
# Tkinter Kılavuzu

***Sürüm 2.x***

**Fırat Özgül (istihza)**

17/04/2011

# İÇİNDEKİLER

BÖLÜM 1

# Grafik Arayüz Tasarımı // Temel Bilgiler

Bir programlama dilini ne kadar iyi bilirseniz bilin, bu programlama diliyle ne kadar güçlü programlar yazarsanız yazın, eğer programınız kullanışlı bir arayüze sahip değilse programınızın kullanıcı düzeyinde pek ilgi çekmeyeceği kesin... Belki bilgisayar kurtları komut satırından çalışan programları kullanmaktan rahatsız olmayacaktır, hatta sizi takdir de edecektir, ama "son kullanıcı" denen kesime hitap etmediğiniz sürece dikkati çekmeniz güç... Bu dikkati çekme meselesi de çoğunlukla kullanıcıya pırıltılı arayüzler sunularak aşılabiliyor. Diyelim ki komut satırı üzerinden harika işler yapan muhtemeşem bir program yazdınız ve programınızı kullanıma sundunuz... "Son kullanıcı", programınızı kurup çalıştırmayı denediğinde vereceği ilk tepki: "Ama bu program çalışmıyor!" şeklinde olacaktır... Kızsak da kızmasak da ne yazık ki durum bu.. Dost acı söyler!

Madem arayüz denen şey bu kadar önemli şu halde biz de bu bölümde Python'da nasıl grafik arayüzler tasarlayabileceğimizi inceleyelim.

Python'da arayüz tasarımı için birden fazla seçeneğimiz var. Bunları bir çırpıda sıralayacak olursak, seçeneklerimizin şunlar olduğunu görürüz:

- PyGTK
- PyQt
- wxPython
- Tkinter

Arayüz tasarlamak üzere geliştirilmiş yukarıdaki alternatifler içinde biz Python tarafından resmi olarak desteklenen Tkinter'i kullanacağız. Peki neden ötekiler değil de Tkinter?

- Öncelikle Tkinter'in öteki seçeneklere karşı en büyük üstünlüğü Python dağıtımı ile birlikte gelmesidir.
- Dolayısıyla Tkinter Python tarafından resmi olarak desteklenen bir arayüz tasarım takımıdır.
- Tkinter hem Windows'ta hem de GNU/Linux üzerinde rahatlıkla çalışabilir. Aslında artık günümüzde öteki arayüz takımları da birden fazla platform üzerinde çalışabilmektedir. Ama dağıtımla birlikte geldiği için Tkinter ötekilere göre çok daha kolay ve rahat bir

şekilde kurulup kullanıma hazır hale getirilebilir. Eğer Windows kullanıyorsanız, Tkinter sizde zaten Python'la beraber sisteminize kurulmuştur. Eğer GNU/Linux kullanıyorsanız, kullandığınız dağıtımın paket yöneticisini kullanarak tkinter paketini sisteminize kurabilirsiniz. Farklı GNU/Linux dağıtımlarında bu paketin adı farklı olabilir. Paket yöneticiniz içinde "tk" şeklinde arama yaparsanız muhtemelen doğru paketi bulabilirsiniz. Paket adı python-tk veya python2.x-tk gibi bir şey olacaktır...

- Tkinter oldukça sağlam, kararlı ve oturmuş bir arayüz takımıdır. Üstelik kullanımı da oldukça basittir. Öteki seçeneklere göre çok daha temiz ve sade bir sözdizimi vardır.
- Tkinter'in tek eksiği, bu arayüz takımıyla yapılan programların biraz "çirkin" görünmesidir. Ama mesela resim düzenleme yazılımlarını kullanarak Tkinter'in cemalini düzeltmek o kadar da zor bir iş değildir.
- Ben şahsen, Python programcılarının Tkinter'i kullanmayacak bile olsalar en azından temelini bilmeleri gerektiğini düşünüyorum. Tabii ki sizler Tkinter'i öğrendikten sonra öteki seçenekleri de incelemek isteyebilirsiniz...

Aslında Tkinter Python içinde bir modül... Neyse... Lafı daha fazla uzatmadan işe koyulalım:

## 1.1 Pencere Oluşturmak

Arayüz denilen şey tabii ki penceresiz olmaz. Dolayısıyla arayüz programlamanın ilk adımı çalışan bir pencere yaratmak olacaktır.

Başta söylediğimiz gibi, arayüz tasarlarken Tkinter modülünden faydalanacağız. Daha önceki yazılardan, Python'da nasıl modül "import" edildiğini hatırlıyorsunuz. Şimdi hemen Python'un etkileşimli kabuğunda şu komutu veriyoruz:

```
from Tkinter import *
```

Eğer hiçbir şey olmadan alt satıra geçildiyse sorun yok. Demek ki sizde Tkinter modülü yüklü. Ama eğer bu komutu verdiğinizde alt satıra düşmek yerine bir hata mesajı alıyorsanız, sebebi gerekli modülün, yani Tkinter'in sizde yüklü olmamasıdır. Eğer hal böyleyse yapacağımız işlem çok basit: Kullandığımız GNU/Linux dağıtımının paket yöneticisinden Tkinter modülünü içeren paketi gidip kuracağız. Benim şu anda kullandığım dağıtım olan Kubuntu'da bu paketin adı "python2.5-tk". Sizde de paketin adı buna benzer bir şey olmalı. Mesela Pardus'ta Tkinter modülünü kullanmak için kurmanız gereken paketin adı "python-tk". Dediğim gibi, eğer Windows kullanıyorsanız, sizde Tkinter modülü zaten yüklüdür. Windows kullanıcılarının Tkinter modülünü kullanabilmesi için ayrı bir şey yüklemesine gerek yok...

Modülle ilgili kısmı sağ salim atlattıysanız, artık şu komutu verebilirsiniz:

```
Tk()
```

Eğer buraya kadar herhangi bir hata yapmadıysak, ekrana pırıl pırıl bir arayüz penceresi zıpladığını görmemiz gerekiyor. Çok güzel, değil mi? Ne yalan söyleyeyim, ben bu pencereyi ilk gördüğümde çok heyecanlanmıştım... Neyse... Konumuza dönelim... Burada komutumuzu sadece "Tk()" şeklinde kullanabilmemiz, yukarıda Tkinter modülünü içe aktarma şeklimizden kaynaklanıyor. Yani Tkinter modülünü "from Tkinter import *" şeklinde içe aktardığımız için komutu "Tk()" şeklinde yazmamız yeterli oluyor. Eğer Tkinter modülünü "import Tkinter" şeklinde içe aktarsaydık, sadece "Tk()" dememiz yeterli olmayacaktı. O zaman "Tk()" yerine "Tkinter.Tk()" dememiz gerekirdi... İsterseniz modülü "import Tkinter" şeklinde içe aktarıp, sadece "Tk()" yazdığınızda ne tür bir hata aldığınızı kontrol edebilirsiniz. Neyse, konuyu daha fazla dağıtmadan yolumuza devam edelim:

Gördüğünüz gibi “Tk()” komutuyla önümüze atlayan pencere, bir pencerenin sahip olması gereken bütün temel özelliklere sahip. Yani pencerenin sağ üst köşesinde pencereyi kapatmaya yarayan bir çarpı işareti, onun solunda pencereyi büyütüp küçültmemizi sağlayan karecik ve en solda da pencereyi görev çubuğuna indirmemizi sağlayan işaret bütün işlevselliğiyle karşımızda duruyor. Ayrıca farkedeceğiniz gibi bu pencereyi istediğimiz gibi boyutlandırmamız da mümkün.

Bu komutları hangi platformda verdiğinize bağlı olarak, pencerenin görünüşü farklı olacaktır. Yani mesela bu komutları Windows’ta verdiyseniz, Windows’un renk ve şekil şemasına uygun bir pencere oluşacaktır. Eğer Gnome kullanıyorsanız, pencerenin şekli şemali, Gnome’nin renk ve şekil şemasına uygun olarak KDE’dekinden farklı olacaktır...

Yukarıda verdiğimiz “Tk()” komutuyla aslında Tkinter modülü içindeki “Tk” adlı sınıfı çağırmış olduk. Bu sınıfın neye benzediğini merak eden arkadaşlar /usr/lib/python2.5/lib-tk/ klasörü içindeki Tkinter.py modülü içinde “class Tk” satırını arayarak bunun ne menem bir şey olduğuna bakabilir (Gereksiz bilgi: “ne menem” ifadesinin doğrusu aslında “ne mene”... Ama hep “ne menem” biçiminde kullanıla kullanıla, yanlış kullanım doğru kullanım halini almış...) Bu arada herhangi bir hayal kırıklığı yaşamak istemiyorsanız, yukarıdaki “from Tkinter import *” ve “Tk()” komutlarını verirken büyük-küçük harfe dikkat etmenizi tavsiye ederim. Çünkü “Tkinter” ile “tkinter” aynı şeyler değildir...

Eğer bir arayüz oluşturacaksak sürekli komut satırında çalışamayız: Mutlaka yukarıda verdiğimiz komutları bir yere kaydetmemiz gerekir. Bunun için hemen boş bir belge açıp içine şu satırları ekleyelim:

```
from Tkinter import *
Tk()
```

Eğer bu dosyayı bu şekilde kaydeder ve çalıştırmayı denersek açılmasını beklediğimiz pencere açılmayacaktır. Burada komut satırından farklı olarak ilave bir satır daha eklememiz gerekiyor. Yani kodumuzun görünümü şu şekilde olmalı:

```
from Tkinter import *
Tk()
mainloop()
```

Belgemizin içine yukarıdaki kodları yazdıktan sonra bunu ”.py” uzantılı bir dosya olarak kaydediyoruz ve normal Python dosyalarını nasıl çalıştırıyorsak öyle çalıştırıyoruz...

İsterseniz yukarıdaki kodu biraz derleyip toparlayalım. Çünkü bu şekilde kodlarımız çok sefil görünüyor. Yukarıdaki kodlar aslında bir program içinde şu şekilde görünecektir:

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

from Tkinter import *
pencere = Tk()
mainloop()
```

Bu kodlar içinde gördüğümüz tek yenilik en sondaki “mainloop()” satırı. Bu satır program içinde bir “döngü” yaratarak, kendisinden önce gelen kodlarla belirlenen özelliklere göre bir pencere oluşturulmasını sağlıyor. Bu satırla yaratılan “döngü” sayesinde de oluşturulan pencere (aksi belirtilmedikçe veya kullanıcı pencereyi kapatmadıkça) ekranda hep açık kalabiliyor. Eğer bu son satırı yazmazsak, penceremiz yine de oluşur, ama biz pencereyi ekranda göremeyiz!.. Bu arada buraya küçük bir not düşelim. Eğer yazdığınız programı Python’la birlikte gelen bir metin düzenleyici olan IDLE içinden çalıştırıyorsanız, yazdığınız programın son satırına “mainloop()” ifadesini yerleştirmeseniz bile programınız hatasız bir şekilde çalışacaktır. Neden? Çünkü IDLE da Tkinter ile yazılmış bir uygulamadır. Dolayısıyla IDLE’nin de kendisine ait bir “mainloop()”

satırı vardır. Yani siz kendi programınız içine mainloop()'u koymasanız bile IDLE'nin kendi mainloop()'u sizin programınızın da çalışmasını sağlayacaktır. Ancak mainloop() satırını koymadığınız halde programınız IDLE içinden çalışsa da başka yerde çalışmayacaktır. O yüzden bu mainloop() satırını mutlaka yerine koymalıyız...

Dilerseniz kodlarımızın derli toplu hale getirilmiş biçimine şöyle bir bakalım:

İlk satırı anlatmaya gerek yok. Bunun ne olduğunu zaten biliyorsunuz. Ayrıca zaten bildiğiniz gibi, bu satır sadece GNU/Linux kullanıcılarına yöneliktir. Windows kullanıcıları bu satırı yazmasalar da olur.

İkinci satırımız, yine bildiğiniz gibi, programımızı utf-8 ile kodlamamızı sağlıyor... Windows kullanıcılarının "utf-8" yerine "cp1254" kullanması gerekebilir...

Üçüncü satırda Tkinter modülünü programımızın içine aktarıyoruz (Yani "import" ediyoruz...).

Sonraki satırda, daha önce komut satırında kullandığımız "Tk()" sınıfını bir değişkene atadık. Nesne Tabanlı Programlama konusundan hatırlayacağınız gibi, bir sınıfı değişkene atama işlemine Python'da "instantiation", yani "örnekleme" adı veriliyordu. İşte biz de burada aslında Tkinter modülü içindeki Tk() sınıfını örneklemiş olduk...

Son satırdaki "mainloop()" ise yazdığımız pencerenin ekranda görünmesini sağlıyor.

Daha önceki derslerimizde "Nesne Tabanlı Programlama" konusunu işlerken, sınıflı yapıların genellikle arayüz programlamada tercih edildiğini söylemiştik. İsterseniz yukarıda yazdığımız minik kodun sınıflar kullanılarak nasıl yazılabileceğine bakalım:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        pass

pencere = Tk()
mainloop()
```

Tabii yukarıdaki sınıf örneğinin pek bir anlamı yok... Yaptığı tek şey ekrana boş bir pencere getirmek. Üstelik yukarıdaki örnek, kod yapısı olarak mükemmel olmaktan uzak bir iskeletten ibaret... Bütün eksikliklerine rağmen yukarıdaki örnek bize bir Tkinter programının sınıflı bir yapı içinde temel olarak nasıl görüneceğini gösteriyor. Biz sayfalar ilerledikçe bu sınıfımızı daha iyi ve standart bir hale getireceğiz. Bu arada sınıfımızı oluştururken, Python'un yeni sınıf yapısına göre "object" adlı sınıfı miras aldığımıza dikkat edin.

Şimdiye kadar Python'da Tkinter yardımıyla boş bir pencere oluşturmayı öğrendik... Bu heyecan içinde bu kodları elli defa çalıştırıp elli defa boş bir pencere oluşmasını izlemiş olabilirsiniz... Ama bir süre sonra boş pencere sizi sıkmaya başlayacaktır. O yüzden, gelin isterseniz şimdi de bu boş pencereyi nasıl dolduracağımızı öğrenelim. Hemen yeni bir dosya açarak içine şu satırları ekliyoruz:

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

from Tkinter import *

pencere = Tk()
```

İlk iki satırın ne olduğunu zaten biliyorsunuz, o yüzden açıklamaya gerek yok. Ondan sonra gelen satırda da bildiğiniz gibi Tkinter modülünü çağırdık. Bir alt satırda ise yaptığımız şey daha önce gördüğümüz "Tk()" komutuna bir ad vermekten, yani daha teknik bir dille ifade etmek gerekirse Tk() sınıfını örneklemekten ibaret... Biz "örnekleme" (instantiation) işlemi için burada "pencere" adını tercih ettik.. Tabii ki siz isterseniz emmoğlunuzun adını da verebilirsiniz... Hiç problem değil... Hatta hiç isim vermeseniz de olur. Ama kullanım kolaylığı ve performans açısından isimlendirmenin zararını değil, yararını görürsünüz... Şimdi kodlamaya kaldığımız yerden devam ediyoruz:

```
etiket = Label(text="Elveda Zalim Dünya!")
etiket.pack()

mainloop()
```

Burada "etiket" adlı yeni bir değişken oluşturduk. Her zamanki gibi bu "etiket" adı zorunlu bir isimlendirme değil. İstediğiniz ismi kullanmakta serbestsiniz. Ancak değişkenin ismi önemli olmasa da, içeriği önemli... "pencere = Tk()" komutu yardımıyla en başta oluşturduğumuz pencereyi bir kutu olarak düşünürsek, "etiket" adıyla tanımladığımız değişken de bu kutu üzerine yapıştırılacak bir etiket olarak düşünülebilir. Bahsettiğimiz bu kutunun üzerine etiketini yapıştırmak için "Label" adlı özel bir işlevden faydalanıyoruz. (Bu arada "label" kelimesi İngilizce'de "etiket" anlamına geliyor) "Label" ifadesini bu adla aklımıza kazımamız gerekiyor. Daha sonra bu Label ifadesine parametre olarak bir metin işliyoruz. Metnimiz "Elveda Zalim Dünya!". Metnimizin adı ise "text". "Label" ifadesinde olduğu gibi, "text" ifadesi de aklımıza kazımamız gereken ifadelerden birisi. Bunu bu şekilde öğrenmemiz gerekiyor. Kendi kendinize bazı denemeler yaparak bu satırda neleri değiştirip neleri değiştiremeyeceğinizi incelemenizi tavsiye ederim...

Bir alt satırda "etiket.pack()" ifadesini görüyoruz. Bu satır, yukarıdaki satırın işlevli hale gelmesini sağlıyor. Yani kutu üzerine etiketi yapıştırdıktan sonra bunu alıp kargoya vermemize yarıyor. Eğer bu satırı kullanmazsak bir güzel hazırlayıp etiketlediğimiz kutu kargoya verilmemiş olacağı için kullanıcıya ulaşmayacaktır.

En sondaki "mainloop()" ifadesini ise zaten tanıyorsunuz: Yukarıda yazdığımız bütün kodların döngü halinde işletilerek bir pencere şeklinde sunulmasını sağlıyor.

Kodlarımızın son hali şöyle olmalı:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Elveda Zalim Dünya!")
etiket.pack()

mainloop()
```

Gördüğünüz gibi yine gayet şık, içinde "Elveda Zalim Dünya!" yazan bir pencere elde ettik. Alıştırma olsun diye, yukarıda yazdığımız kodlara çok benzeyen başka bir örnek daha yapalım:

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

from Tkinter import *

pencere = Tk()
```

```
etiket = Label(text = "Resimler klasörünü silmek istediğinizden emin misiniz?")
etiket.pack()

mainloop()
```

Şimdi isterseniz yukarıdaki kodları sınıflı yapı içinde nasıl gösterebileceğimize bakalım:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.etiket = Label(text="Dosyayı silmek istediğinize emin misiniz?")
        self.etiket.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Sınıfsız yapıdaki kodların sınıflı yapı içinde nasıl değişikliklere uğradığına dikkat edin. self'leri nasıl kullandığımızı inceleyin. Burada "pencere = Tk()" satırıyla Tk() sınıfını örnekledikten sonra "uyg = Uygulama()" satırıyla da, yukarıdaki kodların içinde yazdığımız "Uygulama" adlı sınıfımızı örnekliyoruz. En son satıra da "mainloop()" ifadesini yerleştirmeyi unutmuyoruz. Aksi halde oluşturduğumuz pencere ekranda görünmeyecektir...

Yukarıdaki sınıflı kodu isterseniz şu şekilde de yazabilirsiniz:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.araclar()

    def araclar(self):
        self.etiket = Label(text="Dosyayı silmek istediğinizden emin misiniz?")
        self.etiket.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Gördüğünüz gibi burada "etiket" öğesini ayrı bir fonksiyon olarak belirledik. Programınız büyüyüp birbirinden farklı öğeler içermeye başladığı zaman bu öğeleri yukarıdaki şekilde farklı bölümlere ayırmak, kodlarınızı daha düzenli bir hale getirecek, ileride kodlarınızın bakımını yapmayı kolaylaştıracaktır.

Eğer bu sınıflı kodları anlamakta zorluk çekiyorsanız, yukarıda da bağlantısını verdiğimiz "Nesne Tabanlı Programlama" konusunu gözden geçirebilirsiniz. Bu arada hatırlatalım, Tkinter'i kullanmak için sınıflı yapıları bilmek zorunda değilsiniz. Ben de zaten bu sayfalarda sınıflı ve sınıfsız kullanımları bir arada vereceğim. Eğer sınıflı yapıyı anlamazsanız, sadece sınıfsız örneklerle de ilerleyebilirsiniz. Ama yine de benim size tavsiyem, sınıfları öğrenmeniz yönünde olacaktır. Çünkü internet üzerinde belli bir seviyeden sonra bu sınıflı yapılar hep karşınıza çıkacaktır. Yani siz kullanmasanız da, sınıfları kullanmayı bilmeniz sizin için faydalı olacaktır. Bu arada dikkat ederseniz, yukarıda verdiğimiz sınıflı son iki kod, daha önce verdiğimiz sınıflı kod-

dan daha yetkin. Dediğimiz gibi, yeni özellikler öğrendikçe, kodlarımızı adım adım standartlara daha yakın hale getireceğiz.

Bu derste Tkinter'i kullanarak basit bir pencereyi nasıl oluşturabileceğimizi öğrendik. Örneklerimizi hem sınıflı hem de sınıfsız kodlarla gösterdik. Tabii ki sınıflı kodlar bu tür küçük örneklerde çok lüzumlu değildir. Ama büyük bir projede çalışırken sınıflı yapı size esneklik ve vakit kazandıracak, kodlarınızın bakımını daha kolay yapmanızı sağlayacaktır.

Bu açıklamaları da yaptığımıza göre başka bir bölüme geçebiliriz.

## 1.2 Pencere Başlığı

Bildiğiniz gibi, gündelik yaşamda karşımıza çıkan pencerelerde genellikle bir başlık olur. Mesela bilgisayarınız size bir hata mesajı gösterirken pencerenin tepesinde "Hata" ya da "Error" yazdığını görürsünüz. Mesela şu anda baktığınız pencerenin en tepesinde, "Grafik Arayüz Tasarımı // Temel Bilgiler – Tkinter 2.x Belgelendirme Çalışması" yazıyor. Eğer istersek biz de oluşturduğumuz pencerelerin tepesine böyle ifadeler yerleştirebiliriz. İşte bu işi yapmak, yani penceremizin başlığını değiştirmek için "title()" adlı metottan yararlanmamız gerekiyor. Yani bunun için yazdığımız kodlara şuna benzer bir satır eklemeliyiz:

```
pencere.title("Başlık")
```

Hemen bu özelliği bir örnekle gösterelim ki bu bilgi kafamızda somutlaşsın:

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

from Tkinter import *

pencere = Tk()

baslik = pencere.title("Hiç bir işe yaramayan bir pencereyim ben...")

etiket= Label(text="...ama en azından bir başlığım var!")
etiket.pack()

mainloop()
```

Gördüğünüz gibi pencereye başlık eklemek son derece kolay... Tek bir satırla işi halledebiliyoruz. Üstelik herhangi bir "paketleme" işlemi yapmamız da gerekmiyor.

Dilerseniz şimdi yukarıdaki kodları sınıflı yapı içinde gösterelim:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.etiket = Label(text="Sabit diskiniz siliniyor...")
        self.etiket.pack()

        self.baslik = pencere.title("Çok Önemli Uyarı!")

pencere = Tk()
```

```
uyg = Uygulama()
mainloop()
```

Bu kodlar içindeki tek yenilik, “self.baslik = .....” satırıdır. Geri kalan kısımları zaten önceki örneklerden de biliyorsunuz.

Pencerelerle ilgili daha karmaşık özelliklere geçmeden önce isterseniz burada bir durup şimdiye kadar öğrendiklerimizi çok kısa bir şekilde özetleyelim:

1. Tkinter’le arayüz tasarımı yapabilmek için öncelikle “from Tkinter import *” diyerek gerekli modülü yüklememiz gerekiyor.
2. Ardından “Tk()” yardımıyla boş bir pencere oluşturuyoruz. Kullanım kolaylığı ve performans açısından “Tk()” ifadesini isimlendirerek bir değişkene atamayı (yani Tk() sınıfını örneklemeyi) tercih edebilirsiniz. Örneğin; kebap = Tk(). Performans açısından sınıflarımızı örneklemek gerekir, çünkü Nesne Tabanlı Programlama adlı dersten hatırlayacağınız gibi, örneklenmeyen sınıflar çöp toplama (garbage collection) adı verilen bir sürece tabi tutulacaktır...
3. Bundan sonra pencere içine yazmak istediğimiz ifadeyi içeren bir etiket hazırlamamız gerekiyor. Bu etiketi alıp “Tk()” isimli kutunun üzerine yapıştıracağız. Bu iş için bize “Label()” adlı araç yardımcı olacak. Bu fonksiyonun parametresi olarak yazacağımız “text” adlı değişkene istediğimiz metni girebiliriz. Mesela: oyun = Label(text=”Bilgisayarınıza virüs bulaşmış!”)
4. Etiketimizi hazırladıktan sonra paketleyip kargoya vermemiz gerekiyor. Grafik arayüzün kullanıcıya ulaşması için bu gerekli... Bunun için “xxx.pack()” ifadesini kullanacağız. Buradaki “xxx” yerine, bir üst satırda hazırladığımız etiket için kullandığımız ismi yazacağız. Mesela: oyun.pack()
5. Bütün bu yazdığımız kodların işlevli hale gelebilmesi için ise en sona “mainloop()” ifadesini yerleştirmemiz gerekiyor.
6. Eğer hazırladığımız pencerenin bir de başlığı olsun istiyorsak “title()” adlı metottan yararlanıyoruz.

Şu ana kadar yapabildiğimiz tek şey bir pencere oluşturup içine bir metin eklemek ve pencerede basitçe bir başlık oluşturmak. Ama tabii ki bir süre sonra bu da bizi sıkacaktır. Hem sadece bu özellikleri öğrenerek heyecan verici arayüzler hazırlamamız pek mümkün değil... En fazla, arkadaşlarımıza ufak tefek eşek şakaları yapabiliriz bu öğrendiklerimizle!

Yeri gelmişken, hazırladığımız programı, simgesi üzerine çift tıklayarak nasıl çalıştıracağımızı görelim şimdi. Gerçi daha önceki bölümlerden hatırlıyoruz bu işin nasıl yapılacağını, ama biz yine de tekrar hatırlatalım:

**GNU/Linux kullanıcıları:**

1. .py uzantılı dosyamızı hazırlarken ilk satıra “#!/usr/bin/env python” ifadesini mutlaka yazıyoruz.
2. Kaydettiğimiz .py uzantılı dosyaya sağ tıklayarak “özellikler”e giriyoruz
3. Burada “izinler” sekmesinde “çalıştırılabilir” ifadesinin yanındaki kutucuğu işaretliyoruz
4. “Tamam” diyerek çıkıyoruz.

**Windows kullanıcıları:**

1. Windows kullanıcıları .py uzantılı dosyaya çift tıklayarak programı çalıştırabilir. Ancak bu şekilde arkada bir de siyah DOS ekranı açılacaktır.

2. O DOS ekranının açılmaması için, kodlarımızı barındıran dosyamızı .py uzantısı ile değil .pyw uzantısı ile kaydediyoruz.
3. Dosya uzantısını .pyw olarak belirledikten sonra dosya üzerine çift tıklayarak Tkinter programınızı çalıştırabilirsiniz.
4. Windows kullanıcıları dosya kodlaması için "*#--coding:cp1254" veya "*#--coding:utf-8-*-" satırlarını kullanabilir.

Böylece artık programımızın simgesi üzerine çift tıklayarak arayüzümüzü çalıştırabiliriz.

## 1.3 Renkler

Dikkat ettiyseniz şimdiye dek oluşturduğumuz pencerelerde yazdığımız yazılar hep siyah renkte. Tabii ki, siz oluşturduğunuz pencereler içindeki metinlerin her daim siyah olmasını istemiyor olabilirsiniz. Öntanımlı renkleri değiştirmek sizin en doğal hakkınız. Tkinter'le çalışırken renklerle oynamak oldukça basittir. Renk işlemleri Tkinter'de birtakım ifadeler vasıtasıyla hallediliyor. Python dilinde bu ifadelere "seçenek" (option) adı veriliyor. Mesela daha önce öğrendiğimiz ve etiket içine metin yerleştirmemizi sağlayan "text" ifadesi de bir "seçenek"tir... Şimdi öğreneceğimiz seçeneklerin kullanımı da tıpkı bu "text" seçeneğinin kullanımına benzer.

## 1.4 fg seçeneği

Diyelim ki pencere içine yazdığımız bir metni, daha fazla dikkat çekmesi için kırmızı renkle yazmak istiyoruz. İşte bu işlem için kullanmamız gereken şey "fg" seçeneği... Bu ifade İngilizce'deki "foreground" (önplan, önalan) kelimesinin kısaltması oluyor. Hemen bir örnek verelim:

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

from Tkinter import *

pencere = Tk()
pencere.title("Hata!")

etiket = Label(text = "Hata: Bilinmeyen Hata 404", fg="red")
etiket.pack()

mainloop()
```

