ve etiketle birlikte kullanabiliyoruz.
        return self.hbox
```

```
def main(self):
    gtk.main()
```

```
uyg = Uygulama()
uyg.main()
```

Yukarıdaki örnekte yaptığımız açıklamaları dikkatle inceleyin. Özellikle `kutu_olustur()` fonksiyonunun nasıl bir işlev gördüğüne dikkat edin. Bu fonksiyon, resim ve etiket öğelerini bir `HBox()` haznesi içine yerleştirip tek bir paket halinde bize veriyor. Biz de bu paketi `__init__()` fonksiyonu içinde "self.btn" olarak tanımladığımız düğmeye yerleştiriyoruz. Böylece düğme içinde hem resmi hem de etiketi aynı anda gösterme imkanına kavuşuyoruz.

Bu örnekte, `gtk.Button()` adlı pencere aracının `HBox()` haznesini içine alabildiğini görüyoruz. Daha önceki derslerimizde `HBox()` haznesini hep doğrudan pencere üzerine veya başka bir hazne içine yerleştiriyorduk. `Pygtk`'de `gtk.Button()` adlı pencere aracı da aslında bir haznedir. Normalde bu pencere aracı içinde yalnızca bir etiket bulunur. Ama gördüğünüz gibi, `gtk.Button()` aslında `HBox()` haznesini de içinde barındırma kapasitesine sahip.

Bu bilgiden yola çıkarak, düğme içindeki etiket ve resmin konumlarıyla rahatlıkla oynayabilirsiniz. Mesela "Hbox()" yerine "VBox()" haznesini kullanacak olursanız resim üstte, etiket altta olacaktır...

Dilerseniz bir `gtk.Button()` aracı içine tablo dahi yerleştirebilirsiniz. Yani yukarıdaki örneği temel alarak şöyle bir şey yazabilirsiniz:

```
import pygtk
pygtk.require20()
import gtk

class Uygulama(object):
    def __init__(self):
        self.pencere = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.pencere.connect("delete_event", gtk.main_quit)

        self.kutu = self.kutu_olustur("istihzacom.png", "www.istihza.com")

        self.btn = gtk.Button()
        self.btn.add(self.kutu)

        self.pencere.add(self.btn)

        self.pencere.show_all()

    def kutu_olustur(self, rsm, etk):
        self.tablo = gtk.Table(2, 2)

        self.resim = gtk.Image()
        self.resim.set_from_file(rsm)

        self.etiket = gtk.Label(etk)

        self.tablo.attach(self.resim, 0, 1, 0, 1)
        self.tablo.attach(self.etiket, 0, 1, 1, 2)

        self.resim.show()
        self.etiket.show()
        return self.tablo
```

```
def main(self):  
    gtk.main()
```

```
uyg = Uygulama()  
uyg.main()
```

Bu örneklerde temel olarak yaptığımız şey, resim ve etiketi tek bir hazne içine yerleştirdikten sonra (bu hazne HBox olabilir, VBox olabilir, hatta gördüğümüz gibi Table dahi olabilir...), alıp bunu bir düğme içine sokmaktan ibaret...