Gördüğünüz gibi yaptığımız tek şey "fg" seçeneğini etiketin içine yerleştirmek. Burada kırmızı renk için kullandığımız kelime, İngilizce'de "kırmızı" anlamına gelen "red" sözcüğü... Yani "insan" dilinde kullanılan renkleri doğrudan Tkinter'de de kullanabiliyoruz. Tabii bu noktada biraz İngilizce bilmek işinizi bir hayli kolaylaştıracaktır. Tkinter'de kullanabileceğimiz, İngilizce renk adlarından birkaç tanesini hemen sıralayalım:

red = kırmızı

white = beyaz

black = siyah

yellow = sarı

blue = mavi

brown = kahverengi

green = yeşil

pink = pembe

Tkinter'de kullanılabilecek renk adları bunlarla sınırlı değil. Eğer mevcut renk listesini görmek isterseniz, şu adrese bir bakabilirsiniz: http://www.tcl.tk/man/tcl8.3/TkCmd/colors.htm. Tabii o listede sıralanan renkler arasından istediğinizi bulup çıkarmak pek kolay değil. Aynı zamanda renklerin kendisini de gösteren bir liste herhalde daha kullanışlı olurdu bizim için... Neyse ki çaresiz değiliz... Tkinter bize, renk adlarının yanısıra renk kodlarını kullanarak da yazılarımızı renklendirebilme imkanı veriyor. Renk kodlarını bulmak için şu adresten yararlanabilirsiniz: http://html-color-codes.info. Buranın avantajı, renklerin kendisini de doğrudan görebiliyor olmamız. Mesela orada beğendiğimiz herhangi bir rengin üzerine tıkladığımızda, alttaki kutucukta o rengin kodu görünecektir. Diyelim ki renk paletinden bir renk seçip üzerine tıkladık ve alttaki kutucukta şu kod belirdi: #610B0B. Bu kodu kopyalayıp kendi yazdığımız program içinde gerekli yere yapıştırabiliriz:

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

from Tkinter import *

pencere = Tk()
pencere.title("Hata!")

etiket = Label(text = "Hata: Bilinmeyen Hata 404", fg="#610B0B")
etiket.pack()

mainloop()
```

Doğrudan renk adları yerine renk kodlarını kullanmak daha kesin sonuç verecektir. Ayrıca renk kodları size daha geniş bir renk yelpazesi sunacağı için epey işinize yarayacaktır.

Son olarak, yukarıdaki kodu sınıflı yapı içinde kullanalım:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.etiket = Label(text="Hello",fg = "red")
        self.etiket.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Veya öğeleri ayrı bir fonksiyon içinde belirtmek istersek...

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

from Tkinter import *

class Uygulama(object):
```

```
    def __init__(self):
        self.araclar()

    def araclar(self):
        self.etiket = Label(text="Hata: Bilinmeyen Hata 404", fg="#610B0B")
        self.etiket.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Gördüğünüz gibi, bir önceki sınıflı örnekten pek farkı yok. Tek yenilik fg seçeneğinin de kodlar arasına eklenmiş olması...

## 1.5 bg seçeneği

Bu da İngilizce'deki "background" (arkaplan, arkaalan) kelimesinin kısaltması. Adından da anlaşılacağı gibi pencereye yazdığımız metnin arkaplan rengini değiştirmeye yarıyor. Kullanımı şöyle:

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

from Tkinter import *

pencere = Tk()
pencere.title("Hata!")

etiket = Label(text = "Hata: Bilinmeyen Hata 404", bg="blue")
etiket.pack()

mainloop()
```

Yukarıda verdiğimiz renkleri bu seçenek için de kullanabilirsiniz. Ayrıca bu seçenek bir önceki "fg" seçeneğiyle birlikte de kullanılabilir:

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

from Tkinter import *

pencere = Tk()
pencere.title("Hata!")

etiket = Label(text = "Hata monitörle sandalyenin tam arasında!",
               fg="white",
               bg="black")
etiket.pack()

mainloop()
```

Son örnekte renk kodları yerine renk adlarını kullandığımıza dikkat edin.

## 1.6 Yazı Tipleri (Fonts)

Tkinter bize renklerin yanısıra yazı tipleriyle de oynama imkanı veriyor. Hazırladığımız pencerelerdeki yazı tiplerini değiştirmek için, tıpkı renklerde olduğu gibi, yine bazı seçeneklerden faydalanacağız. Burada kullanacağımız seçeneğin adı, "font". Bunu kullanmak için kodlarımız arasına şuna benzer bir "seçenek" eklememiz gerekiyor:

```
font= "Helvetica 14 bold"
```

Burada tahmin edebileceğiniz gibi, "Helvetica" yazı tipini; "14" yazı boyutunu; "bold" ise yazının kalın olacağını gösteriyor. Örnek bir kullanım şöyledir:

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

from Tkinter import *
pencere = Tk()
etiket = Label(text="Merhaba Dostlar!", font="Times 15 italic")
etiket.pack()
mainloop()
```

Burada yazı tipi olarak "Times"; yazı boyutu olarak "15"; yazı biçimi olarak ise "italic", yani "yatık" biçim seçtik. "font" seçeneği ile birlikte kullanabileceğimiz ifadeler şunlardır:

italic = yatık

underline = altı çizili

bold = kalın

overstrike = kelimenin üstü çizili

Yazı tipi olarak neleri kullanabileceğinizi, daha doğrusu kendi sisteminizde hangi yazıtiplerinin kurulu olduğunu ise şöyle öğrenebilirsiniz:

```
from Tkinter import *
import tkFont

pencere = Tk()
yazitipleri = list(tkFont.families() )
yazitipleri.sort()
for i in yazitipleri:
    print i
```

Bu listede birden fazla kelimeden oluşan yazı tiplerini gösterirken kelimeleri birleşik yazmamız gerekiyor. Mesela "DejaVu Sans"ı seçmek için "DejaVuSans" yazmamız lazım...

## 1.7 Metin Biçimlendirme

Daha önceki yazılarımızda öğrendiğimiz metin biçimlemeye yarayan işaretleri Tkinter'de de kullanabiliyoruz. Yani mesela şöyle bir şey yapabiliyoruz:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
from Tkinter import *
pencere = Tk()
etiket = Label(text="Merhaba Dostlar!\n\tNasılsınız?", font="DejaVuSans 15 italic")
```

```
etiket.pack()
mainloop()
```

Gördüğünüz gibi, daha önceki yazılarımızda öğrendiğimiz “\n” ve “\t” kaçış dizilerini burada da kullanabiliyoruz.

## 1.8 İmleçler

Arayüzlerinizi oluştururken farklı imleç şekilleri kullanmak da isteyebilirsiniz. Bunun için kullanacağımız seçenek “cursor”. Mesela:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
from Tkinter import *
pencere = Tk()
etiket = Label(text="Deneme 1,2,3...", cursor="bottom_side")
etiket.pack()
mainloop()
```

Burada, imleci pencere üzerine getirdiğimizde imlecin ters ok şekli aldığını görürüz. Kullanılabilecek imleç isimleri için şu sayfaya bakabilirsiniz: http://www.dil.univ-mrs.fr/~garreta/PythonBBSG/docs/Tkinter_ref.pdf

Burada 10. ve 11. sayfalarda hoşunuza gidebilecek imleç isimlerini seçebilirsiniz. Bunun dışında yararlanabileceğiniz başka bir kaynak da şudur: http://www.tcl.tk/man/tcl8.4/TkCmd/cursors.htm

## 1.9 Pencere Boyutu

Diyelim ki içinde sadece bir “etiket” barındıran bir pencere oluşturduk:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Hata!")
etiket.pack()

mainloop()
```

Bu kodları çalıştırdığımızda karşımıza çıkan pencere çok küçük. O yüzden pencerenin tepesindeki eksi, küçük kare ve çarpı işaretleri görünmüyor. Bunları görebilmek için pencereyi elle boyutlandırmamız gerekiyor. Tabii bu her zaman arzu edilecek bir durum değil. Bu gibi durumları engelleyebilmemiz ve penceremizi istediğimiz boyutta oluşturabilmemiz için, Tkinter bize gayet kullanışlı bir imkan sunuyor:

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

from Tkinter import *
```

```
pencere = Tk()
pencere.geometry("100x100+15+100")

etiket = Label(text="Hata!")
etiket.pack()

mainloop()
```

Gördüğünüz gibi, burada pencere.geometry ifadesinin karşısına birtakım sayılar ekleyerek penceremizi boyutlandırıyoruz. İlk iki rakam (100x100) penceremizin 100x100 boyutunda olduğunu; son iki rakam ise (+15+100) penceremizin ekrana göre soldan sağa doğru 15. pozisyonda; yukarıdan aşağıya doğru ise 100. pozisyonda açılacağını gösteriyor. Yani bu satır sayesinde penceremizin hem boyutunu hem de konumunu değiştirmiş oluyoruz. Bu dört rakamı değiştirerek kendi kendinize denemeler yapmanızı tavsiye ederim.

Şimdi son bir özellikten bahsedip bu konuyu kapatalım... Gördüğünüz gibi oluşturduğumuz bir pencere ek bir çabaya gerek kalmadan bir pencerenin sahip olabileceği bütün temel nitelikleri taşıyor. Buna pencerenin kullanıcı tarafından serbestçe boyutlandırılabilmesi de dahil... Ancak bazı uygulamalarda bu özellik anlamsız olacağı için (mesela hesap makinelerinde) kullanıcının pencereyi boyutlandırmasına engel olmak isteyebilirsiniz. Bu işi yapmak için şu kodu kullanıyoruz:

```
pencere.resizable(width=FALSE, height=FALSE)
```

Örnek bir uygulama şöyle olacaktır:

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

from Tkinter import *

pencere = Tk()
pencere.resizable(width=FALSE, height=FALSE)

etiket = Label(text="Deneme 1,2,3...", cursor="bottom_side")
etiket.pack()

mainloop()
```

Gördüğünüz gibi bu kodlar çalıştırıldığında ortaya çıkan pencere hiçbir şekilde boyutlandırma kabul etmiyor...

Şimdi isterseniz yukarıda verdiğimiz kodları sınıflı yapı içinde nasıl kullanacağımızı görelim:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.etiket = Label(text="Hata!")
        self.etiket.pack()

pencere = Tk()
pencere.resizable(width=FALSE, height=FALSE)
uyg = Uygulama()
mainloop()
```

Böylelikle Arayüz Tasarımı konusunun “Temel Bilgiler” kısmını bitirmiş oluyoruz. Bir sonraki bölümde Tkinter’de “Pencere Araçlarını” (widgets) kullanmayı öğreneceğiz. Ama isterseniz yeni bölüme geçmeden önce şimdiye kadar öğrendiğimiz hemen hemen bütün konuları kapsayan bir örnek yapalım.

## 1.10 == Tekrar ==

Şimdiye kadar Tkinter’in özelliklerine ve nasıl kullanılacağına ilişkin pek çok konu işledik. Yani artık temel olarak Tkinter’in neye benzediğini ve bununla neler yapabileceğimizi az çok biliyoruz. Temel Bilgiler kısmını geride bırakıp yeni bir bölüme geçmeden önce, şimdiye kadar öğrendiklerimizi bir araya getiren bir örnek yapalım. Böylece öğrendiklerimizi test etme imkanı bulmuş olacağız. Bu yazıda vereceğim örneği olabildiğince ayrıntılı bir şekilde açıklayacağım. Eğer aklınıza takılan bir yer olursa bana nasıl ulaşacağınızı biliyorsunuz...

Şimdi örneğimize geçelim... Amacımız 1 ile 100 arasında 6 tane rastgele sayı üretmek. Yalnız bu 6 sayının her biri benzersiz olacak. Yani 6 sayı içinde her sayı tek bir defa geçecek... Önce kodlarımızın tamamını verelim:

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

from Tkinter import *
import random

liste = []

pencere = Tk()
pencere.geometry("200x50+600+460")

etiket = Label(fg="white", bg="#61380B",font="Helvetica 12 bold")
etiket.pack()

for i in range(6):
    while len(liste) != 6:
        a = random.randint(1,100)
        if a not in liste:
            liste.append(a)

etiket["text"] = liste

mainloop()
```

Bu kodları her çalıştırışımızda, 1 ile 100 arasında, birbirinden farklı altı adet rastgele sayı ekranda görünecektir... Tkinter’e ilişkin kodlara geçmeden önce, isterseniz yukarıda kullandığımız, rastgele sayı üretmemizi sağlayan Python kodunu biraz açıklayalım:

- Rastgele sayılar üreteceğimiz için öncelikle Python’un “random” adlı modülünü içe aktarıyoruz. Bu modül bu tür rastgele sayı üretme işlemlerinde kullanılır.
- random modülü içindeki randint metodu bize, örneğin, 1’den 100’e kadar rastgele sayılar üretme imkanı sağlar. Ancak random.randint(1,100) dediğimiz zaman, bu kod 1’den 100’e kadar rastgele tek bir sayı üretecektir. Bizim amacımız altı adet rastgele sayı üretmek.
- Amacımız altı adet sayı üretmek olduğu için öncelikle “for i in range(6):” satırı yardımıyla altı kez işletilecek bir döngü oluşturuyoruz. (Bunun mantığıyla ilgili aklınızda şüpheler

varsa, for i in range(6): print “Merhaba Dünya!” komutunu vererek şüphelerinizi giderebilirsiniz...) Bu döngü nihai olarak random.randint() fonksiyonunu altı kez çalıştıracaktır. Böylelikle elimizde altı adet sayı olmuş olacak. Ancak üretilen bu altı sayının hepsi birbirinden farklı olmayabilir. Yani bazı sayılar birden fazla üretilebilir. Dolayısıyla “1, 5, 5, 7, 56, 64” gibi bir diziyle başbaşa kalabiliriz. Ama bizim amacımız altı sayının hepsinin birbirinden farklı olması. O yüzden bunu sağlayacak bir önlem almamız gerekiyor.

- Bu bahsettiğimiz önlemi “if a not in liste: liste.append(a)” satırlarıyla alıyoruz. Aradaki while döngüsü başka bir işe yarıyor. Onu birazdan açıklayacağız. “If a not in liste: liste.append(a)” satırlarıyla, “Eğer random.randint(1,100) komutu ile üretilen sayı liste içinde yer almıyorsa bu sayıyı listeye ekle!” emrini veriyoruz. Yani random.randint ile her sayı üretilişinde Python’un bu sayıyı listeye eklemeden önce, listede bu sayının zaten var olup olmadığını kontrol etmesini istiyoruz. Eğer üretilen sayı listede yoksa, liste.append() komutu sayesinde sayımız listeye eklenecek, yok eğer üretilen sayı listede zaten varsa o sayı listeye alınmayacaktır. Ancak burada bir sorun çıkıyor karşımıza. Eğer üretilen sayı listede zaten varsa, yukarıda yazdığımız komut gereği o sayı listeye alınmayacak, bu durumda listede 6’dan az sayı kalacaktır. Bizim, üretilen sayıların aynı olması durumunda random.randint’in altıyı tamamlayana kadar tekrar tekrar sayı üretmesini sağlamamız gerekiyor.
- İşte bu işlemi yukarıdaki while döngüsü yardımıyla yapacağız. “while len(liste) != 6:” satırı ile liste uzunluğunun 6’ya eşit olup olup olmadığını denetliyoruz. Yani Python’a şu emri veriyoruz. “liste’nin uzunluğu 6’ya eşit olmadığı sürece random.randint(1,100) komutunu işletmeye devam et!”. Böylelikle, listedeki tekrar eden sayılar yüzünden liste içeriği 6’dan küçük olduğunda, bu “while” döngüsü sayesinde Python listeyi altıya tamamlayana kadar random.randint’i çalıştırmaya devam edecektir... “if a not in liste:” satırı nedeniyle zaten listede var olan sayıları listeye ekleyemeyeceği için de, benzersiz bir sayı bulana kadar uğraşacak ve neticede bize altı adet, birbirinden farklı sayı verecektir...

Kodların Tkinter ile ilgili kısmına gelecek olursak... Esasında yukarıdaki kodlar içinde geçen her satırı anlayabilecek durumdayız. Şimdiye kadar görmediğimiz tek biçim, “etiket[”text”] = liste” satırı... Bu biçimle ilgili olarak şunu söyleyebiliriz: Tkinter’deki etiket, düğme, vb öğelerin niteliklerini, yukarıda görüldüğü şekilde, tıpkı bir sözlüğün öğelerini değiştirirmiş gibi değiştirebiliriz. Yukarıdaki yapının, sözlük öğelerini değiştirme konusunu işlerken gördüğümüz sözdiziminin aynısı olduğuna dikkat edin... Mesela bu yapıyı kullanarak şöyle bir şey yapabiliriz. Diyelim ki programımız içinde aşağıdaki gibi bir etiket tanımladık:

```
etiket = Label(fg="white", bg="#61380B",font="Helvetica 12 bold")
etiket.pack()
```

Etiketin “text” seçeneğinin bulunmadığına dikkat edin. İstersek yukarıdaki satırda “text” seçeneğini, text = “” şeklinde boş olarak da tanımlayabiliriz. Bu etiketi bu şekilde tanımladıktan sonra, programımız içinde başka bir yerde bu etiketin “text” seçeneğini, öteki özellikleriyle birlikte tek tek değiştirebiliriz. Şöyle ki:

```
etiket["text"] = "www.istihza.com"
etiket["fg"] = "red"
etiket["bg"] = "black"
etiket["font"] = "Verdana 14 italic"
```

Bu yapının Python’daki sözlüklere ne kadar benzediğine bir kez daha dikkatinizi çekmek isterim... Bu yüzden bu yapıyı öğrenmek sizin için zor olmasa gerek.

Şimdi isterseniz yukarıda verdiğimiz örneğin sınıflı yapı içinde nasıl görüneceğine bir bakalım:

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

```
import random

liste = []

class Uygulama(object):
    def __init__(self):
        self.araclar()
        self.kodlar()

    def kodlar(self):
        for i in range(6):
            while len(liste) != 6:
                a = random.randint(1,100)
                if a not in liste:
                    liste.append(a)
        self.etiket["text"] = liste

    def araclar(self):
        self.etiket = Label(fg="white", bg="#61380B",font="Helvetica 12 bold")
        self.etiket.pack()

pencere = Tk()
pencere.geometry("150x50+600+460")
uyg = Uygulama()
mainloop()
```

Gördüğünüz gibi, sınıflı yapı içinde, kodları ufak bölümlere ayırarak incelenmelerini kolaylaştırmış olduk... Artık örneğimizi de verip açıklamalarımızı da yaptığımıza göre, gönül rahatlığıyla bir sonraki bölüme geçebiliriz.

BÖLÜM 2

# Pencere Araçları (Widgets) – 1. Bölüm

Arayüz uygulamaları tabii ki sadece pencerelerden ibaret değildir. Arayüzler üzerinde, istediğimiz işlemleri yapabilmemiz için yerleştirilmiş birtakım menüler, yazılar, düğmeler, kutucuklar ve buna benzer araçlar da bulunur. İşte herhangi bir programın arayüzü üzerinde bulunan düğmeler, etiketler, sağa-sola, yukarı-aşağı kayan çubuklar, kutular, kutucuklar, menüler, vb. hepsi birden pencere araçlarını, yani widget'leri, oluşturuyor.

Aslında şimdiye kadar bu pencere araçlarından bir tanesini gördük. Bildiğimiz bir pencere aracı olarak elimizde şimdilik "Label" bulunuyor. Gelin bu "Label" adlı pencere aracına biraz daha yakından bakalım:

## 2.1 "Label" Pencere Aracı

Daha önce de söylediğimiz gibi bu kelime İngilizce'de "etiket" anlamına geliyor. Bu araç, anlamına uygun olarak pencerelerin üzerine etiket misali öğeler yapıştırmamızı sağlıyor.

Hatırlayacağınız gibi bu aracı "Label()" şeklinde kullanıyorduk. İsterseniz bununla ilgili olarak hemen basit bir örnek verelim:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text = "Hata: Bellek Read Olamadı!")
etiket.pack()

mainloop()
```

Gördüğünüz gibi yukarıdaki kullanımda "Label" aracı bir metnin pencere üzerinde görüntülenmesini sağlıyor.

## 2.2 “Button” Pencere Aracı

Bu araç yardımıyla pencerelerimize, tıklandıklarında belli bir işlevi yerine getiren düğmeler ekleyebileceğiz. Hemen bir örnek verelim:

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

from Tkinter import *

pencere = Tk()
dugme = Button(text="TAMAM", command = pencere.quit)
dugme.pack()

mainloop()
```

Dikkat ederseniz, “Button()” aracının kullanımı daha önce gördüğümüz “Label” aracının kullanımına çok benziyor. Burada da parantez içinde bazı parametreler kullandık. “text” parametresini zaten biliyoruz: Kullanıcıya göstermek istediğimiz metni bu “text” parametresi yardımıyla belirliyoruz. Aynı parantez içinde gördüğümüz “command” parametresi ise düğme üzerine tıklandığında işletilecek komutu gösteriyor. Biz burada “pencere.quit” komutunu vererek, düğmeye tıklandığında pencerenin kapatılmasını istedik. Dolayısıyla bir satır içinde üç yeni özellik görmüş oluyoruz:

**Button** Kullanacağımız pencere aracı (İngilizce “button” kelimesi Türkçe’de “düğme” anlamına gelir).

**command** Oluşturduğumuz düğmeye tıklandığında çalıştırılacak komut

**xxx.quit** Düğmeye tıklandığında pencerenin kapatılmasını sağlayan komut.

Daha sonra gelen satırdaki ifade size tanıdık geliyor olmalı: “dugme.pack()”. Tıpkı daha önce “etiket.pack()” ifadesinde gördüğümüz gibi, bu ifade de hazırladığımız pencere aracının kargoya verilmek üzere “paketlenmesini” sağlıyor.

En son satırdaki “mainloop()” ifadesinin ne işe yaradığını artık söylemeye bile gerek yok...

Yukarıda bahsettiğimiz “command” parametresi çok güzel işler yapmanızı sağlayabilir. Mesela diyelim ki, “oluştur” düğmesine basınca bilgisayarda yeni bir dosya oluşturan bir arayüz tasarlamak istiyoruz. O halde hemen bu düşüncemizi tatbik sahasına koyalım:

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

from Tkinter import *

pencere = Tk()

def olustur():
    dosya = open("deneme.txt", "w")

dugme = Button(text = "oluştur", command=olustur)
dugme.pack()

mainloop()
```

Gördüğünüz gibi, Tkinter modülünü çağırıp “pencere = Tk()” şeklinde penceremizi oluşturduktan sonra bir fonksiyon yazdık. Tkinter dışından bir komut çalıştırmak istediğimizde bu şekilde bir fonksiyon tanımlamamız gerekir.

Daha sonra Button() pencere aracı yardımıyla, pencereye yerleştireceğimiz düğmeyi meydana getirdik. Burada "command" parametresine biraz önce oluşturduğumuz fonksiyonu atayarak düğmeye basıldığında yeni bir dosya oluşturulmasına zemin hazırladık. "text" parametresi yardımıyla da düğmemizin adını "oluştur" olarak belirledik. Ondan sonraki satırları ise zaten artık ezbere biliyoruz.

Eğer bu kodları yazarken, yukarıdaki gibi bir fonksiyon oluşturmadan;

```
dugme = Button(text = "oluştur", command=open("deneme.txt","w"))
```

gibi bir satır oluşturursanız, "deneme.txt" adlı dosya düğmeye henüz basmadan oluşacaktır...

İsterseniz bu arayüze bir de "çıkış" düğmesi ekleyebiliriz:

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

from Tkinter import *

pencere = Tk()

def olustur():
    dosya = open("deneme.txt", "w")

dugme = Button(text = "oluştur", command=olustur)
dugme.pack()

dugme2 = Button(text = "çıkış", command=pencere.quit)
dugme2.pack()

mainloop()
```

Burada yaptığımız şey, ikinci bir düğme oluşturmaktan ibaret... Oluşturduğumuz "dugme2" adlı değişken için de "dugme2.pack" ifadesini kullanmayı unutmuyoruz.

Düğmelerin pencere üstünde böyle alt alta görünmesini istemiyor olabilirsiniz. Herhalde yan yana duran düğmeler daha zarif görünecektir. Bunun için kodumuza şu eklemeleri yapmamız gerekiyor:

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

from Tkinter import *

pencere = Tk()

def olustur():
    dosya = open("deneme.txt", "w")

dugme = Button(text = "oluştur", command=olustur)
dugme.pack(side=LEFT)

dugme2 = Button(text = "çıkış", command=pencere.quit)
dugme2.pack(side=RIGHT)

mainloop()
```

"Paketleme" aşamasında düğmelerden birine "side=LEFT" parametresini ekleyerek o düğmeyi sola; öbürüne de "side=RIGHT" parametresini ekleyerek sağa yaslıyoruz.

İsterseniz gelin şu son yaptığımız örneği sınıflı yapı içinde göstererek sınıflarla ufak bir pratik daha yapmış olalım...

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.dugme = Button(text="oluştur", command=self.olustur)
        self.dugme.pack(side=LEFT)

        self.dugme2 = Button(text="çıkış", command=pencere.quit)
        self.dugme2.pack(side=RIGHT)

    def olustur(self):
        self.dosya = open("denene.txt","w")

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Hatırlarsanız önceki derslerimizden birinde, 1-100 arası sayılar içinden rastgele altı farklı sayı üreten bir uygulama yazmıştık. Artık “Button” adlı pencere aracını öğrendiğimize göre o uygulamayı birazcık daha geliştirebiliriz. Mesela o uygulamaya bir düğme ekleyebilir, o düğmeye basıldıkça yeni sayıların üretilmesini sağlayabiliriz:

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

from Tkinter import *
import random

pencere = Tk()
pencere.geometry("300x50+600+460")

def kodlar():
    liste = []
    for i in range(6):
        while len(liste) != 6:
            a = random.randint(1,100)
            if a not in liste:
                liste.append(a)
    etiket["text"] = liste

etiket = Label(text="Sayı üretmek için düğmeye basınız!",
               fg="white",
               bg="#61380B",
               font="Helvetica 12 bold")

etiket.pack()

dugme = Button(text="Yeniden",command=kodlar)
dugme.pack()

mainloop()
```

Burada, programda ana işlevi gören kodları “kodlar()” adlı bir fonksiyon içine aldık. Çünkü “But-

ton” pencere aracı içindeki “command” seçeneğine atayabilmemiz için elimizde bir fonksiyon olması gerekiyor. Düğmeye bastığımızda Tkinter, fonksiyon içindeki bu kodları işletecek. Düğmeye her basıldığında liste öğelerinin yenilenebilmesi için “liste = []” satırını fonksiyon içine alıyoruz. Böylece düğmeye ilk kez basıldığında önce liste öğesinin içi boşaltılacak, ardından altı adet sayı üretilip etikete yazdırılacaktır. Böylelikle düğmeye her basışımız yeni bir sayı dizisi oluşturacaktır. Eğer “liste = [ ]” satırını fonksiyon dışında tanımlarsak, düğmeye ilk basışımızda altı adet sayı üretilecektir, ama ikinci kez düğmeye bastığımızda, liste hâlâ eski öğeleri tutuyor olacağı için etiket yenilenmeyecek, düğmeye her basışımızda eski sayılar etikete yazdırılmaya devam edecektir. Bizim istediğimiz şey ise, düğmeye her basışta yeni bir sayı setinin etikete yazdırılması. Bunun için “liste = []” satırını fonksiyon içine almamız gerekiyor...

Programımızın işlevini yerine getirebilmesi için kullanıcının bir düğmeye basması gerekiyor. Bu yüzden kullanıcıya ne yapması gerektiğini söylesek iyi olur. Yani kullanıcımızı düğmeye basmaya yönlendirmemiz lazım. Bu amaçla etiket’in “text” seçeneğine “Sayı üretmek için düğmeye basınız!” şeklinde bir uyarı yerleştirdik. Programımız ilk kez çalıştırıldığında kullanıcı öncelikle bu uyarıyı görecek, böylelikle programı kullanabilmek için ne yapması gerektiğini bilecektir. Belki bu küçük programda çok belirgin olmayabilir, ama büyük programlarda, etiketler üzerine yazdığımız yönergelerin açık ve anlaşılır olması çok önemlidir. Aksi halde kullanıcı daha programı deneyemeden bir kenara atabilir... Tabii biz böyle olmasını istemeyiz, değil mi?

Şimdi de bu kodları sınıflı yapıya çevirelim:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
from Tkinter import *
import random

class Uygulama(object):
    def __init__(self):
        self.araclar()

    def kodlar(self):
        self.liste = []
        for i in range(6):
            while len(self.liste) != 6:
                a = random.randint(1, 100)
                if a not in self.liste:
                    self.liste.append(a)
        self.etiket["text"] = self.liste

    def araclar(self):
        self.etiket = Label(text="Sayı üretmek için düğmeye basınız!",
                            fg="white",
                            bg="#61380B",
                            font="Helvetica 12 bold")

        self.etiket.pack()

        self.dugme = Button(text="Yeniden", command = self.kodlar)
        self.dugme.pack()

pencere = Tk()
pencere.geometry("300x50+600+460")
uyg = Uygulama()
mainloop()
```

Bu kodlar da önceki gördüklerimizden çok farklı değil. Artık sınıflı bir Tkinter kodunun neye benzediğini gayet iyi biliyoruz...

## 2.3 “Entry” Pencere Aracı

Bu araç yardımıyla kullanıcının metin girebileceği tek satırlık bir alan oluşturacağız. Bu pencere aracının kullanımı da diğer araçların kullanımına benzer. Hemen bir örnek verelim:

```
from Tkinter import *

pencere = Tk()

giris = Entry()
giris.pack()

mainloop()
```

Her zamanki gibi ilk önce Tkinter modülünü çağırarak işe başladık. Hemen ardından da “pencere” adını verdiğimiz boş bir pencere oluşturduk. Dediğimiz gibi, “Entry” aracının kullanımı daha önce sözünü ettiğimiz pencere araçlarının kullanımına çok benzer. Dolayısıyla “giris” adını verdiğimiz “Entry()” aracını rahatlıkla oluşturabiliyoruz. Bundan sonra yapmamız gereken şey, tabii ki, bu pencere aracını paketlemek. Onu da “giris.pack()” satırı yardımıyla hallediyoruz. Son darbeyi ise “mainloop” komutuyla vuruyoruz.

Gördüğünüz gibi, kullanıcıya tek satırlık metin girme imkanı veren bir arayüz oluşturmuş olduk. İsterseniz şimdi bu pencereye birkaç düğme ekleyerek daha işlevli bir hale getirelim arayüzümüzü. Mesela pencere üzerinde programı kapatmaya yarayan bir düğme ve kullanıcının yazdığı metni silen bir düğme bulunsun:

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

from Tkinter import *

pencere = Tk()

def sil():
    giris.delete(0, END)

giris = Entry()
giris.pack()

dugme1 = Button(text = "KAPAT", command = pencere.quit)
dugme1.pack(side = LEFT)

dugme2 = Button(text = "SİL", command = sil)
dugme2.pack(side = RIGHT)

mainloop()
```

Bu kodlar içinde bize yabancı olan tek ifade “giris.delete(0, END)”. Bu komut, görünüşünden de anlaşılacağı gibi, “giris” değişkeninin içeriğini silmeye yarıyor. Parantez içindeki “0, END” ifadesi, metin kutusuna girilen kelimenin en başından en sonuna kadar bütün harflerin silinmesi emrini veriyor. Eğer “0, END” yerine, mesela “2, 4” gibi bir ifade koysaydık, girilen kelimenin 2. harfinden itibaren 4. harfine kadar olan kısmın silinmesi emrini vermiş olacaktık. Son bir alıştırma yaparak bu aracı da tamamlayalım:

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

from Tkinter import *

pencere = Tk()

def olustur():
    dosya = open("deneme.txt","w")
    metin = giris.get()
    dosya.write(metin)

giris = Entry()
giris.pack()

dugme = Button(text = "OLUŞTUR", command = olustur)
dugme.pack(side=LEFT)

dugme2 = Button(text = "ÇIK", command = pencere.quit)
dugme2.pack(side=RIGHT)

mainloop()
```

Burada da bize yabancı olan tek ifade “giris.get()”... Bu ifade “Entry” pencere aracı ile kullanıcıdan aldığımız metni elimizde tutup saklamamızı sağlıyor. Burada bu ifadeyi önce “metin” adlı bir değişkene atadık, sonra da bu metin değişkeninin içeriğini “dosya.write(metin)” komutunun yardımıyla boş bir dosyaya aktardık. Metin kutusunun içeriğini barındıran “deneme.txt” isimli dosya /home klasörünüzün altında veya masaüstünde oluşmuş olmalı...

Bir örnek daha yapalım...

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

from Tkinter import *
import random

pencere = Tk()

def bas():
    a = random.randint(1, 100)
    giris.delete(0, END)
    giris.insert(0, a)

giris = Entry(width=10)
giris.pack()

dugme = Button(text="bas", command=bas, width=2, height=0)
dugme.pack()

mainloop()
```

Gördüğünüz gibi, bu uygulama 1 ile 100 arasında rastgele sayılar seçiyor... Aslında yaptığımız işlem çok basit:

Öncelikle Python’un “random” adlı modülünü çağırdık. Daha önce de gördüğümüz gibi, rastgele sayılar seçerken bize bu modül yardımcı olacak. Ardından da bu rastgele sayıları oluşturup ekrana yazdırmamızı sağlayacak fonksiyonu oluşturuyoruz. Bu fonksiyonda öncelikle rastgele

sayıların hangi aralıkta olacağını belirleyip bunu “a” adlı bir değişkene atıyoruz. Böylece rastgele sayıları ekrana yazdırmak için gereken altyapıyı oluşturmuş olduk. Şimdi bu noktada eğer bir önlem almazsak ekrana basılacak sayılar yan yana sıralanacaktır. Yani mesela diyelim ki ilk rastgele sayımız 3 olsun. Bu 3 sayısı ekrana yazıldıktan sonra ikinci rastgele sayı ekrana gelmeden önce bu ilk sayının ekrandan silinmesi gerekiyor. Bütün sayılar yan yana ekrana dizilmemeli. Bunun için kodumuza şu satırı ekliyoruz:

```
giris.delete(0, END)
```

Bu satır sayesinde, ilk sayı ekrana basıldıktan sonra ekranın 0. konumundan sonuncu konumuna kadar bütün her şeyi siliyoruz. Böylelikle ikinci gelecek sayı için yer açmış oluyoruz. İsterseniz yukarıdaki kodları bu satır olmadan çalıştırmayı bir deneyebilirsiniz. Ondan sonra gelen satırın ne işe yaradığını anlamışsınızdır: “a” değişkenini 0. konuma yerleştiriyoruz. Fonksiyonumuzu böylece tamamlamış olduk. Daha sonra normal bir şekilde “Entry” ve Button” adlı pencere araçları yardımıyla düğmelerimizi ve metin alanımızı oluşturuyoruz. Burada bazı yeni “seçenekler” dikkatiniz çekmiş olmalı: Bunlar, “width” ve “height” adlı seçenekler... “width” seçeneği yardımıyla bir pencere aracının genişliğini; “height” seçeneği yardımıyla ise o aracın yüksekliğini belirliyoruz.

## 2.4 Frame()

Frame() Tkinter içinde bulunan sınıflardan biridir. Bu sınıf aynı zamanda bir pencere aracı (widget) olarak da kullanılabilir. Bir pencere aracı olarak kullanıldığında bu sınıfın görevi, pencere içindeki öbür pencere araçlarını pencere içinde rahatça konumlandırmayı sağlamaktır. Bu araçtan, ileride geometri yöneticilerini (geometry managers) işlerken daha ayrıntılı olarak bahsedeceğiz. Bu derste Frame() pencere aracının temel olarak nasıl kullanıldığına dair birkaç örnek vermekle yetineceğiz. Mesela şu örneğe bir bakalım:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
etiket.pack()

giris = Entry()
giris.pack()

dugme = Button(text="Gönder",command=pencere.destroy)
dugme.pack()

mainloop()
```

Burada gördüğünüz gibi, “gönder” düğmesi hemen üstündeki kutucuğa çok yakın duruyor. Bu arayüzün daha iyi görünmesi için bu iki pencere aracının arasını biraz açmak isteyebiliriz. İşte Frame() aracı böyle bir ihtiyaç halinde işimize yarayabilir:

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

from Tkinter import *
```

```
pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
etiket.pack()

giris = Entry()
giris.pack()

cerceve = Frame()
cerceve.pack()

dugme = Button(text="Gönder",command=pencere.destroy)
dugme.pack()

mainloop()
```

Gördüğünüz gibi, Frame() aracını “giris” ve “dugme” adlı pencere araçlarının arasına yerleştirdik. Bu kodları çalıştırdığımızda, düğmenin azıcık da olsa aşağıya kaydığını göreceksiniz. Eğer kayma oranını artırmak isterseniz, “pad” seçeneğinden yararlanabilirsiniz. Bu seçenek, bir pencere aracının, öteki pencere araçları ile arasını açmaya yarıyor. pad seçeneği iki şekilde kullanılabilir: “padx” ve “pady”. Bu iki seçenekten “padx”, x düzlemi üzerinde işlem yaparken, “pady” ise y düzlemi üzerinde işlem yapıyor. Yani:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
etiket.pack()

giris = Entry()
giris.pack()

cerceve = Frame()
cerceve.pack(pady=5)

dugme = Button(text="Gönder",command=pencere.destroy)
dugme.pack()

mainloop()
```

Burada sadece “pady” seçeneğini kullanmamız yeterli olacaktır. Çünkü bizim bu örnekte istediğimiz şey, iki pencere aracının arasını y düzlemi üzerinde açmak. Yani yukarıdan aşağıya doğru... Eğer x düzlemi üzerinde işlem yapmak isteseydik (yani soldan sağa doğru), o zaman padx seçeneğini de kullanacaktık. İsterseniz bu padx ve pady seçeneklerine bazı sayılar vererek birtakım denemeler yapabilirsiniz. Bu sayede bu seçeneklerin ne işe yaradığı daha net anlaşılacaktır. Ayrıca dikkat ederseniz, bu padx ve pady seçenekleri, daha önce gördüğümüz “side” seçeneğine kullanım olarak çok benzer. Hem padx ve pady, hem de side seçenekleri, pack aracının birer özelliğidir. Bu özelliklere ileride daha ayrıntılı olarak değineceğiz.

İsterseniz yukarıdaki kodların sınıflı yapı içinde nasıl görüneceğine de bir bakalım...

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

```
from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.guiPenAr()

    def guiPenAr(self):
        self.etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
        self.etiket.pack()

        self.giris = Entry()
        self.giris.pack()

        self.cerceve = Frame()
        self.cerceve.pack(pady=5)

        self.dugme = Button(text="Gönder",command=pencere.destroy)
        self.dugme.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Şimdilik bu Frame() pencere aracını burada bırakıyoruz. Bir sonraki bölümde bu araçtan ayrıntılı olarak bahsedeceğiz.

Böylelikle Pencere Araçları konusunun ilk bölümünü bitirmiş olduk. Şimdi başka bir pencere aracı grubuna geçmeden önce çok önemli bir konuya değineceğiz: Tkinter'de Geometri Yöneticileri (Geometry Managers).

BÖLÜM 3

# Geometri Yöneticileri

Şimdiye kadar yaptığımız örneklerden de gördüğünüz gibi, bir pencere aracını kullanıcılara gösterebilmemiz için bu aracı tanımlamak yeterli olmuyor. Aracı tanımlamanın yanısıra özel bir metodu da kullanmamız gerekiyor. Şu ana dek Tkinter ile hazırladığımız örneklerde pencere araçlarını kullanıcıya gösterebilmek için ".pack()" adlı özel bir metottan yararlandık. Bu metodun kullanımına şöyle bir örnek verebiliriz:

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

from Tkinter import *

pencere = Tk()
pencere.geometry("200x100+50+100")

dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")
dolar.pack()

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")
avro.pack()

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")
lira.pack()

mainloop()
```

Yukarıdaki pencere araçlarını gösterebilmek için kullandığımız ".pack()" adlı metoda Tkinter'de "geometri yöneticisi" (geometry manager) adı veriliyor. Anladığınız gibi, geometri yöneticilerinin temel işlevi Tkinter ile oluşturulan pencere araçlarını (yani "widget"leri) pencere üzerinde görünür hale getirmektir. Tkinter'de esasen üç farklı geometri yöneticisi bulunur. Bu geometri yöneticilerini şöyle sıralayabiliriz:

- pack()
- grid()
- place()

Geometri Yöneticileri bir pencere aracının konumunu belirler. Bu Geometri Yöneticilerinin her birinin ayrı özellikleri vardır ve bunların her biri belli bir işlevi yerine getirmede daha başarılıdır. Pack ve Grid Geometri Yöneticileri genel olarak hemen her işi yapmamıza izin verir. O yüzden bu iki Yönetici gayet kullanışlıdır. Place Geometri Yöneticisi ise pencere araçlarını pencere üzerinde yerli yerine oturtma konusunda çok ustadır. O yüzden pencere araçlarını istediğimiz şekilde pencere üzerine konumlandırmak istediğimiz zaman bu işi en kolay Place Geometri Yöneticisi yardımıyla yapabiliriz.

Geometri Yöneticileri ile ilgili olarak aklımızda tutmamız gereken en önemli şey aynı pencere üzerinde birbirinden farklı Yöneticileri bir arada kullanmamaktır... Bunun ne demek olduğunu ilerde örneklerle gösterdiğimiz zaman çok daha iyi anlayacaksınız.

Bu geometri yöneticileri içinde en kolay olanı pack() adlı yöneticidir. Zaten bu sebeple şimdiye kadar gördüğümüz örneklerde de hep bu pack() metodunu kullandık. Ancak hangi geometri yöneticisini kullanırsak kullanalım, öteki yöneticilerin nasıl kullanıldığını da öğrenmemiz gerekir. Çünkü bu geometri yöneticilerinin her birinin iyi olduğu bir alan vardır. Şimdi isterseniz bu geometri yöneticilerini tek tek inceleyelim...

## 3.1 pack() Geometri Yöneticisi

Bir önceki bölümde de söylediğimiz gibi, pack() geometri yöneticisi Tkinter'deki geometri yöneticileri içinde en kolay olanıdır. Bu geometri yöneticisinin kullanımıyla ilgili basit bir örneği önceki bölümde vermiştik. İsterseniz bu örneği tekrar önümüze alalım:

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

from Tkinter import *

pencere = Tk()
pencere.geometry("200x100+50+100")

dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")
dolar.pack()

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")
avro.pack()

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")
lira.pack()

mainloop()
```

pack() metodunu parametresiz olarak kullandığımızda öğelerin alt alta sıralandığına dikkat edin. Ancak gördüğünüz gibi, pack() metodunu bu şekilde kullandığımızda etiketler pencereye orantısız olarak dağılıyor. Yani görünüş biraz özensiz duruyor. Ama eğer biz istersek bu durumu bir nebze olsun düzeltebiliriz. Bunun için ihtiyacımız olan şey, pack() geometri yöneticisinin "expand" adlı seçeneğidir...

**expand seçeneği**

expand seçeneği, pencere araçlarının pencere üzerinde daha orantılı bir şekilde dağılmasını sağlar. Bu expand seçeneği "YES" ve "NO" olmak üzere iki farklı değer alır. Varsayılan değer "NO"dur... Bunu yukarıdaki örnek üzerinde görelim:

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

from Tkinter import *

pencere = Tk()
pencere.geometry("200x100+50+100")

dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")

dolar.pack(expand=YES)

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")

avro.pack(expand=YES)

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")

lira.pack(expand=YES)

mainloop()
```

"expand" kelimesi İngilizce'de "genişlet, yay" gibi anlamlara gelir. Yukarıdaki örnekte gördüğünüz gibi, pack() metoduna parametre olarak "expand = YES" seçeneğini vermek yoluyla öğelerin pencere üzerine "dağılmasını" veya "yayılmasını" sağlıyoruz. Eğer "expand = NO" gibi bir biçim kullanırsak, öğeler pencere üzerine yayılmayacak, yani sanki bu seçeneği hiç kullanmamışız gibi davranılacaktır...

Gördüğünüz gibi, öğeler yukarıdaki şekilde birbirinden ayrı duruyor. Biz bu öğelerin birbirine değmesini de isteyebiliriz. Yani tercihimiz, tıpkı bazı ülkelerin bayraklarında olduğu gibi, bu üç rengin şeritler halinde birbirine değecek şekilde uzanmasını sağlamak olabilir... Bu iş için kullanabileceğimiz ayrı bir seçenek var. Bu seçeneğin adı: fill.

**fill seçeneği**

Bu "fill" kelimesi İngilizce'de "doldurmak" anlamına gelir. Bu seçenek yardımıyla, araçlarımızın pencereyi doldurmasını sağlayacağız. "fill" seçeneği üç farklı değer alabilir: "X", "Y" ve "BOTH".

- “X” seçeneği, pencere aracımızın, içinde bulunduğu pencereyi x düzlemi boyunca doldurmasını sağlar (yani soldan sağa doğru)
- “Y” seçeneği, pencere aracımızın, içinde bulunduğu pencereyi y düzlemi boyunca doldurmasını sağlar (yani yukarıdan aşağıya doğru)
- “BOTH” seçeneği, pencere aracımızın, içinde bulunduğu pencereyi x ve y düzlemi boyunca doldurmasını sağlar (“both” kelimesi İngilizce’de “her ikisi de” anlamına gelir)

Şimdi bu seçenekleri örnekler üzerinde görerek kafamızda somutlaştıralım:

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

from Tkinter import *

pencere = Tk()
pencere.geometry("200x100+50+100")

dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")

dolar.pack(expand=YES, fill = X)

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")

avro.pack(expand=YES, fill = Y)

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")

lira.pack(expand=YES, fill = BOTH)

mainloop()
```

Burada her bir öğede ayrı bir değer kullandık. Ama eğer siz isterseniz her üçü için tek bir değer kullanarak, bu değerlerin tam olarak ne yaptığını daha net görmeyi tercih edebilirsiniz...

**side seçeneği**

side seçeneğini daha önce de kullanmıştık. Bu seçenek bir pencere aracını istediğimiz bir yöne yaslamamızı sağlıyor... Bu seçenek dört farklı değer alır: TOP, BOTTOM, LEFT ve RIGHT

- TOP: pencere aracını üste yaslar.
- BOTTOM: pencere aracını dibe yaslar.
- LEFT: pencere aracını sola yaslar.
- RIGHT: pencere aracını sağa yaslar.

Ancak bu dört değer içinde en sık kullanılanlar TOP ve LEFT’tir. Eğer pencere araçlarını yan yana sıralamak istersek kısaca “LEFT” değerini kullanabiliriz. Eğer pencere araçlarını üst üste sıralamak istersek de “TOP” değerini kullanabiliriz. Şu iki örneği dikkatlice inceleyin:

LEFT ile Kullanım:

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

from Tkinter import *

pencere = Tk()
dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")

dolar.pack(side=LEFT)

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")

avro.pack(side=LEFT)

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")

lira.pack(side=LEFT)

mainloop()
```

TOP ile Kullanım:

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

from Tkinter import *

pencere = Tk()
dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")

dolar.pack(side=TOP)

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")

avro.pack(side=TOP)

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")

lira.pack(side=TOP)
```

```
mainloop()
```

Ayrıca istersek birkaç seçeneği bir arada kullanarak aşağıdaki şekilde bir şey de yapabiliriz:

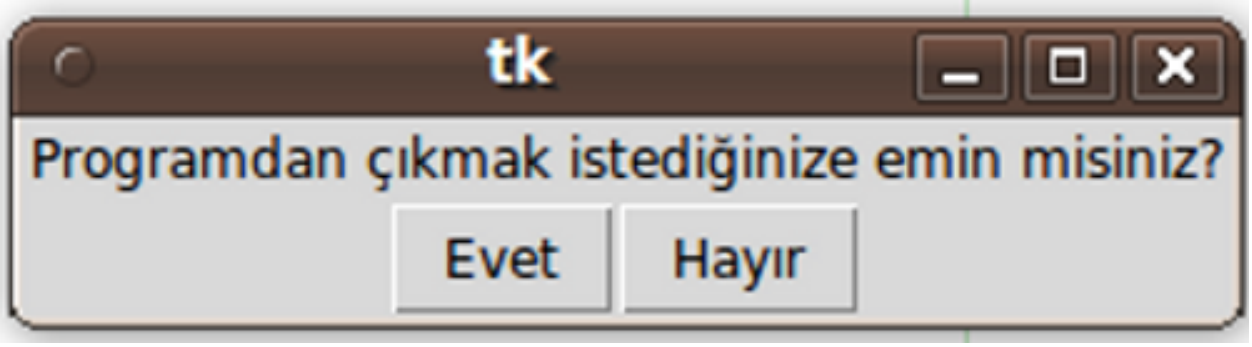
```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from Tkinter import *

pencere = Tk()
dolar = Label(text="Dolar",
              fg = "white",
              bg = "red",
              font="Verdana 13 bold")

dolar.pack(side=TOP, expand=YES,fill=BOTH)

avro = Label(text="Avro",
             fg = "white",
             bg = "blue",
             font="Verdana 13 bold")

avro.pack(side=TOP,expand=YES, fill=BOTH)

lira = Label(text="Lira",
             fg = "white",
             bg = "green",
             font="Verdana 13 bold")

lira.pack(side=TOP,expand=YES, fill=BOTH)

mainloop()
```

Bu pencereyi elle sağa-sola veya yukarı-aşağı doğru boyutlandırın. Gördüğünüz gibi, pencereyi ne şekilde boyutlandırırsanız boyutlandırın, etiket öğeleri pencereyi doldurmaya devam edecektir...

Elbette pack() geometri yöneticisi yalnızca etiketlerle birlikte kullanılmıyor. Öğrendiğimiz öteki pencere araçlarını da kullanarak, örneğin, şöyle bir şey yapabiliriz:

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

from Tkinter import *

pencere = Tk()


uyari = Label(pencere,text="Programdan çıkmak istediğinize emin misiniz?")
uyari.pack()

evet = Button(text="Evet",command=pencere.quit)
evet.pack()

hayir = Button(text="Hayır")
hayir.pack()

mainloop()
```

Burada pack() metotları içinde değişik seçeneklere farklı değerler uygulayarak istediğinize yakın bir görüntü elde edebilirsiniz. Ancak mesela aşağıdaki gibi bir görüntü elde etmek için biraz uğraşmak gerekiyor:




Yukarıdaki gibi bir görüntü elde edebilmek için daha önce gördüğümüz Frame() adlı pencere aracından yararlanmanız gerekir. Frame() aracının nasıl kullanıldığını hatırlıyorsunuz:

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

from Tkinter import *

pencere = Tk()

yeni = Frame()
yeni.pack(side=BOTTOM)

uyari = Label(pencere,text="Programdan çıkmak istediğinize emin misiniz?")
uyari.pack()

evet = Button(yeni,text="Evet",command=pencere.quit)
evet.pack(side=LEFT)

hayir = Button(yeni,text="Hayır")
hayir.pack()

mainloop()
```

Burada dikkatimizi çekmesi gereken bazı çok önemli yenilikler var. Mesela yukarıda geçen şu satıra bakın:

```
uyari = Label(pencere,text="Programdan çıkmak istediğinize emin misiniz?")
```

Kalın harflerle gösterdiğimiz ifade bizim için bir yenilik sayılır. Daha önceden yazdığımız pencere araçlarında, bu “pencere” ifadesini kullanmamıza gerek yoktu. Çünkü yazdığımız programlarda hep sadece tek bir “taşıyıcı pencere aracı” (container widget) yer alıyordu. Bu “taşıyıcı pencere aracı” ifadesi de ne oluyor, diye sorarsanız hemen açıklayalım: Görevi başka pencere araçlarını taşımak olan pencere araçlarına Tkinter’de “taşıyıcı pencere aracı” (container widget) adı veriliyor. Mesela yazdığımız bütün programlarda “pencere = Tk” ifadesi ile oluşturduğumuz ana penceremiz esasında bir taşıyıcı pencere aracıdır. Yani bu pencerenin görevi pencere araçlarını tutmaktır. Aynı şekilde “Frame()” de bir taşıyıcı pencere aracıdır. Bu pencere aracı da, tıpkı ana penceremiz gibi, başka pencere araçlarını üzerinde taşıyabilir. Mesela yukarıdaki örnekte Frame() pencere aracı bu görevde kullanılıyor. Nasıl “uyarı” adlı etiketi “pencere” adlı taşıyıcının üstüne yerleştirmişsek, “evet” ve “hayır” adlı pencere araçlarını da “yeni” adlı taşıyıcının üzerine yerleştiriyoruz. Bu yerleştirme işlemini, yukarıda gördüğünüz gibi, pencere araçlarının ilk argümanı olarak, herhangi bir taşıyıcı adı belirtmek

suretiyle gerçekleştiriyoruz. Mesela “evet” adlı pencere aracını “yeni” adlı taşıyıcı üzerine yerleştirmişiz. Bunu “Button(yeni, text=”Evet”.....”) komutundaki kalın harflerle gösterilen ifadeden anlıyoruz... Aynı şekilde “uyarı” adlı pencere aracını “pencere” adlı taşıyıcının üzerine yerleştirdiğimizi de “Label(pencere,text=”Pr...”) komutundaki kalın harflerle gösterilen ifadeden anlıyoruz...

Peki Frame() aracını (ve pack yöneticisini) kullanarak başka neler yapabiliriz? Hemen bir örnek daha verelim:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
etiket.pack()

giris = Entry()
giris.pack()

cerceve = Frame(height=2, bd=1, relief=SUNKEN)
cerceve.pack(fill=X, pady=5, padx=5)

dugme = Button(text="Gönder", command=pencere.destroy)
dugme.pack()

mainloop()
```

Burada birkaç yeni özellik görüyoruz. Daha önce görmediğimiz özellikler, “bd” ve “relief” seçenekleri... “bd” İngilizce’de “kenar, kenarlık, sınır” anlamına gelen “border” kelimesinin kısaltması... Bu seçenek yardımıyla Frame() aracının kenarlık ölçülerini belirliyoruz. Varsayılan olarak Frame() aracının kenarlık boyutu 0 (sıfır)’dır. Biz bunu yukarıda 1 olarak değiştirdik. Aynı satırda gördüğümüz “height” seçeneğini daha önce pencere boyutu ve entry konularını işlerken de görmüştük. Oradan hatırladığımıza göre, bu seçenek bir pencere aracının “yüksekliğini” ayarlamamızı sağlıyordu... Yukarıdaki örnekte de Frame() pencere aracının yüksekliğini “2” olarak ayarladık. “relief” seçeneği ise İngilizce bir kelime olarak Türkçe’de “kabartma (rölyef)” anlamına geliyor. Dolayısıyla bu seçenek yardımıyla pencere aracının “kabartma” tipini belirliyoruz. Burada altı farklı stil uygulayabiliriz:

- SOLID
- FLAT
- RIDGE
- SUNKEN
- GROOVE
- RAISED

Bu stillerin nasıl bir görüntü ortaya çıkardığını daha net görebilmek için “height” ve “bd” seçeneklerini daha yüksek değerlere ayarlamayı tercih edebilirsiniz.

Yukarıdaki örneğin sınıflı yapı içinde nasıl görüneceğine bakalım bir de...

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

```
from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.guiPenAr()

    def guiPenAr(self):
        self.etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
        self.etiket.pack()

        self.giris = Entry()
        self.giris.pack()

        self.cerceve = Frame()
        self.cerceve.pack(side=BOTTOM,
                          padx=5,
                          pady=5)

        self.dugme = Button(self.cerceve,
                            text="Gönder",
                            width=5,
                            command=pencere.destroy)

        self.dugme.pack(side=LEFT,padx=5)

        self.dugme2 = Button(self.cerceve,
                             text="İptal",
                             width=5)

        self.dugme2.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

Frame() aracını kullanarak mesela şöyle bir şey de yapabiliriz:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
etiket.pack()

giris = Entry()
giris.pack()

cerceve = Frame()
cerceve.pack(side=BOTTOM,
             padx=5,
             pady=5)

dugme = Button(cerceve,
               text="Gönder",
               width=5,
```

```
                command=pencere.destroy)

dugme.pack(side=LEFT,padx=5)

dugme2 = Button(cerceve,
                text="İptal",
                width=5)

dugme2.pack()

mainloop()
```

Burada pencere aracı seçeneklerini nasıl kullandığımıza dikkat edin.

İsterseniz bu örneği de sınıflı olarak ifade edelim:

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

from Tkinter import *

class Uygulama(object):
    def __init__(self):
        self.guiPenAr()

    def guiPenAr(self):
        self.etiket = Label(text="Aşağıdaki kutucuğa e.posta adresinizi yazınız!")
        self.etiket.pack()

        self.giris = Entry()
        self.giris.pack()

        self.cerceve = Frame()
        self.cerceve.pack(side=BOTTOM,
                          padx=5,
                          pady=5)

        self.dugme = Button(self.cerceve,
                            text="Gönder",
                            width=5,
                            command=pencere.destroy)

        self.dugme.pack(side=LEFT,padx=5)

        self.dugme2 = Button(self.cerceve,
                             text="İptal",
                             width=5)

        self.dugme2.pack()

pencere = Tk()
uyg = Uygulama()
mainloop()
```

pack() metodunu kullanarak çok karmaşık arayüz görüntüleri elde edebiliriz. Aslında bu metodu kullanarak yapamayacağımız şey yok gibidir. Ama bazı şeyleri bu metotla yapmaya kalkışmak bir süre sonra işkenceye dönüşebilir. O yüzden yeri geldiğinde pack() yerine başka metotları kullanmayı da bilmeliyiz... Mesela bazı işler için grid() metodu daha uygun olabilir...

grid() metodunu anlatmaya geçmeden önce isterseniz pack() metodunu nerelerde kullanabileceğimizi şöyle bir sıralayalım:

- Pencere araçlarını alt alta dizmek istediğimizde,
- Pencere araçlarını yan yana dizmek istediğimizde,
- Pencere araçlarını alt alta veya yan yana dizip, bunların pencereyi (x veya y düzleminde ya da her ikisinde birden) tamamen kaplamasını istediğimizde.

Bu metodu nerelerde kullanmamızın uygun olacağını gördüğümüze göre artık başka bir geometri yöneticisini anlatmaya başlayabiliriz.

## 3.2 grid() Geometri Yöneticisi

Bu geometri yöneticisi, pencere araçlarının "ızgara" biçiminde sıralanacağı arayüzler için uygundur. Bununla hemen basit bir örnek yapalım:

```
# -*- coding: utf-8 -*-
from Tkinter import *

dugme1 = Button(text="Ürünler")
dugme1.grid()

dugme2 = Button(text="Hizmetler")
dugme2.grid()

dugme3 = Button(text="Ulaşım")
dugme3.grid()

dugme4 = Button(text="Hakkında")
dugme4.grid()

mainloop()
```

grid() adlı geometri yöneticisi basitçe bu şekilde kullanılıyor. Ama tabii ki bu haliyle pek cazip görünmüyor. Buna istediğimiz şekli verebilmek için yine birtakım seçeneklerden yararlanmamız gerekecek:

**row seçeneği**

Bu geometri yöneticisinin en önemli iki seçeneğinden birincisi bu "row" seçeneğidir. Bu kelime İngilizce'de "satır" anlamına gelir. Bu seçenek yardımıyla, pencere aracının hangi satırda yer alacağını belirleyeceğiz. Bununla herhangi bir örnek yapmadan önce, isterseniz grid() geometri yöneticisinin öteki seçeneğinden de bahsedelim. Çünkü bu seçenek hep "row" seçeneğiyle birlikte kullanılır.

**column seçeneği**

"column" kelimesi İngilizce'de "sütun" anlamına gelir. Dolayısıyla bu seçenek yardımıyla pencere aracımızın pencere üzerinde hangi sütunda yer alacağını belirleyeceğiz. Artık bu iki seçeneği kullanarak bir örnek verebiliriz. Aşağıdaki örneği dikkatle inceleyin:

```
# -*- coding: utf-8 -*-
from Tkinter import *

dolar = Label(text="Dolar")
dolar.grid(row=0,column=0)
```

```
avro = Label(text="Avro")
avro.grid(row=1,column=0)

lira = Label(text="Lira")
lira.grid(row=2,column=0)

dolar_g = Entry()
dolar_g.grid(row=0,column=1)

avro_g = Entry()
avro_g.grid(row=1,column=1)

lira_g = Entry()
lira_g.grid(row=2,column=1)
```

Gördüğünüz gibi bu grid() metodu yardımıyla pack() metoduna kıyasla daha “ince” ayarlamalar yapabiliyoruz. “grid()” metodu bize, pencere aracımızın koordinatlarını daha hassas bir şekilde belirtme imkanı sunuyor. Şimdi isterseniz ilk verdiğimiz, “cazip görünmeyen” örneği yeniden ele alalım. Örneğimiz şuydu:

```
# -*- coding: utf-8 -*-
from Tkinter import *

dugme1 = Button(text="Ürünler")
dugme1.grid()

dugme2 = Button(text="Hizmetler")
dugme2.grid()

dugme3 = Button(text="Ulaşım")
dugme3.grid()

dugme4 = Button(text="Hakkında")
dugme4.grid()

mainloop()
```

Şimdi yapacağımız şey, pencere üzerinde dağınık bir şekilde duran bu araçların her biri için uygun bir satır-sütun belirlemek. Hemen bakalım:

```
# -*- coding: utf-8 -*-
from Tkinter import *

dugme1 = Button(text="Ürünler")
dugme1.grid(row=0, column=0)

dugme2 = Button(text="Hizmetler")
dugme2.grid(row=0, column=1)

dugme3 = Button(text="Ulaşım")
dugme3.grid(row=1, column=0)

dugme4 = Button(text="Hakkında")
dugme4.grid(row=1,column=1)

mainloop()
```

İsterseniz daha önceden gördüğümüz “fg”, “bg” ve “font” gibi seçenekler yardımıyla bu

düğmelerin ve üzerindeki yazıların renklerini gönlünüze göre değiştirebilirsiniz de...

Dediğimiz gibi, grid() yöneticisinin en uygun kullanım alanı “ızgara” biçimli arayüzlerdir. Mesela hesap makineleri... Şu örneğe bir bakalım:

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

from Tkinter import *

pencere = Tk()

#bir hesap makinesinde pencerenin boyutlandırılabilir olması anlamsız olacaktır.
#O yüzden pencereleri boyutlandırılamaz hale getiriyoruz.
pencere.resizable(width=FALSE,height=FALSE)

#hesap makinemizin tuş takımında yer alacak simgeleri ve sayıları oluşturuyoruz.
liste = [\
    "9", "8", "7",
    "6", "5", "4",
    "3", "2", "1",
    "0", "+", "-",
    "/", "*", "=",
    "C"]

#grid() yöneticisinin row (sıra) ve column (sütun) seçenekleri için birer başlangıç
#değeri belirliyoruz.
sira = 1
sutun = 0

#tuş takımındaki bütün pencere araçlarını tek bir for döngüsü ile oluşturuyoruz.
for i in liste:
    Button(text=i,width=4,relief=GROOVE).grid(row=sira,column=sutun)
    sutun += 1 #column seçeneğinin değeri her defasında bir sayı artıyor.

    if sutun > 2: #sutun'un yani "column'un" değeri 2'yi geçtiğinde,
        sutun = 0 #column'un değerini sıfırlıyoruz. Böylece tuşlar
                  #alt alta dizilebiliyor.
        sira += 1 #row'un (sıra) değeri de her defasında bir sayı artıyor.

mainloop()
```

Gördüğünüz gibi, yukarıda bir hesap makinesinde olması gereken temel tuşları, basit bir for döngüsü yardımıyla bir çırpıda oluşturduk. Yukarıda kodlar içinde verdiğim yorumları dikkatlice incelemenizi öneririm. Aşağıda hesap makinemizin bitmiş halini bulabilirsiniz:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
from __future__ import division

from Tkinter import *

depo = ""
pencere = Tk()
pencere.resizable(width=FALSE,height=FALSE)

ekran = Entry(width=20)
ekran.grid(row=0, column=0, columnspan=3,ipady=4)
```

```
def hesapla(tus):
    global depo
    if tus in "0123456789":
        ekran.insert(END,tus)
        depo = depo + tus


    if tus in "+-/*":
        depo = depo + tus
        ekran.delete(0,END)

    if tus == "=":
        ekran.delete(0,END)
        #Normalde tehlikeli bir fonksiyon olan eval()'i nasıl
        #kullandığımıza dikkat edin...
        hesap = eval(depo,{"__builtins__":None},{})
        depo = str(hesap)
        ekran.insert(END,depo)

    if tus == "C":
        ekran.delete(0,END)
        depo = ""

liste = [\
    "9", "8", "7",
    "6", "5", "4",
    "3", "2", "1",
    "0", "+", "-",
    "/", "*", "=",
    "C"]

sira = 1
sutun = 0
for i in liste:
    komut = lambda x=i: hesapla(x)
    Button(text=i,
           width=5,
           relief=GROOVE,
           command=komut).grid(row=sira,
                               column=sutun)
    sutun += 1

    if sutun > 2:
        sutun = 0
        sira += 1

mainloop()
```

Bu hesap makinesinde aklınıza takılan bir konu olursa bana nasıl ulaşacağınızı biliyorsunuz... Bu arada isterseniz hesap makinemizi bir de sınıflı yapı içinde görelim:

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

from __future__ import division

from Tkinter import *
```

```
class Uygulama(object):
    def __init__(self):
        self.guiPenAr()
        self.gV()

    def gV(self):
        self.depo = ""

    def guiPenAr(self):
        self.ekran = Entry(width=20)
        self.ekran.grid(row=0, column=0, columnspan=3,ipady=4)

        self.liste = [\
                    "9", "8", "7",
                    "6", "5", "4",
                    "3", "2", "1",
                    "0", "+", "-",
                    "/", "*", "=",
                    "C"]

        self.sira = 1
        self.sutun = 0

        for i in self.liste:
            self.komut = lambda x=i: self.hesapla(x)
            Button(text=i,
                   width=5,
                   relief=GROOVE,
                   command=self.komut).grid(row=self.sira,
                                            column=self.sutun)
            self.sutun += 1

            if self.sutun > 2:
                self.sutun = 0
                self.sira += 1

    def hesapla(self, tus):
        self.tus = tus
        if self.tus in "0123456789":
            self.ekran.insert(END,self.tus)
            self.depo = self.depo + self.tus

        if self.tus in "+-/*":
            self.depo = self.depo + self.tus
            self.ekran.delete(0,END)

        if self.tus == "=":
            self.ekran.delete(0,END)
            self.hesap = eval(self.depo,{"__builtins__":None},{})
            self.depo = str(self.hesap)
            self.ekran.insert(END,self.depo)

        if self.tus == "C":
            self.ekran.delete(0,END)
            self.depo = ""

pencere = Tk()
pencere.resizable(width=FALSE,height=FALSE)
```

```
uyg = Uygulama()

mainloop()
```

Dikkat ederseniz, bir önceki bölümde pack() yöneticisini incelerken öğrendiğimiz “GROOVE”, “SUNKEN”, vb. stilleri grid() geometri yöneticisiyle birlikte de kullanabiliyoruz.

Böylelikle bir geometri yöneticisini daha incelemiş olduk. Sıra geldi sonuncu geometri yöneticimize...

## 3.3 place() Geometri Yöneticisi

Tahminimce bu geometri yöneticisi Tkinter’deki üç yönetici içinden en çok beğeneceğiniz yönetici olmaya adaydır... Bütün geometri yöneticileri içinde, pencere araçlarını en hassas şekilde yerleştirmemizi sağlayan yönetici place() adlı geometri yöneticisidir...

place() geometri yöneticisi ile çalışırken bize temel olarak iki seçenek yardımcı olacak: “relx” ve “rely”. Tahmin edebileceğiniz gibi, “relx” x düzlemi üzerindeki işlemler için, “rely” ise “y” düzlemi üzerindeki işlemler için kullanılacak. Hemen bir örnek görelim:

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

from Tkinter import *

pencere = Tk()

giris1 = Entry()
giris1.place(relx=0.0, rely=0.0,relheight=0.15)

dugme1 = Button(text="Düğme1")
dugme1.place(relx=0.7, rely=0.0,relheight=0.16)

dugme2 = Button(text="Düğme2")
dugme2.place(relx=0.0,rely=0.2,relwidth=1)

giris2 = Entry()
giris2.place(relx=0.0,rely=0.35,relheight=0.5,relwidth=1)

mainloop()
```

Bu örnekte “relx” ve “rely” seçeneklerinin yanısıra, “relheight” ve “relwidth” seçeneklerini de kullandık. Rahatlıkla anlayabileceğiniz gibi, bu place() yöneticisi yardımıyla pencerelerimizi istediğimiz şekilde düzenleyebiliyoruz. Ama tabii ki, gördüğünüz gibi, koordinatları tam olarak vermeyi gerektirdiği için bu yöneticiyle çalışmak biraz emek istiyor. Bu yüzden bazı işler için bu yöneticiyi kullanmak pek pratik olmayabilir. Böyle durumlarda öteki yöneticilerin neler yapabileceğini de göz önünde bulundurmakta fayda var... Bir sonraki konumuz olan “checkbutton pencere aracı”nda place() adlı geometri yöneticisinden bahsetmeye devam edeceğiz.

# Pencere Araçları (Widgets) – 2. Bölüm

## 4.1 “Checkbutton” Pencere Aracı

“Checkbutton” denen şey, bildiğimiz “onay kutusu”... Basitçe şu şekilde oluşturuyoruz bu onay kutusunu:

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

from Tkinter import *

pencere = Tk()

onay = Checkbutton()
onay.pack()

mainloop()
```

Tabii ki muhtemelen bu onay kutumuzun, hatta kutularımızın birer adı olsun isteriz:

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

from Tkinter import *

pencere = Tk()

onay_el = Checkbutton(text="elma")
onay_el.pack()

onay_sa = Checkbutton(text="salatalık")
onay_sa.pack()

onay_do = Checkbutton(text="domates")
onay_do.pack()

onay_ka = Checkbutton(text="karnıbahar")
```

```
onay_ka.pack()

mainloop()
```

Gayet güzel.. Ama gördüğünüz gibi öğeler alt alta sıralanırken hizalı değiller. Dolayısıyla göze pek hoş görünmüyorlar. Eğer istersek şöyle bir görünüm verebiliriz onay kutularımıza:

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

from Tkinter import *

pencere = Tk()

onay_el = Checkbutton(text="elma")
onay_el.pack(side=LEFT)

onay_sa = Checkbutton(text="salatalık")
onay_sa.pack(side=LEFT)

onay_do = Checkbutton(text="domates")
onay_do.pack(side=LEFT)

onay_ka = Checkbutton(text="karnıbahar")
onay_ka.pack(side=LEFT)

mainloop()
```

Öğeler böyle yan yana dizildiklerinde fena görünmüyorlar, ama biz yine de öğelerin düzgün şekilde alt alta dizilmesini isteyebiliriz. Burada geometri yöneticilerinden biri olan place() adlı olanı tercih edebiliriz... Şimdi bununla ilgili bir örnek görelim.. Mesela bir tane onay kutusu oluşturup bunu en sola yaslayalım:

```
from Tkinter import *

pencere = Tk()

onay = Checkbutton(text="Kubuntu")
onay.place(relx = 0.0, rely = 0.1)

mainloop()
```

Bir önceki bölümde öğrendiğimizi place() yöneticisini nasıl kullandığımızı görüyorsunuz. Bu yöneticinin “relx” ve “rely” seçenekleri yardımıyla onay kutusunun koordinatlarını belirliyoruz. Buna göre onay kutumuz x düzlemi üzerinde 0.0 konumunda; y düzlemi üzerinde ise 0.1 konumunda yer alıyor. Yani yukarıdan aşağıya 0.0; soldan sağa 0.1 konumunda bulunuyor pencere aracımız.

Hemen Birkaç tane daha onay kutusu ekleyelim:

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

from Tkinter import *

pencere = Tk()

onay_pa = Checkbutton(text="Kubuntu")
```

```
onay_pa.place(relx = 0.0, rely = 0.1)

onay_de = Checkbutton(text="Debian")
onay_de.place(relx = 0.0, rely = 0.2)

onay_ub = Checkbutton(text="Ubuntu")
onay_ub.place(relx = 0.0, rely = 0.3)

onay_wix = Checkbutton(text="Windows XP")
onay_wix.place(relx = 0.0, rely = 0.4)

mainloop()
```

Dikkat ederseniz yukarıdaki bütün onay kutularının relx seçeneği 0.0 iken, rely seçenekleri birer birer artmış... Bunun nedeni, bütün onay kutularını sola yaslayıp hepsini alt alta dizmek isteyişimiz... Bu örnek relx ve rely seçeneklerinin nasıl kullanılacağı konusunda iyi bir fikir vermiş olmalı. Eğer bir kolaylık sağlayacaksa, relx'i sütun, rely'yi ise satır olarak düşünebilirsiniz.

Şimdi de mesela "Kubuntu"yu "Debian"ın yanına, "Ubuntu"yu da "Windows XP"nin yanına gelecek şekilde alt alta sıralayalım:

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

from Tkinter import *

pencere = Tk()

onay_pa = Checkbutton(text="Kubuntu")
onay_pa.place(relx = 0.0, rely = 0.1)

onay_de = Checkbutton(text="Debian")
onay_de.place(relx = 0.4, rely = 0.1)

onay_ub = Checkbutton(text="Ubuntu")
onay_ub.place(relx = 0.0, rely = 0.2)

onay_wix = Checkbutton(text="Windows XP")
onay_wix.place(relx = 0.4, rely = 0.2)

mainloop()
```

Kubuntu'yu 0.0 no'lu sütunun 0.1 no'lu satırına; Debian'ı da 0.4 no'lu sütunun 0.1 no'lu satırına yerleştirdik. Aynı şekilde Ubuntu'yu 0.0 no'lu sütunun 0.2 no'lu satırına; Windows XP'yi de 0.4 no'lu sütunun 0.2 no'lu satırına yerleştirdik.

Eğer oluşturduğumuz bu onay kutularına biraz canlılık katmak istersek, yani mesela kutunun seçili olup olmamasına göre bazı işler yapmak istersek kullanmamız gereken bazı özel kodlar var... Önce hemen yazmamız gereken ilk satırları yazalım:

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

from Tkinter import *

pencere = Tk()
```

Bunun hemen ardından şöyle iki satır ekleyelim:

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

from Tkinter import *

pencere = Tk()

d = IntVar()
d.set(0)
```

Burada “d=IntVar()” satırıyla yaptığımız şey “d” adlı bir değişken oluşturup, bunun değeri olarak “IntVar()” ifadesini belirlemek... Peki bu “IntVar()” denen şey de ne oluyor?

IntVar(), İngilizce “Integer Variable” (Sayı Değişkeni) ifadesinin kısaltması. Bu ifade yardımıyla değişken olarak bir sayı belirlemiş oluyoruz. Bunun dışında Tkinter’de kullanacağımız, buna benzer iki ifade daha var: StringVar() ve DoubleVar()

StringVar() yardımıyla karakter dizilerini; DoubleVar() yardımıyla da Ondalık Sayıları (kayan noktalı sayılar – floats) depolayabiliyoruz. Buraya kadar anlattıklarımız biraz bulanık gelmiş olabilir... Ama endişeye hiç gerek yok. d=IntVar() satırının hemen altındaki d.set(0) ifadesi pek çok şeyi açıklığa kavuşturacak. O halde hemen bu satırın anlamını kavramaya çalışalım..

Aslında d = IntVar() ifadesi yardımıyla başladığımız işi tamamlamamızı sağlayan satır bu d.set(0) satırı... Bunun yardımıyla “d=IntVar()” ifadesinin değeri olarak “0”ı seçtik. Yani bu satırı bir onay kutusu için yazdığımızı düşünürsek, onay kutusunun değerini “seçili değil” olarak belirlemiş olduk. Bu iki satırdan sonra oluşturacağımız onay kutusu karşımıza ilk çıktığında işaretlenmemiş olacaktır. Hemen görelim:

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

from Tkinter import *

pencere = Tk()

d = IntVar()
d.set(0)

btn1 = Checkbutton(text="Kubuntu", variable=d)
btn1.place(relx=0.0,rely=0.1)

mainloop()
```

Burada dikkatimizi çeken bir farklılık, “btn1” adlı onay kutusunun “seçenekleri” arasına “variable” adlı bir seçeneğin girmiş olması... Bu seçeneğin işlevini az çok anlamış olmalısınız: Yukarıda belirlediğimiz “d = IntVar()” ve “d.set(0)” ifadeleri ile “Checkbutton” pencere aracı arasında bağlantı kuruyor. Yani bu seçenek yardımıyla pencerece aracına, “değişken olarak d’yi kullan” emrini veriyoruz.

Bu kodları çalıştırdığımızda karşımıza içinde bir onay kutusu bulunan bir pencere açılır. Bu penceredeki onay kutusu “seçilmemiş” haldedir. İsterseniz yukarıdaki kodlar içinde yer alan “d.set(0)” ifadesini “d.set(1)” olarak değiştirip kodlarımızı öyle çalıştıralım:

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

from Tkinter import *
```

```
pencere = Tk()

d = IntVar()
d.set(1)

btn1 = Checkbutton(text="Kubuntu", variable=d)
btn1.place(relx=0.0,rely=0.1)

mainloop()
```

Gördüğünüz gibi, bu defa oluşan onay kutusu “seçili” vaziyette... Eğer yukarıdaki kodlar içindeki “d.set()” satırını tamamen kaldırırsak, Tkinter varsayılan olarak d.set() için “0” değerini atayacaktır.

Gelin şimdi bu kodları biraz geliştirelim. Mesela penceremizde bir “Entry” aracı olsun ve onay kutusunu seçtiğimiz zaman bu “Entry” aracında bizim belirlediğimiz bir cümle görünsün... Bu kodlar içinde göreceğimiz ”.get” ifadesine aslında pek yabancı sayılmayız. Bunu daha önceden “Entry” aracını kullanırken de görmüştük. Bu ifade yardımıyla bir değişkeninin değerini sonradan kullanmak üzere depolayabiliyoruz. Aşağıdaki örneği dikkatle inceleyin:

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

from Tkinter import *

pencere = Tk()

# Onay kutusu için bir değişken oluşturuyoruz
v = IntVar()
# Bu değişkene değer olarak "0" atayalım
v.set(0)

# Öbür onay kutusu için başka bir değişken daha oluşturuyoruz
z = IntVar()
# Bu değişkene de "0" değerini atıyoruz.
z.set(0)
# Bu kez bir karakter değişkeni oluşturalım
d = StringVar()
# Bu değişkenin değeri "Kubuntu" olsun.
d.set("Kubuntu")
# Bir tane daha karakter değişkeni oluşturalım
e = StringVar()
# Bunun da değeri "Debian" olsun.
e.set("Debian")

# Şimdi onay kutularını seçili hale getirdiğimizde çalışmasını
#istediğimiz komut için bir fonksiyon oluşturuyoruz:
def onay():
    # Eğer "v" değişkeninin değeri "1" ise....
    if v.get() == 1:
        # d.get() ile "d" değişkeninin değerini alıyoruz...
        giris.insert(0,"Merhaba %s kullanıcısı" %d.get())

    # Yok eğer "z" değişkeninin değeri "1" ise...
    elif z.get() == 1:
        giris.delete(0,END)
        # e.get() ile "e" değişkeninin değerini alıyoruz...
        giris.insert(0,"Merhaba %s kullanıcısı" %e.get())
```

```
    # Değişkenlerin değer "1" değil, başka bir değer ise..."
    else:
        giris.delete(0,END)

# Aşağıda "text" seçeneğinin değerinin "d.get()" olduğuna dikkat edin.
onay_pa = Checkbutton(text=d.get(), variable=v, command=onay)
onay_pa.place(relx = 0.0, rely = 0.1)
# Aşağıda "text" seçeneğinin değerinin "e.get()" olduğuna dikkat edin.
onay_de = Checkbutton(text=e.get(), variable=z, command=onay)
onay_de.place(relx= 0.0, rely= 0.2)
giris = Entry(width=24)
giris.place(relx = 0.0, rely=0.0)
mainloop()
```

## 4.2 “Toplevel” Pencere Aracı

Toplevel aracı bizim ana pencere dışında farklı bir pencere daha açmamızı sağlar. Mesela bir ana pencere üstündeki düğmeye bastığınızda başka bir pencere daha açılsın istiyorsanız bu işlevi kullanmanız gerekiyor. En basit kullanımı şöyledir:

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

from Tkinter import *

pencere = Tk()
pencere2 = Toplevel()

mainloop()
```

Bu kodları çalıştırdığımızda ekranda ikinci bir pencerenin daha açıldığını görürüz. Diyelim ki bu ikinci pencerenin, bir düğmeye basıldığında açılmasını istiyoruz:

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

from Tkinter import *
pencere = Tk()

def ekle():
    pencere2 = Toplevel()

btn_pen2 = Button(text="ekle", command=ekle)
btn_pen2.pack()

mainloop()
```

Gördüğünüz gibi pencere2’nin açılışını bir fonksiyona bağladık. Ardından da ana pencere üzerinde btn_pen2 adlı bir düğme oluşturduk ve bu düğmeye “command” seçeneği yardımıyla yukarıda tanımladığımız fonksiyonu atayarak düğmeye basılınca ikinci pencerenin açılmasını sağladık.

Eğer açılan ikinci pencere üzerinde de başka düğmeler olsun istiyorsak şu şekilde hareket etmemiz gerekir:

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

from Tkinter import *
pencere = Tk()

def ekle():
    pencere2 = Toplevel()
    btn_pen = Button(pencere2,text="çıkış", command=pencere2.destroy)
    btn_pen.pack()

btn_pen2 = Button(pencere,text="ekle", command=ekle)
btn_pen2.pack()

mainloop()
```

Yine gördüğünüz gibi, ilk tanımladığımız ekle fonksiyonu altında bir düğme oluşturduk. Burada yeni bir durum dikkatinizi çekmiş olmalı. Ekle fonksiyonu altında yeni tanımladığımız düğmede ilave olarak "text" seçeneğinden önce bir "pencere2" ifadesini görüyoruz. Bunu yapmamızın nedeni şu: Elimizde "pencere" ve "pencere2" adında iki adet pencere var. Böyle bir durumda oluşturulan pencere aracının hangi pencere üzerinde gösterileceğini Tkinter'e anlatmamız gerekiyor. Sadece bir pencere varken pencereleri açıkça belirtmeye gerek olmuyordu, ama eğer birden pencere var ve siz oluşturulan düğmenin hangi pencere görüntüleneceğini belirtmezseniz, örneğin bizim durumumuzda Tkinter "ekle" adlı düğmeye her basışta otomatik olarak ana pencere üzerinde "çıkış" adlı bir düğme oluşturacaktır. Anladığınız gibi, eğer aksi belirtilmemişse, pencere araçları otomatikman ana pencere üzerinde açılacaktır...

Diyelim ki "a", "b", "c" ve "d" adlarında dört adet penceremiz var. İşte hangi düğmenin hangi pencerede görüntüleneceğini belirlemek için bu özellikten faydalanacağız:

```
btn1 = Button(a, text="btn1")
btn2 = Button(b, text="btn2")
btn3 = Button(c, text="btn3")
btn4 = Button(d, text="btn4")
```

yukarıdaki kodlarda yeni bir özellik olarak bir de "command=pencere2.destroy" seçeneğini görüyoruz. Aslında `destroy` komutu biraz `quit` komutuna benziyor; görevi mevcut pencereyi kapatmak. Peki burada pencere2.destroy komutu yerine pencere2.quit komutunu kullanamaz mıyız? Tabii ki kullanabiliriz, ancak kullanırsak "çıkış" düğmesine bastığımızda sadece pencere2 değil, doğrudan ana pencerenin kendisi de kapanacaktır. Eğer ikinci pencerenin kapanıp ana pencerenin açık kalmasını istiyorsanız kullanmanız gereken komut `destroy`; yok eğer ikinci pencere kapandığında ana pencere de kapansın istiyorsanız kullanmanız gereken komut `quit` olmalıdır. btn_pen2 adlı düğmeyi ise, "text" seçeneğinin hemen önüne yazdığımız "pencere" ifadesinden de anlayacağınız gibi "pencere" adlı pencerenin üzerine yerleştiriyoruz. Yukarıda bahsettiğimiz gibi burada "pencere" ifadesini kullanmasanız da olur, çünkü zaten Tkinter siz belirtmesiniz de düğmeyi otomatik olarak ana pencere üzerinde açacaktır. Tabii düğmeyi ana pencerede değil de ikincil pencereler üzerinde açmak isterseniz bunu Tkinter'e açıkça söylemeniz gerekir...

## 4.3 "Listbox" Pencere Aracı

Bu araç bize pencereler üzerinde bir "liste içeren kutu" hazırlama imkanı veriyor. Hemen basit bir "Listbox" kullanımı örneği görelim:

```
liste = Listbox()
liste.pack()
```

Gördüğünüz gibi, bu aracın da kullanımı öteki araçlardan hiç farklı değil... İsterseniz bu aracı bir de bağlamında görelim:

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

from Tkinter import *

pencere = Tk()

liste = Listbox()
liste.pack()

mainloop()
```

Bu haliyle pencerenin tamamımını kapladığı için çok belirgin olmayabilir liste kutumuz... Ama pencereye bir de düğme eklersek liste kutusu daha rahat seçilebilir hale gelecektir:

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

from Tkinter import *

pencere = Tk()

liste = Listbox()
liste.pack()

btn = Button(text="ekle")
btn.pack()

mainloop()
```

Hatta şimdiye kadar öğrendiğimiz başka özellikleri kullanarak daha şık bir görünüm de elde edebiliriz:

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

from Tkinter import *

pencere = Tk()

liste = Listbox(bg="white")
liste.pack()

etiket = Label(text="####################", fg="magenta", bg="light green")
etiket.pack()

btn = Button(text="ekle",bg="orange",fg="navy")
btn.pack()

etiket2 = Label(text="####################", fg="magenta", bg="light green")
etiket2.pack()
```

```
mainloop()
```

Tabii ki siz pencere araçlarını ve yaratıcılığınızı kullanarak çok daha çekici görünümler ve renkler elde edebilirsiniz...

Şimdi liste kutumuza bazı öğeler ekleyelim. Bunun için şu basit ifadeyi kullanacağız:

```
liste.insert(END,"öğe 1")
```

Tabii ki bunu bu şekilde tek başına kullanamayız. Hemen bu parçacığı yukarıdaki kod içine yerleştirelim:

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

from Tkinter import *

pencere = Tk()

liste = Listbox(bg="white")
liste.insert(END,"öğe 1")
liste.pack()

etiket = Label(text="####################", fg="magenta", bg="light green")
etiket.pack()

btn = Button(text="ekle",bg="orange",fg="navy")
btn.pack()

etiket2 = Label(text="####################", fg="magenta", bg="light green")
etiket2.pack()

mainloop()
```

Gördüğünüz gibi, bu parçacığı, liste kutusunu tanımladığımız satırın hemen altına ekledik. Biz liste kutumuza aynı anda birden fazla öğe de eklemek isteyebiliriz. Bunun için basitçe bir “for döngüsü” kullanabiliriz:

```
gnulinux_dagitimlari = ["Pardus", "Debian", "Ubuntu", "PclinuxOS",
                        "TruvaLinux", "Gelecek Linux"]

for i in gnulinux_dagitimlari:
    liste.insert(END, i)
```

Hemen bu kodları da yerli yerine koyalım:

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

from Tkinter import *

pencere = Tk()

liste = Listbox(bg="white")
liste.pack()

gnulinux_dagitimlari = ["Pardus", "Debian", "Ubuntu", "PclinuxOS",
                        "TruvaLinux", "Gelecek Linux"]
```

```
for i in gnulinux_dagitimlari:
    liste.insert(END, i)

etiket = Label(text="#################", fg="magenta", bg="light green")
etiket.pack()

btn = Button(text="ekle",bg="orange",fg="navy")
btn.pack()

etiket2 = Label(text="#################", fg="magenta", bg="light green")
etiket2.pack()

mainloop()
```

Gayet güzel bir liste kutusu oluşturduk ve listemizdeki öğeleri de rahatlıkla seçebiliyoruz... Yalnız dikkat ettiyseniz ana pencere üzerindeki ekle düğmesi şu anda hiçbir işe yaramıyor... Tabii ki onu oraya boşu boşuna koymadık.. Hemen bu düğmeye de bir işlev atayalım... Mesela, bu düğmeye basıldığında ayrı bir pencere açılsın ve kullanıcıdan girdi alarak ana penceredeki liste kutusuna eklesin... Tabii ki bu yeni açılan pencerede de bir giriş kutusu ve bir de işlemi tamamlamak için bir düğme olsun. Hemen işe koyulalım:

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

from Tkinter import *

pencere = Tk()

liste = Listbox(bg="white")
liste.pack()

gnulinux_dagitimlari = ["Pardus", "Debian", "Ubuntu", "PclinuxOS",
                        "TruvaLinux", "Gelecek Linux"]

for i in gnulinux_dagitimlari:
    liste.insert(END, i)

def yeni():
    global giris
    pencere2 = Toplevel()
    giris = Entry(pencere2)
    giris.pack()
    btn2 = Button(pencere2, text="tamam",command=ekle)
    btn2.pack()

def ekle():
    liste.insert(END,giris.get())
    giris.delete(0,END)

etiket = Label(text="#################", fg="magenta", bg="light green")
etiket.pack()

btn = Button(text="ekle",bg="orange",fg="navy", command=yeni)
btn.pack()

etiket2 = Label(text="#################", fg="magenta", bg="light green")
etiket2.pack()
```

```
mainloop()
```

İsterseniz daha anlaşılır olması için parça parça ilerleyelim:

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

from Tkinter import *
pencere = Tk()
```

Bu kısmı zaten biliyoruz... Herhangi bir açıklamaya gerek yok...

```
liste = Listbox(bg="white")
liste.pack()

gnulinux_dagitimlari = ["Pardus", "Debian", "Ubuntu", "PclinuxOS",
                        "TruvaLinux", "Gelecek Linux"]

for i in gnulinux_dagitimlari:
    liste.insert(END, i)
```

Bu kısımda bildiğiniz gibi önce “liste” adında bir “liste kutusu” (Listbox) oluşturduk. Liste kutumuzun arkaplan rengini de beyaz olarak belirledik.

Ardından da “gnulinux_dagitimlari” adında bir liste oluşturduk ve tek tek öğelerini yazdık... Hemen alt satırda bir “for döngüsü” yardımıyla gnulinux_dagitimlari adlı listedeki bütün öğeleri “liste” adı verdiğimiz “liste kutusu” içine yerleştirdik:

```
def yeni():
    global giris
    pencere2 = Toplevel()
    giris = Entry(pencere2)
    giris.pack()
    btn2 = Button(pencere2, text="tamam",command=ekle)
    btn2.pack()
```

Burada yaptığımız iş basit bir fonksiyon oluşturmaktan ibaret... Önce “yeni” adlı bir fonksiyon oluşturuyoruz. Ardından “pencere2” adıyla ana pencereden ayrı bir pencere daha oluşturuyoruz.. Bu yeni pencere, ileride “ekle” tuşuna bastığımızda açılmasını istediğimiz pencere oluyor... Alt satırda, bu yeni pencere üzerine yerleştirmek için “giris” adlı bir “Entry” pencere aracı oluşturuyoruz. Parantez içine “pencere2” yazmayı unutmuyoruz, çünkü bu “Entry” aracının oluşmasını istediğimiz yer ikinci pencere... Dikkat ettiyseniz fonksiyonu tanımlarken “global giris” adlı bir satır daha ekledik... Bu satırın amacı, fonksiyon içindeki “giris” adlı “Entry” aracını fonksiyon dışında da kullanabilmek... Çünkü bu “Entry” aracı bize daha sonra da lazım olacak... “Entry” aracına benzer şekilde bir de “btn2” adıyla bir düğme oluşturuyoruz. Bunu da ikinci penceremize yerleştiriyor, adını “tamam” koyuyor ve komut olarak aşağıda tanımlayacağımız “ekle” fonksiyonunu seçiyoruz:

```
def ekle():
    liste.insert(END,giris.get())
    giris.delete(0,END)
```

İşte bu parçada da “ekle” adlı bir fonksiyon oluşturduk. Burada “liste.insert” ifadesi “liste” adlı “liste kutusuna” ekleme yapmamızı sağlıyor. Parantez içindeki “giris.get()” ifadesi size tanıdık geliyor olmalı.. Çünkü aynı ifadeyi “Entry” pencere aracını anlatırken de görmüştük... Hatırlarsanız bu ifade sayesinde “Entry” aracına kullanıcı tarafından girilen verileri daha sonra kullanmak amacıyla elimize tutabiliyorduk... İşte burada da bu ifade yardımıyla “giris” adlı

“Entry” pencere aracının içeriğini alıp “liste” adlı “liste kutusu” içine yerleştiriyoruz. Alt satırdaki “giris.delete(0,END)” ifadesi ise “Entry” aracına kullanıcı tarafından giriş yapıldıktan sonra kutunun boşaltılmasını sağlıyor:

```
etiket = Label(text="################", fg="magenta", bg="light green")
etiket.pack()
btn = Button(text="ekle",bg="orange",fg="navy", command=yeni)
btn.pack()
etiket2 = Label(text="################", fg="magenta", bg="light green")
etiket2.pack()
mainloop()
```

Bu son parçada bilmediğimiz hiçbir şey yok... Normal bir şekilde “etiket” adı verdiğimiz “Label” aracını tanımlıyoruz... Burada “Label” aracını süsleme amacıyla nasıl kullandığımıza dikkat edin.... “fg” ve “bg” seçeneklerini de önceki bölümlerden hatırlıyoruz. “fg” önplandaki rengi; “bg” ise arkaplandaki rengi seçmemizi sağlıyor. “magenta” ve “light green” ise kullanacağımız renklerin adları oluyor. Bunun altında ise basit bir “Button” aracı tanımlıyoruz. İsmini ve renklerini belirledikten sonra da “command” seçeneği yardımıyla yukarıda tanımladığımız “yeni” adlı fonksiyonu bu düğmeye bağlıyoruz. Bunun aşağısındaki “etiket2”nin ise “etiket” adlı araçtan hiçbir farkı yok...

Kodlarımızı çalıştırdığımızda karşımıza gayet hoş, üstelik güzel de bir işlevi olan bir pencere çıkıyor. Burada “ekle” düğmesine bastığımızda karşımıza yeni bir pencere açılıyor. Bu yeni pencerede, kullanıcının giriş yapabilmesi için bir “Entry” aracı, bir de işlemi tamamlayabilmesi için “Button” aracı yer alıyor. Kullanıcı “Entry” aracına bir veri girip “tamam” düğmesine bastığında “Entry” aracına girilen veri ana penceredeki “liste kutusu”na ekleniyor... Ayrıca ilk veri girişinin ardından “Entry” aracı içindeki alan tamamen boşaltılıyor ki kullanıcı rahatlıkla ikinci veriyi girebilsin... Çok hoş değil mi?!

Siz de burada kendinize göre değişiklikler yaparak özellikle ikinci pencereyi göze daha hoş görünecek bir hale getirebilirsiniz...

Burada dikkat ederseniz, ikinci pencerede, giriş kutusuna hiçbir şey yazmadan “tamam” düğmesine basarsak ana penceredeki liste kutusuna boş bir satır ekleniyor. Şimdi öyle bir kod yazalım ki, kullanıcı eğer ikinci penceredeki giriş kutusuna hiçbir şey yazmadan “tamam” düğmesine basarsa giriş kutusu içinde “Veri Yok!” yazısı belirsin ve bu yazı ana penceredeki liste kutusuna eklenmesin:

Bunun için kodlarımız içindeki “ekle” fonksiyonuna iki adet “if” koşulu eklememiz gerekiyor...

```
def ekle():
    if not giris.get():
        giris.insert(END,"Veri Yok!")
    if giris.get() != "Veri Yok!":
        liste.insert(END,giris.get())
        giris.delete(0,END)
```

Gördüğünüz gibi “giris” boşken “tamam” tuşuna basıldığında “Veri Yok!” ifadesi ekrana yazdırılıyor... Ancak burada şöyle bir problem var: Eğer “Veri Yok!” ifadesi ekrana yazdırıldıktan sonra kullanıcı bu ifadeyi silmeden bu ifadenin yanına bir şeyler yazıp “tamam”a basarsa “Veri Yok!” ifadesiyle birlikte o yeni yazdığı şeyler de listeye eklenecektir... Bunu engellemek için kodumuzu şu hale getirebiliriz:

```
def ekle():
    if not giris.get():
        giris.insert(END,"Veri Yok!")
    if not "Veri Yok!" in giris.get():
        liste.insert(END,giris.get())
```

```
        giris.delete(0,END)
```

Yani şöyle demiş oluyoruz bu ifadelerle: Eğer “giris” adlı “Entry” aracı boş ise, araç içinde “Veri Yok!” ifadesini göster. Eğer “giris” adlı “Entry” aracı içinde “Veri Yok!” ifadesi bulunmuyorsa, “liste” adlı “Listbox” aracına “giris” içindeki bütün veriyi yazdır...

Liste kutumuza öğelerimizi ekledik... Peki bu öğeleri silmek istersek ne yapacağız?

Niyetimiz liste kutusundan öğe silmek olduğunu göre en başta bir silme düğmesi oluşturmamız mantıklı olacaktır:

```
btn_sil = Button()
btn_sil.pack()
```

Bu düğmenin bir iş yapabilmesi için de bu düğmeye bir fonksiyon atamamız gerekir:

```
def sil():
    liste.delete(ACTIVE)
```

Burada gördüğünüz gibi, silme işlemi için “liste.delete” ifadesini kullanıyoruz... Parantez içindeki “ACTIVE” ifadesi ise liste kutusu içinden bir seçim yapıp “sil” düğmesine basınca bu seçili öğenin silinmesini sağlayacak... Yani “aktif” olan öğe silinecek. Bu iki parçayı öteki komutlarla birlikte bir görelim bakalım:

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

from Tkinter import *

pencere = Tk()

liste = Listbox(bg="white")
liste.pack()

gnulinux_dagitimlari = ["Pardus", "Debian", "Ubuntu", "PclinuxOS",
                        "TruvaLinux", "Gelecek Linux"]

for i in gnulinux_dagitimlari:
    liste.insert(END, i)

def yeni():
    global giris
    pencere2 = Toplevel()
    giris = Entry(pencere2)
    giris.pack()
    btn2 = Button(pencere2, text="tamam",command=ekle)
    btn2.pack()

def ekle():
    if not giris.get():
        giris.insert(END,"Veri Yok!")
    if not "Veri Yok!" in giris.get():
        liste.insert(END,giris.get())
        giris.delete(0,END)

def sil():
    liste.delete(ACTIVE)

etiket = Label(text="#################", fg="magenta", bg="light green")
```

```
etiket.pack()

btn = Button(text="ekle",bg="orange",fg="navy", command=yeni)
btn.pack()

btn_sil = Button(text="sil",bg="orange", fg="navy",command=sil)
btn_sil.pack()

etiket2 = Label(text="#################", fg="magenta", bg="light green")
etiket2.pack()

mainloop()
```

Tabii ki, sil düğmesinin görünüşünü pencere üzerindeki öteki öğelere uydurmak için “fg” ve “bg” seçenekleri yardımıyla ufak bir renk ayarı yapmayı da unutmadık...

Böylece bir pencere aracını daha bitirmiş olduk... Gelelim sıradaki pencere aracımıza:

## 4.4 “Menu” Pencere Aracı

Adından da anlaşılacağı gibi bu araç bize pencerelerimiz üzerinde menüler hazırlama olanağı sağlıyor... “Menu” pencere aracının kullanımı öteki araçlardan birazcık daha farklıdır, ama kesinlikle zor değil... Hemen küçücük bir menü hazırlayalım:

Önce standart satırlarımızı ekliyoruz:

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

from Tkinter import *
pencere = Tk()
```

Şimdi menümüzü oluşturmaya başlayabiliriz...

```
menu = Menu(pencere)
```

Burada “menu” adlı bir “Menu pencere aracı” oluşturduk... Adı “menu” olmak zorunda değil... Siz istediğiniz ismi kullanabilirsiniz... Ama tabii ki pencere aracımızın adı olan “Menu”yu değiştiremeyiz... Buraya kadar öteki pencere araçlarından hiçbir farkı yok...

Parantez içindeki “pencere” ifadesinden de anladığımız gibi, bu pencere aracını ana pencere üzerinde oluşturuyoruz. Hatırlayacağınız gibi burada “pencere” diye açık açık belirtmesek de Tkinter pencere aracımızı otomatik olarak ana pencere üzerinde oluşturacaktır. Aslında bu satır yardımıyla ana pencerenin en üstünde, sonradan gelecek menüler için bir “menü çubuğu” oluşturmuş oluyoruz:

```
pencere.config(menu=menu)
dosya = Menu(menu)
```

Burada “config” metodu yardımıyla öncelikle “menu” adlı aracı “pencere”ye bağlıyoruz. Parantez içindeki ilk “menu” ifadesi, tıpkı öteki pencere araçlarında gördüğümüz “text” ifadesi gibi bir “seçenek”... ikinci “menu” ifadesi ise yukarıda bizim “Menu” aracına kendi verdiğimiz isim... Bu isim herhangi bir kelime olabilirdi... Yani en başta menünün adını “kepap” olarak belirleseydik, burada “menu=kebap” da diyebilirdik...

İkinci satırda ise “dosya” adlı başka bir “Menu pencere aracı” daha oluşturuyoruz. Hatırlarsanız ilk Menu aracını oluştururken parantez içine “pencere” yazarak aracı pencereye bağlamıştık.

Bu kezse aracımızı bir önceki “menu”nün üzerinde oluşturuyoruz... Aslında bu satır yardımıyla bir önceki aşamada oluşturduğunuz “araç çubuğu” üzerinde “iner menü” (drop-down menu) için bir yer açmış oluyoruz:

```
menu.add_cascade(label="Dosya",menu=dosya)
```

Şimdi yapmamız gereken, menünün aşağıya doğru açılmasını yani “inmesini” sağlamak. Bu iş için yukarıda gördüğünüz “add_cascade” metodunu kullanıyoruz. Bu metodun “menu” adlı “menü araç çubuğuna” bağlı olduğuna dikkat edin. Parantez içinde gördüğümüz “label” ifadesi de tıpkı “text” gibi, menüye ad vermemizi sağlıyor. Menümüzün adını “Dosya” olarak belirledik. Parantez içindeki diğer ifade olan “menu” de “Dosya”nın hangi “araç çubuğu” üzerinde üzerinde oluşturulacağını gösteriyor...

```
dosya.add_command(label="Aç")
dosya.add_command(label="Kaydet")
dosya.add_command(label="Farklı Kaydet...")
dosya.add_command(label="Çıkış",command=pencere.quit)

mainloop()
```

Burada gördüğümüz ifadeler ise bir üstte oluşturduğumuz “Dosya” adlı menünün alt başlıklarını oluşturmamızı sağlıyor... Burada “add_command” metotlarının “dosya” adlı araç çubuğuna bağlandığına dikkat edin... Bu satırlardan anladığınız gibi, “Dosya” adlı menümüzün altında “Aç”, “Kaydet”, “Farklı Kaydet...” ve “Çıkış” gibi alt başlıklar olacak...

Şimdi kodlarımızın hepsini birlikte görelim:

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

from Tkinter import *

pencere = Tk()

menu = Menu(pencere)
pencere.config(menu=menu)
dosya = Menu(menu)
menu.add_cascade(label="Dosya",menu=dosya)
dosya.add_command(label="Aç")
dosya.add_command(label="Kaydet")
dosya.add_command(label="Farklı Kaydet...")
dosya.add_command(label="Çıkış",command=pencere.quit)

mainloop()
```

Bu kodları çalıştırdığınızda gayet güzel bir pencere elde etmiş olacaksınız... Yalnız dikkat ettiyseniz, “Dosya”ya bastıktan sonra açılan alt menünün en üstünde “———-” gibi bir şey görüyoruz... Oraya tıkladığımızda ise bütün menü içeriğinin tek bir grup halinde toplanıp ayrı bir pencere oluşturduğunu görüyoruz... Eğer bu özellikten hoşlanmadıysanız, bu minik çizgileri kodlar arasına “tearoff=0” ifadesini ekleyerek yok edebilirsiniz (“tearoff=0” ifadesini “dosya” adlı değişkeni oluştururken ekliyoruz...)

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

from Tkinter import *

pencere = Tk()
```

```
menu = Menu(pencere)
pencere.config(menu=menu)
dosya = Menu(menu, tearoff=0)
menu.add_cascade(label="Dosya",menu=dosya)
dosya.add_command(label="Aç")
dosya.add_command(label="Kaydet")
dosya.add_command(label="Farklı Kaydet...")
dosya.add_command(label="Çıkış",command=pencere.quit)

mainloop()
```

Konu belki biraz karışık gelmiş olabilir... Ama aslında hiç de öyle değil... İşin mantığını anlamak için yukarıdaki kodlarda geçen şu satırlar bize epeyce yardımcı olabilir:

```
menu = Menu(pencere)
dosya = Menu(menu, tearoff=0)
menu.add_cascade(label="Dosya",menu=dosya)
```

Dikkat ettiyseniz, önce "Menu pencere aracını" oluşturuyoruz. Bu araç ilk oluşturulduğunda, parantez içi ifadeden anladığımız gibi, "pencere" adlı ana pencere üzerine bağlanıyor...

İkinci satır vasıtasıyla ikinci kez bir "Menu pencere aracı" oluştururken ise, parantez içi ifadeden anlaşıldığı gibi, oluşturduğumuz menüyü bir önceki "Menu pencere aracı"na bağlıyoruz.

Üçüncü satırda "inen menü"yü oluştururken de, bunu bir önceki "Menu pencere aracı" olan "dosya"ya bağlıyoruz...

Menülere başka alt menüler de eklerken bu mantık çok daha kolay anlaşılıyor... Şöyle ki:

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

from Tkinter import *

pencere = Tk()

menu = Menu(pencere)
pencere.config(menu=menu)
dosya = Menu(menu, tearoff=0)
menu.add_cascade(label="Dosya",menu=dosya)
dosya.add_command(label="Aç")
dosya.add_command(label="Kaydet")
dosya.add_command(label="Farklı Kaydet...")
dosya.add_command(label="Çıkış",command=pencere.quit)
yeni = Menu(dosya,tearoff=0)
dosya.add_cascade(label="Yeni",menu=yeni)
yeni.add_command(label="Metin Belgesi")
yeni.add_command(label="Resim Dosyası")
yeni.add_command(label="pdf dokümanı")

mainloop()
```

Gördüğünüz gibi, bu kez "Menu pencere aracımızı" ilk olarak "dosya" adlı araç üzerine bağlıyoruz. Çünkü yeni pencere aracımız, bir önceki pencere aracı olan "dosya"nın üzerinde oluşturulacak. Bir sonraki satırda "add_command" metodunu kullanırken de alt menüleri "yeni" adlı "Menu pencere aracı" üzerine bağlıyoruz... Çünkü bu alt menüler "yeni"nin içinde yer alacak...

Aynı şekilde eğer "Dosya" başlığının yanına bir de mesela "Düzen" diye bir seçenek eklemek istersek şöyle bir bölüm ekliyoruz kodlarımız arasına:

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

from Tkinter import *

pencere= Tk()

menu= Menu(pencere)
pencere.config(menu=menu)
dosya= Menu(menu, tearoff=0)
menu.add_cascade(label="Dosya",menu=dosya)
dosya.add_command(label="Aç")
dosya.add_command(label="Kaydet")
dosya.add_command(label="Farklı Kaydet...")
dosya.add_command(label="Çıkış",command=pencere.quit)
yeni= Menu(dosya,tearoff=0)
dosya.add_cascade(label="Yeni",menu=yeni)
yeni.add_command(label="Metin Belgesi")
yeni.add_command(label="Resim Dosyası")
yeni.add_command(label="pdf dokümanı")
dosya2= Menu(menu,tearoff=0)
menu.add_cascade(label="Düzen",menu=dosya2)
dosya2.add_command(label="Bul")

mainloop()
```

## 4.5 “Text” Pencere Aracı

Şimdiye kadar bir pencerenin sahp olması gereken pek çok özelliği gördük. Hatta pencerelerimize menüler dahi ekledik... Bütün bunların dışında öğrenmemiz gereken çok önemli bir pencere aracı daha var. O da, “Text” adlı pencere aracıdır.. Bu araç sayesinde birden fazla satır içeren metinler oluşturabileceğiz. En basit haliyle “Text” adlı pencere aracını şu şekilde oluşturabiliriz:

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

from Tkinter import *

pencere= Tk()
metin= Text()
metin.pack()

mainloop()
```

Oluştrduğumuz bu “Text” aracı pek çok işlevi yerine getirebilecek durumdadır: Bu araç içine şu haliyle istediğimiz uzunlukta metin girebiliriz, klavye ve fareyi kullanarak metni yönetebiliriz, hatta oluşturduğumuz bu “Text” aracını oldukça basit bir “metin editörü” olarak da kullanabiliriz. Eğer oluşturduğumuz bu “Text” aracı içine öntanımlı olarak herhangi bir metin yerleştirmek istersek şu kodu kullanmamız gerekir:

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

from Tkinter import *
```

```
pencere= Tk()

metin= Text(fg = "blue",font="Helvetica 13 bold")
metin.insert(END,"Sürüm 0.1.1")
metin.pack()

mainloop()
```

Gördüğünüz gibi, “Text” aracını oluştururken, önceki yazılarda öğrendiğimiz şekilde “fg” seçeneği yardımıyla metni mavi yaptık. “font” seçeneği yardımıyla ise yazı tipini, “Helvetica, 13, koyu ve altı çizili” olarak belirledik.

Kodlarımız içinde kullandığımız “metin.insert” ifadesi de bize öntanımlı bir metin girme imkanı sağladı. Parantez içinde belirttiğimiz “END” ifadesi öntanımlı olarak yerleştireceğimiz metnin pencere aracının neresinde yer alacağını gösteriyor.

Yukarıda verdiğimiz kodu değiştirerek isterseniz daha çekici bir görünüm de elde edebilirsiniz:

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

from Tkinter import *

pencere= Tk()
a= "Sürüm 0.1.1"
metin= Text(bg="orange",fg = "blue",font="Helvetica 13 bold")
metin.insert(END,a.center(112,"*"))
metin.pack()

mainloop()
```

Eğer bir metin içinden herhangi bir bölümü almak isterseniz kullanmanız gereken şey “get” ifadesidir. Bu “get” ifadesinin nasıl kullanıldığını görelim şimdi:

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

from Tkinter import *

pencere= Tk()

metin= Text()
metin.pack()
metin.get(1.0,END)

mainloop()
```

Yukarıdaki örneği sadece “get” ifadesinin nasıl kullanıldığını göstermek için verdik... Şu haliyle bu kod bizim beklentilerimizi karşılayamaz... Çünkü “get” ifadesi yardımıyla metni aldık, ama aldığımız bu metni kullanmamızı sağlayacak araçları henüz penceremize yerleştirmediğimiz için “get” ifadesinin bize sağladığı işlevi kullanamıyoruz. Şimdilik burada şuna dikkat edelim: “metin.get()” gibi bir ifade kullanırken parantez içinde belirttiğimiz ilk sayı 1.0. Bu rakam metin kutusunun ilk satırının ilk sütununa işaret ediyor. Burada ilk satırın 1’den; ilk sütunun ise 0’dan başladığına dikkat edelim. Virgülden sonra gelen “END” ifadesi ise “Text” aracı içindeki metnin en sonuna işaret ediyor. (İngilizce’de “END” kelimesi “SON” anlamına geliyor). Yani bu koda göre “get” ifadesi yardımıyla Text aracı içindeki bir metni, en başından en sonuna kadar alabiliyoruz. İsterseniz parantez içinde farklı sayılar belirterek, alınacak metnin ilk ve son konumlarını belirleyebiliriz. Mesela şu koda bir bakalım:

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

from Tkinter import *

pencere= Tk()

metin= Text()
metin.pack()
metin.get(1.0,1.5)

mainloop()
```

Burada ise “Text” aracının birinci satırı ve birinci sütunundan, birinci satırı ve beşinci sütununa kadar olan aralıktaki metin alınacaktır.

Şimdi henüz hiçbir iş yapmayan bu kodları biraz işlevli bir hale getirelim:

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

from Tkinter import *

pencere= Tk()

def al():
    a= metin.get(1.0,END)
    giris.insert(0, a)

metin= Text()
metin.pack()

btn= Button(text="al", command=al)
btn.pack()

giris= Entry()
giris.pack()

mainloop()
```

Burada öncelikle “Text” aracı içindeki metnin tamamını alıp (metin.get(1.0,END)) “giris” adlı pencere aracına yerleştiren (giris.insert(0,a)) bir fonksiyon oluşturduk. Dikkat ederseniz kullanım kolaylığı açısından “metin.get(1.0,END)” ifadesini “a” adlı bir değişkene atadık.

Daha sonra “metin” adlı “Text” aracımızı ve “btn” adlı “Button” aracımızı oluşturduk. “Button” aracımıza “komut” (command) olarak yukarıda tanımladığımız fonksiyonu göstererek “Button” ile fonksiyon arasında ilişki kurduk.

En sonunda da “giris” adlı “Entry” aracımızı tamamlayarak kodumuzu sona erdirdik.

Bu kodları çalıştırdığımızda karşımıza çıkan boş metin kutusuna herhangi bir şey yazıp alttaki düğmeye basınca, metin kutusunun bütün içeriği düğmenin hemen altındaki küçük metin kutusuna işlenecektir.

Şimdi daha karmaşık bir örnek yapalım...

Aşağıdaki örneği dikkatlice inceleyin:

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

```
from Tkinter import *

pencere= Tk()

def al():
    metin2.insert(END,metin.get(2.0,END))

a= "Sürüm 0.1.1"

metin= Text(height=15,bg="black",fg = "white",font="Helvetica 13 bold")
metin.insert(END,a.center(112,"*"))
metin.pack()

metin2= Text(height=15,width=115, bg="light blue",fg="red")
metin2.pack()

btn= Button(text="al",command=al)
btn.pack()

mainloop()
```

Yukarıdaki kodlarda bir metni ve pencere araçlarını nasıl biçimlendirdiğimize dikkat edin. Ayrıca Python'da karakter dizilerine ait bir metot olan "center" yardımıyla bir kelimenin sol-una ve sağına nasıl başka karakterler yerleştirdiğimizi inceleyin. Bu kodlar içinde kendinize göre bazı denemeler yaparak neyin ne işe yaradığını daha iyi anlayabilirsiniz.

Yukarıda bahsettiğimiz "metnin koordinatlarını verme" yöntemi her zaman tercih edilecek bir durum değildir... Ne de olsa kullanıcılarınızdan satır/sütun saymasını bekleyemezsiniz! Her-halde bu gibi durumlarda en iyi yöntem "seçilen metnin alınması" olacaktır. Bunu da basitçe şu kodlar yardımıyla yapıyoruz:

```
metin.get("sel.first","sel.last")
```

Burada "sel.first" ifadesi "seçimin başlangıç noktasını"; "sel.last" ifadesi ise "seçimin bitiş noktasını" gösteriyor.

Şimdi bu kod parçasını bir bağlam içinde kullanalım:

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

from Tkinter import *

pencere= Tk()

def al():
    a= metin.get("sel.first","sel.last")
    metin2.insert(END,a)

metin= Text(height=10,width=35,bg="white",fg ="blue",font="Helvetica 13 bold")
metin.pack()

metin2= Text(height=10,width=50, bg="black",fg="white")
metin2.pack()

btn= Button(text="al",command=al)
btn.pack()
```

```
mainloop()
```

Bu kodları çalıştırdığımızda, üstteki kutuya yazdığımız metnin istediğimiz bir kısmını seçtikten sonra alttaki düğmeye basarsak, seçili kısım ikinci kutuya işlenecektir.

## 4.6 “Scrollbar” Pencere Aracı

Scrollbar adlı pencere aracı, Tkinter ile yazdığımız uygulamalara kaydırma çubuğu ekleme imkanı veriyor. Bu pencere aracının özelliği, öteki pencere araçlarının aksine tek başına kullanılmamasıdır. Kaydırma çubuğu kavramının mantığı gereği, bu pencere aracını başka pencere araçları ile birlikte kullanacağız. Mesela “Text” pencere aracılığıyla bir metin kutusu oluşturmuş isek, şimdi öğreneceğimiz “Scrollbar” adlı pencere aracı sayesinde bu metin kutusuna bir kaydırma çubuğu ekleyebileceğiz. İsterseniz hemen ufak bir örnek yapalım:

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

from Tkinter import *

pencere = Tk()
```

Buraya kadar olan kısımda bir yenilik yok. Artık ezbere bildiğimiz komutlar bunlar... Devam ediyoruz:

```
kaydirma = Scrollbar(pencere)
kaydirma.pack(side=RIGHT,fill=Y)
```

Gördüğünüz gibi, “Scrollbar” adlı pencere aracımızı oluşturup, bunu “pack” yöneticisiyle paketledik. pack() içindeki argümanlara dikkat edin. “side=RIGHT” ifadesi yardımıyla kaydırma çubuğumuzu sağa yaslarken, “fill=Y” ifadesi yardımıyla da Y düzlemi üzerinde (yani yukarıdan aşağıya doğru) boylu boyunca uzatıyoruz. Devam edelim:

```
metin = Text(yscrollcommand=kaydirma.set)
metin.pack()
```

Kaydırma çubuğunu bir metin kutusu içinde kullanacağımız için, “Text” pencere aracı vasıtasıyla metin kutumuzu tanımlıyoruz. Burada, “yscrollcommand=kaydirma.set” gibi bir ifade kullandığımıza dikkat edin. Bu ifade ile kaydırma işlemini etkinleştiriyoruz. Şimdi son darbeyi vurabiliriz:

```
kaydirma.config(command=metin.yview)

mainloop()
```

Burada “kaydirma.config()” satırını, metin aracını tanımladıktan sonra yazmamız önemli. Eğer metin aracını tanımlamadan config satırını yazarsak komut satırında hataların uçuştuğunu görürüz...

Dilerseniz yukarıda parça parça verdiğimiz uygulamayı bir de topluca görelim:

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

from Tkinter import *

pencere = Tk()
```

```
kaydirma = Scrollbar(pencere)
kaydirma.pack(side=RIGHT,fill=Y)

metin = Text(yscrollcommand=kaydirma.set)
metin.pack()

kaydirma.config(command=metin.yview)

mainloop()
```

Böylece metin kutusuna yazdığımız yazılar aşağıya doğru uzadıkça sağ tarafta bir kaydırma çubuğu görüntülenecektir.

Peki kaydırma çubuğunun, ekranın alt tarafında görünmesini istersek ne olacak? Yani yukarıdaki örnekte yaptığımız gibi, aşağıya doğru giden metinleri değil de, yana doğru giden metinleri kaydırmak istersek ne yapacağız? Çok kolay. Adım adım gidelim yine:

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

from Tkinter import *

pencere = Tk()
```

...devam ediyoruz...

```
kaydirma = Scrollbar(pencere, orient=HORIZONTAL)
kaydirma.pack(side=BOTTOM,fill=X)
```

Gördüğünüz gibi, kaydırma çubuğunun pozisyonunu belirlemek için yeni bir seçenekten yararlanıyoruz. Bu yeni seçeneğin adı "orient". Buna vermemiz gereken değer ise "HORIZONTAL". Bu İngilizce kelime Türkçe'de "yatay" anlamına geliyor. Kaydırma çubuğumuz ekranda düşey değil, yatay vaziyette görüneceği için bu seçeneği kullandık. Ayrıca "pack" parametrelerine de dikkat edin. "side" seçeneğine "BOTTOM" değerini verdik. Çünkü kaydırma çubuğumuzun ekranın alt tarafında görünmesini istiyoruz. Bir önceki örnekte, çubuğun ekranın sağında görünmesini istediğimiz için "side=RIGHT" şeklinde bir kullanım benimsemiştik ("RIGHT" kelimesi "sağ" anlamına gelirken, "BOTTOM" kelimesi "dip, alt" gibi anlamlara gelmektedir). Bu arada "fill" seçeneğinin değerinin de, bir önceki örneğin aksine, "X" olduğuna dikkat edin. Çünkü bu defa kaydırma çubuğumuzun x düzlemi üzerinde (yani soldan sağa doğru) boylu boyunca uzanması gerekiyor... Devam ediyoruz:

```
metin = Text(wrap=NONE, xscrollcommand=kaydirma.set)
metin.pack()
```

Şimdi de metin kutumuzu tanımladık. Buradaki parametrelere dikkat edin. Öncelikle metin kutumuza "wrap" adlı bir seçenek tanımlıyoruz. Bu "wrap" seçeneği metin kutusuna yazılan yazıların, ekranın sağ sınırına ulaşınca bir satır kaydırılıp kaydırılmayacağını belirliyor. Varsayılan olarak, yazılan metin ekranın sağ sınırına dayandığında otomatik olarak alt satıra geçecektir. Ama eğer biz bu "wrap" seçeneğine "NONE" değerini verirsek, metin sınırı geçip sağa doğru uzamaya devam edecektir. İşte aynı parantez içinde belirttiğimiz "xscrollcommand=kaydirma.set" seçeneği sayesinde sağa doğru uzayan bu metni kaydırabileceğiz. Burada, yukarıdaki örnekten farklı olarak "xscrollcommand" ifadesini kullandığımıza dikkat edin. Bir önceki örnekte "yscrollcommand" ifadesini kullanmıştık. Bu farkın nedeni, bir önceki örnekte "y" düzlemi üzerinde çalışırken, şimdiki örnekte "x" düzlemi üzerinde çalışıyor olmamız...

Artık son darbeyi vurabiliriz:

```
kaydirma.config(command=metin.xview)

mainloop()
```

Gördüğünüz gibi, burada da bir önceki örnekten farklı olarak “xview” ifadesini kullandık. Bir önceki örnekte “yview” ifadesini kullanmıştık. Bu farkın nedenini söylemeye gerek yok...

Gelin isterseniz bu parçalı örneği bir araya getirelim:

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

from Tkinter import *

pencere = Tk()

kaydirma = Scrollbar(pencere, orient=HORIZONTAL)
kaydirma.pack(side=BOTTOM,fill=X)

metin = Text(wrap=NONE, xscrollcommand=kaydirma.set)
metin.pack()

kaydirma.config(command=metin.xview)

mainloop()
```

BÖLÜM 5

# Tkinter Uygulamalarını Güzelleştirmek

Tkinter arayüz takımı konusunda en fazla şikayet edilen şey Tkinter ile yazılan uygulamaların "çirkin" görünmesidir. Bu bölümde bu sıkıntıyı aşmanın bazı yollarını göstermeye çalışacağız. Ayrıca bu bölümde, "güzellik" ile ilgili olduğunu düşündüğüm başka konulara da değineceğiz. Mesela bu bölümde, Tkinter uygulamaları içinde kullandığımız pencere araçlarına nasıl simge yerleştirebileceğimizi de inceleyeceğiz. Bu bölüm bir yandan da örnek uygulamalarla zenginleştirilmeye çalışılacaktır. Yani konuları anlatırken ufak program örnekleri üzerinden gitmeye çalışacağız...

## 5.1 Tkinter Programlarının Renk Şemasını Değiştirmek

Tkinter programlarını güzelleştirme yolunda ilk inceleyeceğimiz konu, yazdığımız uygulamaların renk şemasını değiştirmek olacaktır... Biz herhangi bir Tkinter uygulaması yazdığımız zaman, bu uygulamanın varsayılan bir renk şeması olacaktır. Hemen bir örnekle başlayalım:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Örnek Uygulama")
etiket.pack()

mainloop()
```

Buna benzer bütün uygulamalarımızda belli bir renk şeması varsayılan olarak pencerelerimize uygulanacaktır. Eğer biz müdahale etmez isek, oluşturacağımız bütün pencereler ve üzerindeki pencere araçları tek bir renkte görünecektir. Ama eğer istersek bu gidişe bir dur diyebiliriz. Bunu nasıl yapabileceğimizi daha önceki derslerimizde de kısmen görümüştük. Mesela şu seçenekler bize yardımcı oluyordu:

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

```
from Tkinter import *

pencere = Tk()

etiket = Label(text="Örnek Uygulama",
               fg="white",
               bg="blue")

etiket.pack()

mainloop()
```

Burada, kullandığımız etiket (Label) adlı pencere aracını, “bg” ve “fg” seçenekleri yardımıyla farklı renklerde çizdik. Siz renk adları yerine, http://www.html-color-codes.info/ adresine başvurarak, renk kodlarını da kullanabileceğinizi biliyorsunuz...

Aslında benim kastettiğim “güzelleştirme” operasyonu sadece pencere araçlarının rengini değiştirmekle ilgili değil. Ben aynı zamanda, penceremizin rengini değiştirmekten de bahsediyorum. Lafla peynir gemisi yürümüyor, o yüzden isterseniz hemen bir örnek vererek ne demek istediğimizi biraz daha net olarak anlatmaya çalışayım. Şu örneğe bir bakın:

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

from Tkinter import *

pencere = Tk()

pencere.tk_setPalette("#D0A9F5")

etiket = Label(text="Renk şeması değiştirilmiş bir örnek uygulama")
etiket.pack()

mainloop()
```

Gördüğünüz gibi, sadece tek bir pencere aracının rengini değil, aynı zamanda bütün bir pencerenin rengini değiştirdik. Bunu yapmamızı sağlayan şey de kodlar içinde kullandığımız, “pencere.tk_setPalette(“#D0A9F5”)” satırı oldu. Yani buradaki “tk_setPalette” adlı metodu kullanarak bütün pencerenin renk şemasını değiştirebildik. Bu “tk_setPalette()” komutu, Tk() sınıfının metotlarından yalnızca bir tanesidir. Bu sınıfın bize hangi metotları sunduğunu görmek için Python’un etkileşimli kabuğunda şu komutları verebilirsiniz:

```
from Tkinter import *
dir(Tk())
```

Bu komutu verdiğimizde şu devasa listeyi elde ediyoruz:

```
['_Misc__winfo_getint', '_Misc__winfo_parseitem', '__contains__',
'__doc__', '__getattr__', '__getitem__', '__init__', '__module__',
'__setitem__', '__str__', '_bind', '_configure', '_displayof',
'_getboolean', '_getdoubles', '_getints', '_grid_configure',
'_loadtk', '_nametowidget', '_noarg_', '_options', '_register',
'_report_exception', '_root', '_subst_format', '_subst_format_str',
'_substitute', '_tclCommands', '_tkloaded', '_w', 'adderrorinfo',
'after', 'after_cancel', 'after_idle', 'aspect', 'attributes',
'bbox', 'bell', 'bind', 'bind_all', 'bind_class', 'bindtags',
'call', 'cget', 'children', 'client', 'clipboard_append',
'clipboard_clear', 'clipboard_get', 'colormapwindows',
```

```
'colormodel', 'columnconfigure', 'command', 'config', 'configure',
'createcommand', 'createfilehandler', 'createtimerhandler',
'deiconify', 'deletecommand', 'deletefilehandler', 'destroy',
'dooneevent', 'eval', 'evalfile', 'event_add', 'event_delete',
'event_generate', 'event_info', 'exprboolean', 'exprdouble',
'exprlong', 'exprstring', 'focus', 'focus_displayof', 'focus_force',
'focus_get', 'focus_lastfor', 'focus_set', 'focusmodel', 'frame',
'geometry', 'getboolean', 'getdouble', 'getint', 'getvar',
'globalcall', 'globaleval', 'globalgetvar', 'globalsetvar',
'globalunsetvar', 'grab_current', 'grab_release', 'grab_set',
'grab_set_global', 'grab_status', 'grid', 'grid_bbox',
'grid_columnconfigure', 'grid_location', 'grid_propagate',
'grid_rowconfigure', 'grid_size', 'grid_slaves', 'group',
'iconbitmap', 'iconify', 'iconmask', 'iconname', 'iconposition',
'iconwindow', 'image_names', 'image_types', 'interpaddr',
'keys', 'lift', 'loadtk', 'lower', 'mainloop', 'master',
'maxsize', 'merge', 'minsize', 'nametowidget', 'option_add',
'option_clear', 'option_get', 'option_readfile', 'overrideredirect',
'pack_propagate', 'pack_slaves', 'place_slaves', 'positionfrom',
'propagate', 'protocol', 'quit', 'readprofile', 'record',
'register', 'report_callback_exception', 'resizable', 'rowconfigure',
'selection_clear', 'selection_get', 'selection_handle',
'selection_own', 'selection_own_get', 'send', 'setvar', 'size',
'sizefrom', 'slaves', 'split', 'splitlist', 'state', 'title',
'tk', 'tk_bisque', 'tk_focusFollowsMouse', 'tk_focusNext',
'tk_focusPrev', 'tk_menuBar', 'tk_setPalette', 'tk_strictMotif',
'tkraise', 'transient', 'unbind', 'unbind_all', 'unbind_class',
'unsetvar', 'update', 'update_idletasks', 'wait_variable',
'wait_visibility', 'wait_window', 'waitvar', 'wantobjects',
'willdispatch', 'winfo_atom', 'winfo_atomname', 'winfo_cells',
'winfo_children', 'winfo_class', 'winfo_colormapfull',
'winfo_containing', 'winfo_depth', 'winfo_exists', 'winfo_fpixels',
'winfo_geometry', 'winfo_height', 'winfo_id', 'winfo_interps',
'winfo_ismapped', 'winfo_manager', 'winfo_name', 'winfo_parent',
'winfo_pathname', 'winfo_pixels', 'winfo_pointerx',
'winfo_pointerxy', 'winfo_pointery', 'winfo_reqheight',
'winfo_reqwidth', 'winfo_rgb', 'winfo_rootx', 'winfo_rooty',
'winfo_screen', 'winfo_screencells', 'winfo_screendepth',
'winfo_screenheight', 'winfo_screenmmheight', 'winfo_screenmmwidth',
'winfo_screenvisual', 'winfo_screenwidth', 'winfo_server',
'winfo_toplevel', 'winfo_viewable', 'winfo_visual', 'winfo_visualid',
'winfo_visualsavailable', 'winfo_vrootheight', 'winfo_vrootwidth',
'winfo_vrootx', 'winfo_vrooty', 'winfo_width', 'winfo_x', 'winfo_y',
'withdraw', 'wm_aspect', 'wm_attributes', 'wm_client',
'wm_colormapwindows', 'wm_command', 'wm_deiconify', 'wm_focusmodel',
'wm_frame', 'wm_geometry', 'wm_grid', 'wm_group', 'wm_iconbitmap',
'wm_iconify', 'wm_iconmask', 'wm_iconname', 'wm_iconposition',
'wm_iconwindow', 'wm_maxsize', 'wm_minsize', 'wm_overrideredirect',
'wm_positionfrom', 'wm_protocol', 'wm_resizable', 'wm_sizefrom',
'wm_state','wm_title', 'wm_transient', 'wm_withdraw']
```

Bu liste alfabe sırasına göre dizilmiştir. Dolayısıyla bu listedeki “tk_setPalette” metodunu rahatlıkla bulabilirsiniz...

Gördüğünüz gibi, Tk() sınıfının bir yığın metodu var... İşte tk_setPalette() metodu da bunlardan biri olup, Tkinter pencerelerinin renk şemasını değiştirmemizi sağlıyor...

Şimdi şu örneğe bakalım:

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

from Tkinter import *

pencere = Tk()

pencere.tk_setPalette("#D0A9F5")

etiket = Label(text="Hangi GNU/Linux Dağıtımını Kullanıyorsunuz?")
etiket.pack(pady=10,padx=10)

liste = Listbox()
liste.pack(side=LEFT)

GNULinux = ["Debian","Ubuntu","Kubuntu","RedHat"]

for i in GNULinux:
    liste.insert(END,i)

dugme = Button(text="KAPAT",command=pencere.quit)
dugme.pack()

mainloop()
```

Gördüğünüz gibi, tk_setPalette() metodu ile pencerenin renk şemasını değiştirdiğimizde, bundan pencere ile birlikte üzerindeki bütün pencere araçları da etkileniyor. Peki biz pencere araçlarının pencereden farklı bir renkte olmasını istersek ne yapacağız? Çok basit:

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

from Tkinter import *

pencere = Tk()

pencere.tk_setPalette("#D0A9F5")

etiket = Label(text="Hangi GNU/Linux Dağıtımını Kullanıyorsunuz?",
               fg="#B4045F")

etiket.pack(pady=10, padx=10)

liste = Listbox()
liste.pack(side=LEFT)

GNULinux = ["Debian","Ubuntu","Kubuntu","RedHat"]

for i in GNULinux:
    liste.insert(END,i)

dugme = Button(text="KAPAT",
               bg="#610B5E",
               fg="white",
               command=pencere.quit)

dugme.pack()
```

```
mainloop()
```

Gördüğünüz gibi, farklı renkte olmasını istediğimiz pencere araçlarının renklerini, daha önce öğrendiğimiz seçenekler yardımıyla açık açık belirtiyoruz... Tabii sizin benden daha zevkli renk kombinasyonları seçeceğinizi tahmin ediyorum...

Böylelikle bir Tkinter programının renk şemasını nasıl değiştireceğimizi öğrenmiş olduk... Bir sonraki bölümde Tkinter uygulamaları üzerinde resimleri/simgeleri nasıl kullanabileceğimizi göreceğiz. Neticede resimler ve simgeler de bir programı güzelleştiren öğelerdir...

## 5.2 Pencere Araçlarına Simge Eklemek

Bu bölümde, Tkinter'de yazdığımız programlara nasıl simge ekleyebileceğimiz göreceğiz. Bu iş için ImageTk adlı modülü kullanacağız. Bu modülü kullanmak için "python-imaging" adlı paketi yüklemeniz gerekecektir. GNU/Linux kullanıcıları kendi dağıtımlarının paket yöneticisini kullanarak gerekli modülü kurabilirler. Windows kullanıcıları ise http://www.pythonware.com/products/pil/ adresini ziyaret ederek, kullandıkları Python sürümüne uygun programı indirebilir...

Gelelim asıl konumuza... Öncelikle aşağıdaki gibi bir başlangıç yapalım:

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

from Tkinter import *
import ImageTk
```

Böylelikle gerekli bütün modülleri içe aktarmış olduk. Burada "ImageTk" adlı modülü de içe aktardığımıza dikkat edin. Devam ediyoruz...

```
pencere = Tk()

simge = ImageTk.PhotoImage(file="simge.png")
```

Yukarıdaki kodlar yardımıyla önce bir pencere oluşturduk her zamanki gibi. Ardından da, programın en başında içe aktardığımız ImageTk adlı modülün PhotoImage adlı metodundan ve file adlı seçeneğinden yararlanarak, bilgisayarımızdaki "simge.png" adlı dosyayı programımız içinde kullanılabilir hale getirdik. Dikkat edin, "kullandık," demiyorum. "Kullanılabilir hale getirdik," diyorum... Yani kabaca ifade etmek gerekirse, bu şekilde bir "resim nesnesi" oluşturmuş oluyoruz. Devam edelim:

```
dugme = Button(text="Resim1",
               image=simge,
               compound="top")
dugme.pack()

mainloop()
```

Burada bildiğimiz şekilde, "Button" pencere aracını kullanarak bir "düğme" oluşturduk. Ama burada bilmediğimiz birkaç şey var. Hemen anlatalım: Öncelikle "image" adlı yeni bir seçenek görüyoruz kodlarımız arasında. Buraya, yukarıda "kullanılabilir hale getirdiğimiz" resim nesnesini yerleştiriyoruz. Bu seçeneğin ardından, "compound" adlı başka bir seçenek daha geliyor. Bu da "text" ve "image" seçenekleri ile tanımlanan öğelerin pencerede nasıl görüneceğini belirliyor. Burada kullandığımız "top" değeri, image seçeneğiyle gösterilen öğenin, text seçeneği ile gösterilen öğenin üstünde yer alacağını ifade ediyor. Bu "compound" seçeneği dört farklı değer alabilir:

1. top => resim metnin üstünde
2. bottom => resim metnin altında
3. right => resim metnin sağında
4. left => resim metnin solunda

Biz örneğimizde “top” değerini tercih ettik. Tabii ki siz bu dört seçenek içinden istediğinizi kullanabilirsiniz... Yukarıdaki kodlarda görünen “pack()” ve “mainloop” artık adımız soyadımız gibi bildiğimiz şeyler. O yüzden üzerinde durmuyoruz.

Şimdi isterseniz bu kodları topluca görelim:

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

from Tkinter import *
import ImageTk

dugme = Button(text="Resim1",
               image=simge,
               compound="top")
dugme.pack()

mainloop()
```

Böylece Tkinter’de resimleri ve metinleri bir arada nasıl kullanacağımızı öğrenmiş olduk. Şimdi gelin isterseniz yukarıdaki bilgilerimizi kullanarak örnek bir arayüz yazalım:

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

from Tkinter import *
import ImageTk

pencere = Tk()
pencere.tk_setPalette("#CECEF6")

cerceve1 = Frame()
cerceve1.grid(row=0, column=0, pady=5, padx=5)

cerceve2 = Frame()
cerceve2.grid(row=1, column=0, pady=10, padx=10)

simge1 = ImageTk.PhotoImage(file="simge1.png")
simge2 = ImageTk.PhotoImage(file="simge2.png")
simge3 = ImageTk.PhotoImage(file="simge3.png")
simge4 = ImageTk.PhotoImage(file="simge4.png")

etiket = Label(cerceve1,
               text="Kendinize Uygun bir Simge Seçiniz...",
               fg="#610B0B",
               font="Verdana 10 bold")

etiket.pack()

dugme1 = Button(cerceve2,
                text="Resim1",
                image=simge1,
                compound="top")
```

```
dugme1.grid(row=3, column=0, padx=6)

dugme2 = Button(cerceve2,
                text="Resim2",
                image=simge2,
                compound="top")

dugme2.grid(row=3, column=3, padx=6)

dugme3 = Button(cerceve2,
                text="Resim3",
                image=simge3,
                compound="top")

dugme3.grid(row=3, column=6, padx=6)

dugme4 = Button(cerceve2,
                text="Resim4",
                image=simge4,
                compound="top")

dugme4.grid(row=3, column=9, padx=6)
```

Bu programı çalıştırdığımızda şöyle bir görüntü elde ediyoruz:


Tabii bu programın düzgün çalışması için resimlerinizle program dosyasını aynı klasöre koymayı unutmuyorsunuz... Eğer resimlerle program dosyası ayrı klasörlerde duracaksa simge öğelerini belirlerken resimlerin bulunduğu klasör adlarını tam olarak yazmaya dikkat etmelisiniz...

Burada düğmeleri konumlandırmak için “pack()” geometri yöneticisi yerine “grid()” adlı geometri yöneticisini kullandığımıza dikkat edin. Öğeleri bu şekilde sıralamak için grid() yöneticisi daha uygundur. Ayrıca yukarıdaki kodlarda iki farklı “Frame” oluşturduğumuza da dikkat edin. “Label” adlı pencere aracımızı bir “Frame” üzerine, geri kalan pencere araçlarımızı (yani düğmelerimizi) ise öteki “Frame” üzerine yerleştirdik. İki farklı “Frame” kullanmamız sayesinde aynı program içinde hem “grid()” hem de “pack()” adlı geometri yöneticilerini bir arada kullanma imkanı elde ettik. İki farklı “Frame” kullanmamız aynı zamanda bize, pencere araçlarımızı daha özgür bir şekilde konumlandırma imkanı da sağladı. Ayrıca grid() yöneticisinin “padx” seçeneğini kullanarak pencere araçlarının birbirlerine olan uzaklıklarını da ayarlayabildik. Yukarıdaki kodlarla ilgili olarak anlamadığınız noktalar olması durumunda bana nasıl ulaşabileceğinizi biliyorsunuz...

Böylelikle bu konuyu da bitirmiş olduk. Bir sonraki bölümde Tkinter ile yazdığımız pencere araçlarına nasıl ipucu metni (tooltip) ekleyeceğimizi göreceğiz.

## 5.3 Pencere Araçlarına İpucu Metni (Tooltip) Eklemek

Fareyi pencere araçlarının üzerine getirdiğimizde ekranda beliren ve o aracın ne işe yaradığını kısaca açıklayan ufak metin parçalarına ipucu metni (tooltip) diyoruz. İşte bu bölümde Tkinter'deki pencere araçlarına nasıl ipucu metni ekleyeceğimizi öğreneceğiz. Öncelikle şunu söyleyelim: Tkinter böyle bir özelliği bize kendiliğinden sunmuyor. Ama hem Python'un hem de Tkinter'in en güçlü yanlarından biri, dilde olmayan bir özelliği üçüncü parti yazılımlar aracılığıyla dile ekleyebilmemizdir... Pencere araçlarına ipucu metni ekleyebilmek için bu üçüncü parti yazılımlardan birini kullanacağız. Şu noktada yapmamız gereken ilk iş, http://svn.effbot.org/public/stuff/sandbox/wcklib/ adresinden wckToolTips adlı modülü bilgisayarımıza indirmek olacaktır. Bu modül pencere araçlarına ipucu metni eklememizi sağlayacak. Bu modülü, içinde ipucu metni kullanacağımız programlarımızla aynı klasör içinde tutmamız gerekiyor. Örneğin, diyelim ki falanca.py adında bir program yazdık. Bu programın "FALANCA" adlı klasör içinde yer aldığını varsayarsak, wckToolTips modülünü falanca.py ile aynı klasör içine, yani "FALANCA" adlı klasöre atmamız gerekiyor. Bu sayede, wckToolTips modülünü programımız içine aktarırken (import) herhangi bir sorun yaşamayacağız. İsterseniz bununla ilgili ufak bir örnek yapalım:

```
from Tkinter import *
import wckToolTips
```

Böylece Tkinter modülüyle birlikte wckToolTips modülünü de içe aktarmış olduk. Python, biz bu modülü içe aktarmaya çalıştığımızda, öncelikle programımızın içinde bulunduğu dizini kontrol ederek, wckToolTips.py adlı bir dosya olup olmadığına bakacaktır. Eğer modül bu dizinde bulunamaz ise, Python sys.path çıktısında görünen dizinlerin içini de denetleyecektir. Eğer modül orada da yoksa, Python bize bir hata mesajı gösterecektir. Bu durumda, wckToolTips gibi harici bir modülü programın bulunduğu dizin içine atmak en kolay yol olacaktır. Neyse... Biz örneğimize devam edelim:

```
pencere = Tk()

etiket = Label(text="Gölgelerin gücü adına!")
etiket.pack()

dugme = Button(text="Tamam", command=pencere.destroy)
dugme.pack()
```

Burada normal bir şekilde, penceremizi tanımladık. Bunun yanında, bir adet etiket, bir adet de düğme oluşturduk. Yeni bir şey yok... Devam edelim:

```
wckToolTips.register(dugme, "Titrek'i Atılgan'a çevirir!")

mainloop()
```

İşte wckToolTips modülü burada devreye giriyor. Bu modülün "register" adlı metodunu kullanarak, "dugme" adlı pencere aracının üzerine fare ile geldiğimizde ekranda görünmesini istediğimiz ipucu metnini oluşturuyoruz. Burada ipucu metnimiz şu: "Titrek'i Atılgan'a çevirir!" (Hey gidi günler!)

Programımızı bütünüyle bir görelim bakalım:

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

from Tkinter import *
import wckToolTips
```

```
pencere = Tk()

etiket = Label(text="Gölgelerin gücü adına!")
etiket.pack()

dugme = Button(text="Tamam", command=pencere.destroy)
dugme.pack()

wckToolTips.register(dugme, "Titrek'i Atılgan'a çevirir!")

mainloop()
```

Gördüğünüz gibi, duğmenin üzerine fare ile geldiğimizde ipucu metnimiz ekranda görünüyor... İşte Tkinter'de ipucu metinlerini kullanmak bu kadar kolaydır. Kendi yazdığınız programlarda bu modülü kullanarak, programınızı kullanıcılar açısından daha cazip ve anlaşılır bir hale getirebilirsiniz. Gelin isterseniz bununla ilgili bir örnek daha yapalım:

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

from Tkinter import *
import wckToolTips
import time

pencere = Tk()

etiket = Label(text="Saat 17:00'da randevunuz var!")
etiket.pack(expand=YES, fill=BOTH)

def saat(pencere_araci, fonksiyon):
    return "Şu anda saat: %s"%time.strftime("%H:%M:%S")

dugme = Button(text="Tamam", command=pencere.destroy)
dugme.pack(expand=YES)

wckToolTips.register(pencere, saat)

mainloop()
```

Burada ipucu metnimizi doğrudan pencerenin kendisine atadık. Böylece fare ile pencere üzerine geldiğimizde o an saatin kaç olduğu ekranda görünecektir... Ayrıca burada register metoduna argüman olarak bir fonksiyon atadığımıza dikkat edin. Yukarıda yazdığımız kodları şöyle yazarsak ne olur peki?

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

from Tkinter import *
import wckToolTips
import time

pencere = Tk()

etiket = Label(text="Saat 17:00'da randevunuz var!")
etiket.pack(expand=YES, fill=BOTH)

dugme = Button(text="Tamam", command=pencere.destroy)
dugme.pack(expand=YES)

wckToolTips.register(pencere, "Şu anda saat: %s"%time.strftime("%H:%M:%S"))
```

```
mainloop()
```

Bu şekilde, programımız yine çalışır ve hatta saatin kaç olduğunu ekranda ipucu metni olarak da gösterir. Ama bir farkla: Bu yazım şeklinde ekranda görünen saat durağan olacaktır. Yani program kapatılıp tekrar açılana kadar hep aynı saati gösterecektir. Ama register metoduna bir önceki örnekte olduğu gibi, bir fonksiyon atarsak saat devinecek, yani pencere üzerine ikinci kez geldiğimizde ipucu metni saati güncellenmiş olarak verecektir.

Bu son kodlarda, ayrıca pack() geometri yöneticisinin “expand” ve “fill” seçeneklerini kullanarak pencere boyutlandırmaları sırasında pencere araçlarının uzayıp kısalmasını sağladık. Bu kodları bir de bu seçenekler olmadan çalıştırırsanız ne demek istediğim daha bariz bir şekilde ortaya çıkacaktır.

Son olarak, yukarıdaki kodlarda, “saat” adlı fonksiyonun iki argüman aldığına dikkat edin. Bu fonksiyonu argümansız olarak yazarsak programımız çalışsa da ipucu metni görüntülenmeyecektir. Böyle bir durumda ipucu metninin görüntülenmemesine yol açan hatayı komut satırı çıktılarından takip edebilirsiniz.

BÖLÜM 6

# Nasıl Yapılır?

## 6.1 Tkinter'de Fare ve Klavye Hareketleri (Events and Bindings)

Tkinter'de yazdığımız bir uygulamada, kullanıcının fare hareketlerini ve klavyede bastığı tuşları belli fonksiyonlara bağlamak ve bu hareketleri yakalamak bu bölümün konusunu oluşturuyor. Mesela kullanıcı klavyedeki "enter" tuşuna bastığında programımızın özel bir işlemi yerine getirmesini sağlamak, kullanıcının hangi tuşa bastığını bulmak bu konunun kapsamına giriyor. Her zamanki gibi lafı hiç uzatmadan, bir örnekle derdimizi anlatmaya çalışalım. Diyelim ki şöyle bir şey var elimizde:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa adınızı yazıp Tamam düğmesine basın!\n")
etiket.pack()

giris = Entry()
giris.pack()

def kaydet():
    dosya = open("isimler_python.txt","a")
    veri = giris.get()
    veri = unicode.encode(unicode(veri),"utf8")
    dosya.write(veri+"\n")
    dosya.close()
    etiket["text"] = etiket["text"] + u"%s dosyaya başarıyla eklendi\n"%giris.get()
    giris.delete(0,END)

dugme = Button(text="Tamam",command=kaydet)
dugme.pack()

mainloop()
```

Burada kullanıcı "Tamam" düğmesine bastığında "kaydet()" fonksiyonu içinde tanımlanan

işlemler gerçekleştirilecektir. Bu arada, yukarıdaki kodlarda Türkçe karakterleri alabilmek için nasıl bir yol izlediğimize dikkat edin. Dediğimiz gibi, bu programın çalışması için kullanıcının "Tamam" düğmesine basması gerekiyor. Peki biz aynı işlemin "enter" düğmesine basılarak da yapılabilmesini istersek ne olacak? Yani kullanıcının mutlaka "Tamam" düğmesine basmasını zorunlu kılmadan, klavyedeki "enter" tuşuna bastığında da kutucuğa yazılan isimlerin dosyaya eklenmesini nasıl sağlayacağız? İşte bunu yapabilmek için "bind" fonksiyonundan yararlanmamız gerekiyor. Bu fonksiyon, bize klavye ve fare hareketlerini belli fonksiyonlara bağlama ve bu hareketleri yakalama imkanı verecek. Nasıl mı? Hemen görelim:

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

from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa adınızı yazıp Tamam düğmesine basın!\n")
etiket.pack()

giris = Entry()
giris.pack()

def kaydet(event=None):
    dosya = open("isimler_python.txt","a")
    veri = giris.get()
    veri = unicode.encode(unicode(veri),"utf8")
    dosya.write(veri+"\n")
    dosya.close()
    etiket["text"] = etiket["text"] + u"%s dosyaya başarıyla eklendi\n"%giris.get()
    giris.delete(0,END)

dugme = Button(text="Tamam",command=kaydet)
dugme.pack()

pencere.bind("<Return>",kaydet)

mainloop()
```

Burada, önceki kodlarımıza yalnızca şu eklemeleri yaptık:

```
def kaydet(event=None):
    ....

....
pencere.bind("<Return>",kaydet)
...
```

"kaydet()" fonksiyonuna eklediğimiz "event" parametresi zorunlu bir adlandırma değildir. Hatta oraya "kezban" dahi yazdığınızda programın düzgün çalıştığını göreceksiniz. Ancak oraya bir parametre yazacağınız zaman herhangi bir kelime yerine "event" ifadesini kullanmanızı tavsiye ederim. Çünkü "event" tıpkı sınıflardaki "self" gibi, kemikleşmiştir... Burada dikkat etmemiz gereken iki şey var: Birincisi, o fonksiyonu parametresiz bırakmamak; ikincisi, parametreye varsayılan değer olarak "None" vermek. Eğer bu "None" değerini vermezseniz, kullanıcı "Enter" tuşuna bastığında program çalışır, ama "Tamam" düğmesine bastığında çalışmaz... Çünkü Tkinter "event" parametresinin değerini "command=kaydet" şeklinde gösterdiğimiz koda otomatik olarak atamayacaktır. Bu yüzden, oraya varsayılan değer olarak "None" yazmazsak, "kaydet()" fonksiyonunun bir adet argüman alması gerektiğine" dair bir hata alırız...

Kodlarımıza yaptığımız ikinci ekleme, bind() fonksiyonudur. Burada dikkat etmemiz gereken birkaç şey var. Öncelikle bu "bind()" fonksiyonunu "pencere" ile birlikte kullandığımıza dikkat edin. Bu satırla aslında Tkinter'e şu komutu vermiş oluyoruz:

> Ey Tkinter! "pencere" seçili iken, yani etkinken, kullanıcı "Return" düğmesine (veya başka bir söyleyişle "Enter" düğmesine) bastığında "kaydet" adlı fonksiyonu çalıştır!

Artık kullanıcımız, kutucuğa bir isim yazıp "enter" düğmesine bastığında, yazdığı isim dosyaya eklenecektir.

Gördüğünüz gibi, eğer klavyedeki "enter" düğmesini bağlamak istiyorsak, kullanmamız gereken ifade "<Return>"... Peki başka hangi ifadeler var? Mesela "Escape" için bir örnek verelim. Yukarıdaki kodlara ufacık bir ekleme yapıyoruz:

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

from Tkinter import *
import sys

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa adınızı yazıp Tamam düğmesine basın!\n")
etiket.pack()

giris = Entry()
giris.pack()

def kaydet(event=None):
    dosya = open("isimler_python.txt","a")
    veri = giris.get()
    veri = unicode.encode(unicode(veri),"utf8")
    dosya.write(veri+"\n")
    dosya.close()
    etiket["text"] = etiket["text"] + u"%s dosyaya başarıyla eklendi\n"%giris.get()
    giris.delete(0,END)

dugme = Button(text="Tamam", command=kaydet)
dugme.pack()

pencere.bind("<Return>", kaydet)
pencere.bind("<Escape>", sys.exit)

mainloop()
```

Gördüğünüz gibi, klavyedeki "Escape" düğmesini kullanabilmek için kodlarımız içine "<Escape>" ifadesini eklememiz yeterli oluyor. Peki klavyedeki öbür düğmelerin karşılıkları nelerdir? Listeleyelim:

**F1:** "<F1>" (Bütün F'li tuşlar aynı şekilde kullanılabilir...)

**Num Lock:** "<Num_Lock>"

**Scroll Lock:** "<Scroll_Lock>"

**Backspace:** "<BackSpace>"

**Delete:** "<Delete>"

**Sol ok:** "<Left>"

**Sağ ok:** "<Right>"

**Aşağı ok:** “<Down>”

**Yukarı ok:** “<Up>”

**“Alt” Düğmesi:** “<Alt_L>”

**“Ctrl” Düğmesi:** “<Control_L>”

**“Shift” Düğmesi:** “<Shift_L>”

**“Ctrl” + s:** “<Control-s>” (Öteki harfler de aynı şekilde kullanılabilir...)

**“Shift” + s:** “<Shift-s>” (Öteki harfler de aynı şekilde kullanılabilir...)

**“Alt” + s:** “<Alt-s>” (Öteki harfler de aynı şekilde kullanılabilir...)

**Boşluk düğmesi:** “<space>”

**“Print” düğmesi:** “<Print>”

Klavyedeki harfleri pencere araçlarına bağlamak için ise doğrudan o harfin adını kullanabiliriz. Mesela bir işlemin, “f” harfine basıldığında gerçekleşmesini istersek, yazmamız gereken ifade, “f” olacaktır. Bunun, “<f>” değil, sadece “f” şeklinde olduğuna dikkat edin. Tabii burada, Türkçe’ye özgü harfleri kullanmamaya dikkat ediyoruz...

Klavye hareketlerini bağlamayı öğrendik. Peki ya fare hareketlerini nasıl bağlayacağız? O da çok basittir. Burada da yine bind() fonksiyonundan yararlanacağız. Gelin isterseniz şöyle bir uygulama yazalım:

```
# -*- coding: utf-8 -*-
from Tkinter import *

pencere = Tk()
pencere.geometry("400x200")

etiket = Label(text="Farenin pencere üzerindeki konumu:")
etiket.pack(anchor=NW)

girisx_etiket = Label(text="X:")
girisx_etiket.pack(side=LEFT, anchor=N)

girisx = Entry(width=5)
girisx.pack(side=LEFT,padx=15, anchor=N)

girisy_etiket = Label(text="Y:")
girisy_etiket.pack(side=LEFT, anchor=N)

girisy = Entry(width=5)
girisy.pack(side=LEFT, anchor=N)

def hareket(event=None):
    girisx.delete(0, END)
    girisy.delete(0, END)
    girisx.insert(0, event.x)
    girisy.insert(0, event.y)

pencere.bind("<Button-1>", hareket)

mainloop()
```

Yukarıdaki uygulamayı çalıştırıp pencere üzerinde herhangi bir yere farenin sol tuşu ile tıkladığımızda, tıkladığımız noktanın koordinatları (x ve y düzlemine göre) kutucuklar içinde

görünecektir.

Bu kodları biraz açıklayalım:

Kodlar arasında gördüğünüz “anchor” ifadesi, pack() adlı pencere yöneticisinin seçeneklerinden biridir. Bu seçenek, ilgili pencere aracının pencere üzerinde bir noktaya “çapa atmasını” sağlar (İngilizce “anchor” kelimesi Türkçe’de “çapa atmak, demir atmak” gibi anlamlara gelir...). Bu seçenek dört farklı değer alabilir:

- **N** : “Kuzey” anlamına gelen “North” kelimesinin kısaltmasıdır. Pencere aracının “kuzey” yönüne sabitlenmesini sağlar.
- **S** : “Güney” anlamına gelen “South” kelimesinin kısaltmasıdır. Pencere aracının “güney” yönüne sabitlenmesini sağlar.
- **W** : “Batı” anlamına gelen “West” kelimesinin kısaltmasıdır. Pencere aracının “batı” yönüne sabitlenmesini sağlar.
- **E** : “Doğu” anlamına gelen “East” kelimesinin kısaltmasıdır. Pencere aracının “doğu” yönüne sabitlenmesini sağlar.

Bu yönleri birarada da kullanabiliriz. Örneğin pencere aracının “kuzeybatı” yönünde sabitlenmesini istersek, kullanmamız gereken ifade “NW” olacaktır. Mesela yukarıdaki kodlarda bunu kullandık. “etiket.pack(anchor=NW)” ifadesi yardımıyla, etiket adlı pencere aracımızın kuzeybatı yönüne çapa atmasını sağladık (Yani pencerenin üst-sol ucuna...). Öteki pencere araçlarında sadece “anchor=N” ifadesini kullandık. Çünkü bu araçlardaki “side=LEFT” ifadesi, aracımızın sol yana yerleşmesini zaten sağlıyor... pack() yöneticisi içinde kullandığımız “padx” seçeneğini zaten biliyorsunuz. Bu seçenek yardımıyla pencere araçlarının “dirsek mesafesini” ayarlıyoruz...

Yukarıdaki kodlar içinde en önemli nokta tabii ki hareket() adlı fonksiyon. Burada parantez içinde “event” parametresini belirtmeyi unutmuyoruz. Kabaca ifade etmek gerekirse, bu parametre bizim örneğimizde fare hareketlerini temsil ediyor... Buna göre, fonksiyon içindeki “event.x” ve “event.y” ifadelerini, “farenin x düzlemi üzerindeki hareketi” ve “farenin y düzlemi üzerindeki hareketi” şeklinde düşünebiliriz... Bu arada, “giris.delete()” fonksiyonları yardımıyla, Entry aracının içini sürekli olarak boşalttığımıza dikkat edin. Eğer giris.delete() fonksiyonlarını yazmazsak, fare tıklaması ile bulduğumuz eski ve yeni koordinatlar birbirine karışacaktır...

mainloop() satırından hemen önce, programımızdaki asıl işi yapan bind() fonksiyonunu görüyoruz. Farenin sol düğmesini “Button-1” ifadesinin temsil ettiğini görüyorsunuz. Peki buna benzer başka neler var? Görelim:

**Button-2 :** Farenin orta düğmesine (veya tekerine) basılması

**Button-3 :** Farenin sağ düğmesine basılması

**Motion :** Farenin, hiç bir düğme basılı değilken hareket ettirilmesi

**B1-Motion :** Farenin, sol düğme basılı halde hareket ettirilmesi

**B2-Motion :** Farenin, orta düğme (veya teker) basılı halde hareket ettirilmesi

**B3-Motion :** Farenin, sağ düğme basılı halde hareket ettirilmesi

**ButtonRelease-1 :** Farenin sol düğmesinin serbest bırakılması (yani düğmeye basıldığındaki değil, düğmenin bırakıldığındaki hali...)

**ButtonRelease-2 :** Farenin orta düğmesinin (veya tekerinin) serbest bırakılması

**ButtonRelease-3 :** Farenin sağ düğmesinin serbest bırakılması

**Double-Button-1 :** Farenin sol düğmesine çift tıklanması

**Double-Button-2 :** Farenin orta düğmesine (veya tekerine) çift tıklanması

**Double-Button-3 :** Farenin sağ düğmesine çift tıklanması

**Enter :** Farenin pencere aracının üzerine gelmesi (Bunun "enter" tuşuna basmak anlamına gelmediğine dikkat edin...)

**Leave :** Farenin pencere aracını terketmesi

Sırada klavye hareketlerini yakalamak var. Yani şimdiki görevimiz, bir kullanıcının klavyede hangi tuşlara bastığını bulmak. Bunun için "keysym" metodundan yararlanacağız:

```
# -*- coding: utf-8 -*-
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

etiket = Label(text="Basılan Tuş:\n",wraplength=150)
etiket.pack()

def goster(event=None):
    etiket["text"] += event.keysym

pencere.bind("<Key>",goster)

mainloop()
```

Yine kodlarımızı biraz açıklayalım:

Öncelikle, gördüğünüz gibi, etiket adlı pencere aracımız içinde "wraplength" adlı bir seçenek kullandık. Bu seçenek etikete yazdırılabilecek metnin uzunluğunu sınırlandırıyor. Biz bu değeri 150 piksel olarak belirledik. Buna göre, etikete yazılan metin 150 pikseli aştığında kendiliğinden bir alt satıra geçecektir.

Şimdi fonksiyonumuza bir göz atalım: Burada "event.keysym" adlı bir ifade görüyoruz. Bu ifade, klavye üzerindeki sembolleri, yani tuşları temsil ediyor. Fonksiyonumuz içinde yazdığımız koda göre, kullanıcının girdiği klavye sembolleri ana penceremiz üzerindeki etikete yazdırılacak. Bir sonraki satırda yine bind() fonksiyonunu kullanarak, goster() adlı fonksiyon ile "<Key>" adlı özel ifadeyi birbirine bağlıyoruz. Bu "<Key>" ifadesi, kullanıcının klavye üzerindeki tuşlara basma hareketini temsil ediyor. Kodlarımızı bir bütün halinde düşünürsek, yukarıdaki uygulama, kullanıcının klavyede bastığı bütün tuşları ana pencere üzerindeki etikete atayacaktır. Bu uygulama yardımıyla, esasında "salt okunur" (read-only) bir pencere aracı olan "Label"i "yazılabilir" (writable) hale getirmiş oluyoruz (en azından görünüş olarak...).

Öğrendiğimiz bu "keysym" metodu yardımıyla bazı faydalı işler de yapabiliriz. Mesela bir "Entry" aracına kullanıcı tarafından girilen bir metnin silinmesini engelleyebiliriz. Nasıl mı?

```
# -*- coding: utf-8 -*-
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

giris = Entry()
giris.pack()

def silemezsin(event=None):
    if event.keysym == "BackSpace" or event.keysym == "Delete":
        return "break"
```

```
giris.bind("<Key>", silemezsin)

mainloop()
```

Buradaki kodlara göre, eğer kullanıcı “BackSpace” veya “Delete” tuşlarına basarak yazdığı yazıyı silmek isterse, beklentilerinin aksine bu tuşlar çalışmayacaktır... Burada return “break” adlı özel bir ifadeden yararlanıyoruz. Bu ifade, normal şartlarda gerçekleşmesi engellenemeyecek olan işlemlerin etkisizleştirilmesini sağlayan özel bir kullanımdır... Örneğin yukarıdaki “silemezsin()” fonksiyonunu bir de şöyle yazmayı deneyin:

```
def silemezsin(event=None):
    if event.keysym:
        return "break"
```

Bu şekilde, kullanıcı Entry aracı içine hiçbir şekilde yazı yazamayacaktır. Dolayısıyla yukarıdaki fonksiyonun adını “silemezsin” yerine “yazamazsın” koymak daha uygun olacaktır!...

Son olarak, bu konuyla ilgili olduğu için, “focus_set()” fonksiyonundan da bahsetmekte fayda var. Bu fonksiyon, pencere araçlarını etkin hale getirmemizi sağlıyor. Bu bölümün en başında verdiğimiz örneği hatırlıyorsunuz. Kullanıcı bir kutucuğa adını giriyor ve girdiği bu ad bir dosyaya yazdırılıyordu. O örneği isterseniz yeniden çalıştırın. Göreceğiniz gibi, kutucuğa bir şey yazabilmek için öncelikle o kutucuğa bir kez tıklamak gerekiyor. Yani aslında programımız ilk açıldığında o kutucuk etkin değil. Bizim onu etkin hale getirmek için üzerine bir kez tıklamamız gerekiyor. Ama istersek, o kutucuğun açılışta etkin hale gelmesini sağlayabiliriz. Böylece kullanıcılarımız doğrudan kutuya yazmaya başlayabilirler:

```
# -*- coding: utf-8 -*-
from Tkinter import *

pencere = Tk()

etiket = Label(text="Aşağıdaki kutucuğa adınızı yazıp Tamam düğmesine basın!\n")
etiket.pack()

giris = Entry()
giris.pack()
giris.focus_set()

def kaydet():
    dosya = open("isimler_python.txt","a")
    veri = giris.get()
    veri = unicode.encode(unicode(veri),"utf8")
    dosya.write(veri+"\n")
    dosya.close()
    etiket["text"] = etiket["text"] + u"%s dosyaya başarıyla eklendi\n"%giris.get()
    giris.delete(0,END)

dugme = Button(text="Tamam", command=kaydet)
dugme.pack()

mainloop()
```

Kodlar içinde yaptığımız eklemeyi koyu renkle gösterdim. “giris.focus_set()” fonksiyonu sayesinde, programımız açılır açılmaz kutucuk etkin hale geliyor. Böylece hemen adımızı yazmaya başlayabiliyoruz.

Bu konuyla bağlantılı olan şu örneğe bakalım bir de:

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

from Tkinter import *
from sys import exit

pencere = Tk()

etiket = Label(text="Programdan çıkmak istediğinize emin misiniz?")
etiket.pack()

cerceve = Frame()
cerceve.pack()

def sagol(event=None):
    yeni = Toplevel()
    etiket = Label(yeni, text="Teşekkürler... :)")
    yeni.bind("<Return>", exit)
    etiket.pack()

evet = Button(cerceve, text="Evet", command=pencere.destroy)
evet.grid(row=1,column=0)

hayir = Button(cerceve, text="Hayır", command=sagol)
hayir.grid(row=1,column=1)

evet.bind("<Return>", exit)
hayir.bind("<Return>", sagol)

mainloop()
```

Gördüğünüz gibi, klavyedeki “enter” tuşunu bazı fonksiyonlara bağladığımız halde, programı çalıştırdığımızda “enter” tuşuna basmak hiçbir işe yaramıyor. Bunun nedeni, programımız ilk açıldığında pencerenin kendisi hariç hiç bir pencere aracının etkin olmamasıdır. Bağladığımız tuşların çalışabilmesi için öncelikle ilgili pencere araçlarının etkinleştirilmesi gerekiyor:

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

from Tkinter import *
from sys import exit

pencere = Tk()

etiket = Label(text="Programdan çıkmak istediğinize emin misiniz?")
etiket.pack()

cerceve = Frame()
cerceve.pack()

def sagol(event=None):
    yeni = Toplevel()
    etiket = Label(yeni, text="Teşekkürler... :)")
    yeni.bind("<Return>", exit)
    etiket.pack()

kapat = Button(cerceve, text="Evet", command=pencere.destroy)
kapat.grid(row=1, column=0)

kapatma = Button(cerceve, text="Hayır", command=sagol)
```

```
kapatma.grid(row=1, column=1)

kapat.focus_set()

kapat.bind("<Return>", exit)
kapatma.bind("<Return>", sagol)

mainloop()
```

Buradaki tek fark, kodlar arasına “kapat.focus_set()” ifadesinin eklenmiş olması. Bu ifade sayesinde, programımız ilk açıldığında odak “Evet” düğmesi üzerinde olacaktır. Dolayısıyla “enter” tuşuna bastığımızda, bu düğmenin bağlı olduğu fonksiyon çalışacak ve programımız kapanacaktır. Program açıkken, klavyedeki “tab” tuşuna basarak odağı değiştirebiliriz. Mesela programımızı ilk çalıştırdığımızda odak “Evet” düğmesi üzerinde olacağı için, “tab” tuşuna bir kez bastığımızda odak “Hayır” düğmesine geçecektir. Bu haldeyken “enter” tuşuna basarsak, “Hayır” düğmesinin bağlı olduğu fonksiyon çalışacaktır...

İsterseniz konuyu kapatmadan önce basit bir “oyun” denemesi yapalım:

Önce http://www.akcilaclama.com/images/sinek.ilaclama.jpg adresine gidip oradaki sinek resmini bilgisayarınıza indirin. Adını da “sinek.jpg” koyun... Daha sonra şu kodları yazın:

```
# -*- coding: utf-8 -*-
from Tkinter import *

import random, ImageTk

pencere = Tk()
pencere.geometry("600x600")
pencere.tk_setPalette("white")

def yakala(event=None):
        k = random.randint(0,550)
        v = random.randint(0,550)
        print k, v
        dugme.place(x=k, y=v)

simge = ImageTk.PhotoImage(file="sinek.jpg")
dugme = Button(image=simge, command=yakala, relief="flat")
dugme.place(x=1, y=0)

dugme.bind("<Enter>", yakala)

mainloop()
```

Gördüğünüz gibi, “Enter” ifadesi kullanıcının “enter” tuşuna bastığı anlamına gelmiyor. O iş için “Return” ifadesini kullanıyoruz. Fazlasıyla birbirine karıştırılan bir konu olduğu için özellikle vurgulama gereği duyuyorum...

Burada sineği yakalama ihtimaliniz var. Eğer random’un ürettiği sayılar birbirine yakın olursa sinek elinizden kaçamayabilir...

Böylelikle bu konuyu da bitirmiş olduk. Her ne kadar başlangıçta karışıkmış gibi görünse de aslında hem çok kolay hem de çok keyifli bir konudur fare ve klavye hareketleri konusu... Bu yazıyı birkaç kez gözden geçirerek bazı noktaların tam olarak zihninize yerleşmesini sağlayabilirsiniz.

## 6.2 “Listbox” Öğelerine Görev Atamak

Daha önce “Listbox” (Liste kutusu) adlı pencere aracının ne olduğunu ve nasıl kullanıldığını öğrenmiştik. Tkinter’de liste kutularını şöyle oluşturuyorduk:

```
from Tkinter import *

pencere = Tk()

listekutusu = Listbox()
listekutusu.pack()

mainloop()
```

İsterseniz bu liste kutusunun daha belirgin olması için buna birkaç öğe ekleyelim:

```
from Tkinter import *

pencere = Tk()

listekutusu = Listbox()
listekutusu.pack()

dagitimlar = ["Debian", "Ubuntu", "Mandriva", "Arch", "Gentoo"]

for i in dagitimlar:
    listekutusu.insert(0, i)

mainloop()
```

Elbette “dagitimlar” adlı listedeki öğelerin “Listbox”taki konumlanışını ters de çevirebiliriz:

```
from Tkinter import *

pencere = Tk()

listekutusu = Listbox()
listekutusu.pack()

dagitimlar = ["Debian", "Ubuntu", "Mandriva", "Arch", "Gentoo"]

for i in dagitimlar:
    listekutusu.insert(END, i)

mainloop()
```

Burada “listekutusu.insert()” satırında bir önceki kodlarda “0” sıra numarasını verirken, yukarıdaki kodlarda “END” ifadesini kullandığımıza dikkat edin.

Gelin şimdi isterseniz bu liste öğelerinin her birine bir görev atayalım. Yani mesela kullanıcı “Debian” adlı öğenin üzerine çift tıkladığında kendisine Debian GNU/Linux hakkında bilgi verelim:

```
#-*-coding: utf-8-*-
from Tkinter import *

pencere = Tk()
pencere.geometry("400x200")
```

```
listekutusu = Listbox()
listekutusu["relief"] = "raised"
listekutusu.pack(side=LEFT, anchor=NW, fill=BOTH)

etiket = Label()
etiket["text"] = ""

dagitimlar = ["Debian", "Ubuntu", "Mandriva", "Arch", "Gentoo"]

for i in dagitimlar:
    listekutusu.insert(END, i)

def gosterici(event):
    etiket["text"] = "%s bir GNU/Linux dağıtımıdır!"%listekutusu.get(ACTIVE)
    etiket.pack()

listekutusu.bind("<Double-Button-1>", gosterici)

mainloop()
```

Burada bizi ilgilendiren kısım şu satır:

```
listekutusu.bind("<Double-Button-1>", gosterici)
```

Liste kutusuna görev atama işlemini bu satır yardımıyla yaptık. Kullanıcı listekutusu içindeki herhangi bir öğeye çift tıkladığı zaman, tıklanan öğeyle ilgili bilgi veriyoruz. Listbox() adlı pencere aracının kendisine ait bir "command" seçeneği olmadığı için, istediğimiz işlevi, daha önceki derslerde gördüğümüz "bind" metodu yardımıyla hallediyoruz. Burada "Double-Button-1" ifadesini kullandığımıza dikkat edin. Bu ifade farenin sol tuşuna çift tıklama hareketini temsil ediyor. Liste kutularındaki öğelere işlev atamak istediğimiz zaman en doğru sonucu "Double-Button-1" ile elde ediyoruz. Öteki seçenekler her zaman istediğimiz işlevi yerine getirmeyebilir...

## 6.3 Pencereleri Başlıksız Hale Getirmek

Tkinter'de standart bir pencerenin nasıl oluşturulacağını biliyoruz:

```
pencere = Tk()
```

Bu şekilde bir pencere oluşturduğumuzda, elde ettiğimiz pencere, bir pencerenin sahip olması gereken bütün özellikleri taşır. Dolayısıyla pencereyi kapatmak, pencereyi küçültmek veya pencereyi aşağı indirmek istediğimizde özel olarak herhangi bir kod yazmamıza gerek yoktur. Pencere başlığı üzerindeki çarpı, kare ve eksi düğmelerine tıklayarak gerekli işlevleri yerine getirebiliriz. Normal şartlar altında bu tür özelliklerin olmadığı bir pencere pek bir işimize yaramaz. Ama bazen pencerenin sürükleme çubuğu dahil hiç bir özelliğinin olmamasını isteyebiliriz. İşte bu bölümde bu isteğimizi nasıl yerine getirebileceğimizi göreceğiz. Şimdi normal bir şekilde programımızı yazalım:

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

from Tkinter import *

pencere = Tk()
```

```
mainloop()
```

Bu şekilde içi boş bir pencere oluşturduk. Şimdi bu kodlara şöyle bir şey ekleyelim:

```
pencere.overrideredirect(1)
```

Burada "overrideredirect" kelimesinin yazılışına azami özen göstermek gerekir. Uzun bir kelime olduğu için yanlış yazılma olasılığı oldukça yüksektir.

Kodlarımızın son hali şöyle oldu:

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

from Tkinter import *

pencere = Tk()
pencere.overrideredirect(1)

mainloop()
```

Yalnız bu kodları mutlaka komut satırından çalıştırın. Kesinlikle yazdığınız betiğin üzerine çift tıklayarak çalıştırmayı denemeyin.

Şimdi komut satırını kullanarak betiğimizi çalıştırıyoruz. Gördüğünüz gibi, bomboş bir kare elde ettik. Bu pencerenin, kendisini kapatmamızı sağlayacak bir düğmesi bile yok... Şimdi sanırım neden bu betiği komut satırından çalıştırdığımızı anlamışsınızdır. Bu başlıksız pencereyi CTRL+C (GNU/Linux) veya CTRL+Z (Windows) tuşlarına basarak kapatabilirsiniz. Ya da doğrudan komut ekranını kapatarak da bu pencereyi ortadan kaldırabilirsiniz.

Peki böyle başlıksız bir pencere oluşturmak ne işimize yarar? Mesela bu başlıksız pencereleri, bir düğmeye tıklandığında aşağıda doğru açılan bir menü olarak kullanabiliriz. Şu örneğe bakalım:

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

from Tkinter import *

pencere = Tk()


def liste():
    yenipencere = Toplevel()
    x = dgm.winfo_rootx()
    y = dgm.winfo_rooty() + dgm.winfo_height()
    yenipencere.geometry('+%d+%d' % (x,y))
    menu = Listbox(yenipencere)
    menu.pack()

    yenipencere.overrideredirect(1)

    dagitimlar = ["Debian", "Ubuntu", "Mandriva", "Arch", "Gentoo"]

    for i in dagitimlar:
        menu.insert(0, i)

dgm = Button(text="deneme", command=liste)
dgm.pack(fill=X)
```

```
mainloop()
```

Bu kodlar içinde geçen “x” ve “y” değişkenlerinin ne olduğunu bir sonraki bölümde inceleyeceğiz. Şimdilik bu değişkenlere fazla takılmayalım. Bu “x” ve “y” değişkenleri “yenpencere”nin ana pencereye ve bunun üzerindeki “deneme” adlı düğmeye göre konumunu belirlememizi sağlıyor. Burada “deneme” adlı düğmeye tıkladığımızda bir seçim kutusu açıldığını görüyoruz. Elbette burada sadece kavramsal olarak durumu göstermek için bir örnek verdik. Şu haliyle yukarıdaki kodlar oldukça eksik. Amacım yalnızca başlıksız pencerelerle neler yapılabileceğine dair ufak bir örnek göstermekti... Yukarıdaki kodları işe yarar bir hale getirmek için üzerinde biraz çalışmanız gerekir.

Hatırlarsanız, önceki bölümlerden birinde Pencere Araçlarına İpucu Metni Eklemek” başlıklı bir konu işlemiştik. Orada gösterdiğimiz “wcktooltips.py” adlı modülde de bu “overrideredirect” özelliğinden yararlanılıyor. Şu adrese gidip wcktooltips modülünün kaynağına baktığımızda kodlar arasında “self.popup.overrideredirect(1)” satırını görüyoruz. Yani Fredrik Lundh ipucu penceresini oluşturmak için başlıksız pencerelerden yararlanmış... Gördüğünüz gibi başlıksız pencereler pek çok yerde işimize yarayabiliyor.

## 6.4 Pencere/Ekran Koordinatları ve Boyutları

Bu bölümde bir pencerenin veya üzerinde çalıştığımız ekranın koordinatlarını ve boyutlarını nasıl öğrenebileceğimizi inceleyeğiz. Peki bu bilgi ne işimize yarayacak? Mesela yazdığımız program ilk açıldığında kullanıcının ekranını ortalasın istiyorsak, programımızın çalıştırıldığı bilgisayar monitörünün boyutlarını/çözünürlüğünü bilmemiz gerekir, ki buna göre ekranın orta noktasını hesaplayabilelim... Ayrıca yazdığımız program içindeki pencere araçlarını da özellikle belli noktalarda konumlandırmak istiyorsak pencere ve üzerindeki araçların koordinatlarını bilmemiz gerekir. İsterseniz öncelikle hangi metotlardan yararlanabileceğimize bakalım:

```
>>> from Tkinter import *

>>> pencere = Tk()

>>> for i in dir(pencere):
...     if i.startswith("winfo"):
...         print i
...
winfo_atom
winfo_atomname
winfo_cells
winfo_children
winfo_class
winfo_colormapfull
winfo_containing
winfo_depth
winfo_exists
winfo_fpixels
winfo_geometry
winfo_height
winfo_id
winfo_interps
winfo_ismapped
winfo_manager
winfo_name
winfo_parent
```

```
winfo_pathname
winfo_pixels
winfo_pointerx
winfo_pointerxy
winfo_pointery
winfo_reqheight
winfo_reqwidth
winfo_rgb
winfo_rootx
winfo_rooty
winfo_screen
winfo_screencells
winfo_screendepth
winfo_screenheight
winfo_screenmmheight
winfo_screenmmwidth
winfo_screenvisual
winfo_screenwidth
winfo_server
winfo_toplevel
winfo_viewable
winfo_visual
winfo_visualid
winfo_visualsavailable
winfo_vrootheight
winfo_vrootwidth
winfo_vrootx
winfo_vrooty
winfo_width
winfo_x
winfo_y
```

Gördüğünüz gibi, elimizde epey metot var. Yukarıdaki kod yardımıyla Tk() sınıfı içindeki, "winfo" ile başlayan bütün metotları listelediğimize dikkat edin... Biz bu bölümde bu metotların hepsini değil, bunlar arasında en sık kullanılanları inceleyeceğiz. İnceleyeceğimiz metotlar şunlar olacak:

winfo_height

winfo_width

winfo_rootx

winfo_rooty

winfo_screenheight

winfo_screenwidth

winfo_x

winfo_y

Hemen ilk metodumuzla işe başlayalım:

**winfo_height()**

Bu metot, oluşturduğumuz pencerenin (veya pencere aracının) yüksekliği hakkında bilgi verir bize. Şöyle bir örnekle durumu görelim:

```
>>> pencere = Tk()
>>> yukseklik = pencere.winfo_height()
```

```
>>> print yukseklik

1
```

Galiba bu çıktı pek fazla bir şey anlatmıyor bize. Bu kodlardan böyle bir çıktı almamızın nedeni bir pencere ilk oluşturulduğunda yükseklik değerinin "1" olarak kabul edilmesidir. Eğer pencerenin gerçek boyutunu görmek istersek yukarıdaki kodları şu şekilde yazmamız gerekir:

```
pencere = Tk()

pencere.update()

yukseklik = pencere.winfo_height()
print yukseklik

mainloop()
```

Burada kullandığımız "pencere.update()" komutu penceremizin mevcut durumunu güncellememizi sağlıyor. Dolayısıyla bu komut bize doğru bir şekilde "200" gibi bir çıktı veriyor. Buradan aldığımız sonuca göre, oluşturduğumuz pencerenin yüksekliği 200 piksel. Dediğimiz gibi, bu metodu yalnızca pencere üzerine uygulamak zorunda değiliz. Aynı metodu pencere araçlarıyla birlikte de kullanabiliriz:

```
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

btn = Button(text="deneme")
btn.pack()

btn.update()

yukseklik = btn.winfo_height()

print yukseklik

mainloop()
```

Burada da düğme bilgilerini güncellemek için "btn.update()" gibi bir komuttan yararlandığımıza dikkat edin. Eğer bu örnekte update() metodunu kullanmazsak biraz önce olduğu gibi, alacağımız çıktı "1" olacaktır.

Elbette "winfo_height()" metodunu bir değişkene atamadan da kullanabiliriz. Ama açıkçası o şekilde pek pratik olmayacaktır...

Şimdi sıra geldi winfo_width metodunu incelemeye:

**winfo_width()**

İlk incelediğimiz metot olan "winfo_height", bir pencere veya pencere aracının yüksekliğini veriyordu. "winfo_width()" metodu ise pencere veya pencere aracının genişliğini verir. Hemen görelim:

```
from Tkinter import *

pencere = Tk()

btn = Button(text="deneme")
```

```
btn.pack()

pencere.update()

genislik = pencere.winfo_width()

print genislik

mainloop()
```

Buradan aldığınız çıktı, pencerenin genişliğini gösterir. Eğer pencerenin değil de düğmenin genişliğini almak isterseniz ne yapmanız gerektiğini tahmin edebilirsiniz:

```
from Tkinter import *

pencere = Tk()

btn = Button(text="deneme")
btn.pack()

btn.update()

genislik = btn.winfo_width()

print genislik

mainloop()
```

Muhtemelen aldığınız çıktı, pencereden aldığınız çıktı ile aynı olacaktır. Çünkü Tkinter pencere üzerindeki araçlara göre pencerenin boyutunu ayarlıyor. Burada da Tkinter pencere üzerinde tek bir pencere aracı olduğu için, pencereyi "deneme" adlı düğmenin boyutu kadar ufalttı. Dolayısıyla pencerenin kendisi ve düğme aynı genişliğe sahip oldu... Pencerenin üzerinde birden fazla pencere aracı olduğu durumlarda winfo_width() metodunun işlevi daha net görülecektir.

**winfo_rootx()**

Bu metot pencere veya pencere araçlarının x düzlemi üzerindeki koordinatını verir. Mesela:

```
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

btn = Button(text="deneme")
btn.pack()

pencere.update()

xkoord = pencere.winfo_rootx()

print xkoord

mainloop()
```

Ben bu kodları kendi bilgisayarımda çalıştırdığımda "965" çıktısını aldım. Benim monitörümün çözünürlüğü "1440x900". Demek ki bu pencerenin sol kenarı, ekranın soldan sağa 965'inci noktasına denk geliyormuş. Bir de şuna bakalım:

```
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

btn = Button(text="deneme")
btn.pack()

btn.update()

xkoord = btn.winfo_rootx()

print xkoord

mainloop()
```

Bu kodları çalıştırdığımda ise konsolda “1027” çıktısını gördüm. Demek ki “deneme” adlı düğmenin sol kenarı “1440x900”lük monitörün 1027’inci noktasına denk geliyormuş...

**winfo_rooty()**

Bir önceki metodumuz olan winfo_rootx() x-koordinatlarının bilgisini veriyordu. winfo_rooty() ise y-koordinatlarının bilgisini verir:

```
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

btn = Button(text="deneme")
btn.pack()

pencere.update()

ykoord = pencere.winfo_rooty()

print ykoord

mainloop()
```

Buradan aldığımız sonuç, pencerenin üst sınırının y düzlemi üzerinde hangi noktaya karşılık geldiğini gösteriyor. Penceremiz ekranın en tepesinde bile olsa bu kodlar “0” veya “1” gibi bir sonuç vermez. Çünkü pencere başlığının hemen altındaki alanın karşılık geldiği nokta hesaplanmaktadır. Bir de şu örneğe bakalım:

```
from Tkinter import *

pencere = Tk()
pencere.geometry("200x200")

btn = Button(text="deneme")
btn.pack(pady=30)

btn.update()

ykoord = btn.winfo_rooty()

print ykoord
```

```
mainloop()
```

Burada “pady” seçeneğini kullanarak düğmeyi biraz aşağıya kaydırdık, ki bir önceki kodla arasındaki farkı görebilelim...

**winfo_screenheight()**

Bu metot ekran yüksekliğinin kaç olduğunu söyler. Örneğin 1024x768’lik bir çözünürlüte bu metodun verdiği değer 768 olacaktır...

```
from Tkinter import *

pencere = Tk()

ekran_y = pencere.winfo_screenheight()

print ekran_y

mainloop()

winfo_screenwidth()
```

Bu metot da winfo_screenheight() metoduna benzer. Ama onun aksine, bir pencerenin yüksekliğini değil, genişliğini verir. Dolayısıyla 1024x768’lik bir ekran çözünürlüğünde bu değer 1024 olacaktır:

```
from Tkinter import *

pencere = Tk()

ekran_g = pencere.winfo_screenwidth()

print ekran_g

mainloop()
```

En önemli “winfo” metotlarını gördüğümüze göre bunları kullanarak bazı yararlı işler yapmaya başlayabiliriz...

## 6.5 Programı Tam Ekran olarak Çalıştırmak

Geçen bölümde “winfo” metotlarını gördüğümüze göre, bu bölümde bu metotları kullanarak bazı faydalı işler yapmaya çalışacağız... Bu metotlar yazdığımız programları istediğimiz şekilde konumlandırmada ve boyutlandırmada bize epey yardımcı olacaklardır. Örneğin geçen bölümde öğrendiklerimizi kullanarak, yazdığımız bir programı tam ekran çalıştırabiliriz:

```
from Tkinter import*

pencere = Tk()
gen = pencere.winfo_screenwidth()
yuks = pencere.winfo_screenheight()
pencere.geometry("%dx%d"%(gen, yuks))

dgm = Button(text="~~~~~~~~~~~~TAM EKRAN~~~~~~~~~~")
dgm.pack(expand=YES, fill=BOTH)
```

```
pencere.mainloop()
```

Burada yaptığımız şey şu: Önce “pencere.winfo_screenwidth()” ifadesi yardımıyla ekran genişliğini alıyoruz. Daha sonra “pencere.winfo_screenheight()” ifadesini kullanarak ekran yüksekliğini öğreniyoruz. Bu bilgileri, kullanım kolaylığı açısından “gen” ve “yuks” adlı iki değişkene atadık. Ardından da “pencere.geometry” içinde bu değerleri kullanarak programımızın ilk açılışta ekranın tamamını kaplamasını sağladık.

Python’un arkaplanda neler çevirdiğini daha net görmek için, kullandığımız bu “gen” ve “yuks” değişkenlerini ekrana yazdırmak isteyebilirsiniz...

## 6.6 Ekranı Ortalamak

“winfo” metotlarını kullanarak, yazdığımız bir programın ilk açıldığında ekranın tam ortasına denk gelmesini de sağlayabiliriz:

```
from Tkinter import*

pencere = Tk()

pgen = 200
pyuks = 200

ekrangen = pencere.winfo_screenwidth()
ekranyuks = pencere.winfo_screenheight()

x = (ekrangen - pgen) / 2
y = (ekranyuks - pyuks) / 2

pencere.geometry("%dx%d+%d+%d"%(pgen, pyuks, x, y))

pencere.mainloop()
```

Burada önce “pgen” ve “pyuks” değişkenleri içinde programımızın genişliğini ve yüksekliğini belirttik. Bunları elbette doğrudan “pencere.geometry” ifadesi içine de yerleştirebilirdik. Ama bu iki değerle bazı hesaplamalar yapacağımız için, en iyisi bunları bir değişken içine atmak.

İlk değişkenlerimizi tanımladıktan sonra ekran genişliğini ve ekran yüksekliğini öğreniyoruz. Bunun için “winfo_screenwidth()” ve “winfo_screenheight()” metotlarını kullandık. Yine kullanım kolaylığı açısından bu iki değeri sırasıyla “ekrangen” ve “ekranyuks” adlı değişkenlere atıyoruz.

Şimdi programımızın ekranı tam ortalayabilmesi için ufak bir hesap yapmamız gerekecek... “x” değerini bulabilmek için ekran genişliğinden programımızın pencere genişliğini çıkarıp, elde ettiğimiz değeri ikiye bölüyoruz. Mesela eğer kullandığımız ekran çözünürlüğü “1024x768” ise, “x” değeri şöyle olacaktır:

```
x = (1024 - 200) / 2
```

“y” değerini bulmak için de benzer bir şekilde ekran yüksekliğinden program yüksekliğini çıkarıp, bu değeri yine ikiye bölüyoruz. Dolayısıyla “y” değeri “1024x768”lik bir ekranda şöyle olur:

```
y = (768 - 200) / 2
```

Bu hesaplamadan elde ettiğimiz verileri pencere.geometry() içinde uygun yerlere yerleştirdiğimizde, programımız ekranın tam ortasında açılacaktır....

## 6.7 Pencereleri Her Zaman En Üstte Tutmak

Bazen ana pencereye ek olarak ikinci bir pencere daha oluşturmamız gerekir. Ancak bazen programın işleyişi sırasında, aslında hep üstte durması gereken bu ikinci pencerenin, ana pencerenin arkasına düştüğünü görürüz. Böyle bir durumda yapmamız gereken şey, ikinci pencerenin daima üstte kalmasını sağlayacak bir kod yazmaktır. Neyse ki Tkinter bize böyle bir durumda kullanılmak üzere faydalı bir metot sunar. Bu metodun adı "transient()". İsterseniz hemen bununla ilgili bir örnek yapalım. Diyelim ki şöyle bir uygulamamız var:

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

from Tkinter import *

from tkFileDialog import askopenfilename

pencere = Tk()

pgen = 200
pyuks = 200

ekrangen = pencere.winfo_screenwidth()
ekranyuks = pencere.winfo_screenheight()

x = (ekrangen - pgen) / 2
y = (ekranyuks - pyuks) / 2

pencere.geometry("%dx%d+%d+%d"%(pgen, pyuks, x, y))

def yeniPencere():
    yeni = Toplevel()

    xkonum = pencere.winfo_rootx()
    ykonum = pencere.winfo_rooty()
    yeni.geometry("+%d+%d"%(xkonum, ykonum))

    ybtn = Button(yeni, text="Dosya aç", command=yeniDosya)
    ybtn.pack()

def yeniDosya():
    dosya = askopenfilename()

btn = Button(pencere, text="yeni pencere aç", command=yeniPencere)
btn.pack()

mainloop()
```

Burada gördüğümüz tkFileDialog modülünün şimdilik üzerinde durmayalım. Birkaç bölüm sonra bu ve benzeri modülleri ayrıntılı olarak inceleyeceğiz. Biz şimdilik bu modülün dosya açma işlemlerinde kullanıldığını bilelim yeter...

Programımızın, ekranın tam ortasında açılacak şekilde ayarlandığına dikkat edin. Aynı şekilde ikinci pencere de ana pencerenin üzerinde açılacak şekilde ayarlandı. Bu işlemleri "winfo"

metotları yardımıyla yaptığımızı görüyorsunuz.

Bu programı çalıştırdığımızda pencere araçlarının biraz tuhaf davrandığını görebilirsiniz. Mesela ikinci pencere üzerindeki “dosya aç” düğmesine bastığımızda açılan dosya seçme penceresi ikinci pencerenin altında kalıyor olabilir. Aynı şekilde, dosya seçme ekranında “Cancel” tuşuna bastığımızda ikinci pencere bir anda ortadan kaybolacak ve ana pencerenin arkasına gizlenecektir. Bu program belki her sistemde aynı tepkiyi vermeyebilir. Hatta belki bazı sistemlerde bu şekilde bile düzgün çalışıyor olabilir. Ama bizim istediğimiz şey, programımızın hemen her sistemde mümkün olduğunca aynı şekilde çalışmasını sağlamak... O halde hemen gerekli kodları yazalım. Yazacağımız kod çok basittir. Tek yapmamız gerekn, ikinci pencerenin ana pencereye göre üstte kalmasını garanti etmek. Bunu şu satırla yapacağız:

```
yeni.transient(pencere)
```

Yani kodlarımızın son hali şöyle olacak:

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

from Tkinter import *
from tkFileDialog import askopenfilename

pencere = Tk()

pgen = 200
pyuks = 200

ekrangen = pencere.winfo_screenwidth()
ekranyuks = pencere.winfo_screenheight()

x = (ekrangen - pgen) / 2
y = (ekranyuks - pyuks) / 2

pencere.geometry("%dx%d+%d+%d"%(pgen, pyuks, x, y))

def yeniPencere():
    yeni = Toplevel()
    yeni.transient(pencere)

    xkonum = pencere.winfo_rootx()
    ykonum = pencere.winfo_rooty()
    yeni.geometry("+%d+%d"%(xkonum, ykonum))

    ybtn = Button(yeni, text="Dosya aç", command=yeniDosya)
    ybtn.pack()

def yeniDosya():
    dosya = askopenfilename()

btn = Button(pencere, text="yeni pencere aç", command=yeniPencere)
btn.pack()

mainloop()
```

Yeni eklediğimiz satır, “yeni” adlı ikinci pencerenin ana pencereye göre hep üstte kalmasını temin ediyor. Programımızı bu şekilde çalıştırdığımızda her şeyin olması gerektiği gibi olduğunu göreceğiz. Kodlarımızı bu şekilde yazdığımızda, ikinci pencereyi açtıktan sonra ana penceye tıklasak bile ikinci pencere ana pencerenin arkasına düşmeyecektir...

# Standart Bilgi Pencereleri (Standard Dialogs)

Bu bölümün konusu Tkinter'de Standart Bilgi Pencerelerinin kullanımı. Nedir bu "Standart Bilgi Penceresi" denen şey? Diyelim ki bir program yazıyorsunuz... Bu programda herhangi bir işlemle ilgili olarak kullanıcıya bir mesaj göstermek istediğimizde karşımızda iki yol var. Birinci yolda, Tkinter'in Toplevel() aracını kullanarak, göstermek istediğimiz mesajı ve gerekli düğmeleri içeren bir pencere oluşturabiliriz. Hemen bununla ilgili bir örnek verelim:

```
# -*- coding: utf-8 -*-
from Tkinter import *

pencere = Tk()

def kapat():
    if not entry.get():
        yenipen = Toplevel()

        uyari = Label(yenipen)
        uyari["text"] = "Lütfen geçerli bir e.posta adresi yazın!"
        uyari.pack()

        uybtn = Button(yenipen)
        uybtn["text"] = "Tamam"
        uybtn["command"] = yenipen.destroy
        uybtn.pack()

    else:
        pencere.destroy()

etiket = Label()
etiket["text"] = "e.posta adresiniz:"
etiket.pack()

entry = Entry()
entry.pack()

dgm = Button()
dgm["text"] = "Gönder"
```

```
dgm["command"] = kapat
dgm.pack()

pencere.mainloop()
```

Burada eğer kullanıcı entry() aracına herhangi bir şey yazmadan "gönder" düğmesine basarsa bir uyarı penceresi açılacak ve kullanıcıyı entry() aracına geçerli bir e.posta yazması konusunda uyaracaktır.

Dediğimiz gibi, böyle bir işlev yerine getirmeniz gerekirse yukarıdaki yöntemi kullanabilirsiniz. Ama aslında buna hiç gerek yok. Tkinter bize bu tür işleri yapabilmemiz için bize bazı araçlar sunar. İşte biz de bu bölümde Tkinter'in bize sunduğu bu araçları inceleyeceğiz...

## 7.1 Hata Mesajı Gösteren Pencere

Bu bölümde inceleyeceğimiz ilk penceremiz "hata mesajı gösteren pencere" olacaktır. Yalnız bahsettiğimiz bu pencereleri kullanabilmek için öncelikle "tkMessageBox" adlı modülü içe aktarmamız gerekiyor. Şöyle:

```
from Tkinter import *
from tkMessageBox import *
```

Tkinter'de kullanıcıya bilgi mesajı gösteren bütün pencereler tkMessageBox adlı modülün içinde yer alıyor. Dolayısıyla bu pencereleri kullanmak istediğimizde, Tkinter modülüyle birlikte tkMessageBox modülünü de içe aktarmayı unutmuyoruz.

Bir önceki bölümde hatırlarsanız, kullanıcıdan e.posta adresi girmesini istediğimiz ufak bir uygulama yazmıştık. Gelin isterseniz yine o örnek üzerinden gidelim. Öncelikle kodlarımızın ne olduğuna bakalım:

```
# -*- coding: utf-8 -*-
from Tkinter import *

pencere = Tk()

def kapat():
    if not entry.get():
        yenipen = Toplevel()

        uyari = Label(yenipen)
        uyari["text"] = "Lütfen geçerli bir e.posta adresi yazın!"
        uyari.pack()

        uybtn = Button(yenipen)
        uybtn["text"] = "Tamam"
        uybtn["command"] = yenipen.destroy
        uybtn.pack()

    else:
        pencere.destroy()

etiket = Label()
etiket["text"] = "e.posta adresiniz:"
etiket.pack()

entry = Entry()
```

```
entry.pack()

dgm = Button()
dgm["text"] = "Gönder"
dgm["command"] = kapat
dgm.pack()

pencere.mainloop()
```

Gördüğünüz gibi, aslında basit bir iş için çok fazla kod yazmışız. Gelin bu kodları, tkMessageBox modülünün de yardımıyla sadeleştirelim ve hatta güzelleştirelim:

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

from Tkinter import *
from tkMessageBox import *

pencere = Tk()

def kapat():
    if not entry.get():
        showerror("Hata!", "Lütfen geçerli bir e.posta adresi girin!")
    else:
        pencere.destroy()

etiket = Label()
etiket["text"] = "e.posta adresiniz:"
etiket.pack()

entry = Entry()
entry.pack()

dgm = Button()
dgm["text"] = "Gönder"
dgm["command"] = kapat
dgm.pack()

pencere.mainloop()
```

Ne kadar hoş, değil mi?? Bu şekilde hem daha az kod yazmış oluyoruz, hem de elde ettiğimiz çıktı daha oturmuş bir görünüme sahip. Dikkat ederseniz, hata mesajımız sevimli bir hata simgesi de içeriyor... Bu hata penceresinin bize sağladıklarını kendimiz yapmaya çalışırsak epey kod yazmamız gerekir. Dolayısıyla yazacağımız programlarda böyle bir imkanımızın olduğunu da göz önünde bulundurmamız bizim için faydalı olacaktır.

Bir hata penceresinin sahip olması gereken bütün özellikleri taşıyan bu hata penceresini şu tek satırlık kod yardımıyla oluşturduk:

```
showerror("Hata!", "Lütfen geçerli bir e.posta adresi girin!")
```

Burada neyin ne olduğu belli. Ama isterseniz bu satırı biraz inceleyelim:

Gördüğünüz gibi aslında kullandığımız şey “showerror()” adlı bir fonksiyon... Bu fonksiyonu iki parametreli olarak kullandık. showerror() fonksiyonu içinde kullandığımız ilk parametre “Hata” adlı bir karakter dizisi. Bu ifade, oluşacak hata penceresinin başlığı olacak... İkinci parametremiz olan “Lütfen geçerli bir e.posta adresi girin!” adlı karakter dizisi ise hata penceremizin gövdesi, yani kullanıcıya göstereceğimiz mesajın ta kendisidir.

Burada kullandığımız showerror() fonksiyonu başka bir parametre daha alabilir. Bu parame-

trenin adı “detail”dir. Bu parametreyi şöyle kullanıyoruz:

showerror(“Hata!”, “Lütfen geçerli bir e.posta adresi girin!”, detail = “e.posta adresi uydurma olmasın!”)

“detail” parametresi, varsa hatayla ilgili ayrıntıları da kullanıcıya göstermemizi sağlıyor. Bu arada yukarıdaki örnek kod satırı eğer gözünüze uzun görünüyorsa bu satırı şu şekilde bölebilirsiniz:

```
showerror("Hata!",
          "Lütfen geçerli bir e.posta adresi girin!",
          detail = "e.posta adresi uydurma olmasın!")
```

Özellikle Python kodlarına duyarlı bir metin düzenleyici kullanıyorsanız, (mesela IDLE) virgülden sonra enter tuşuna bastığınızda metin düzenleyiciniz satırları uygun şekilde hizalayacaktır...

Elbette isterseniz daha estetik bir görüntü elde etmek için değişkenlerden de faydalanabilirsiniz:

```
baslik = "Hata!"
mesaj = "Lütfen geçerli bir e.posta adresi girin!"
detay = "e.posta adres uydurma olmasın!"

showerror(baslik, mesaj, detail = detay)
```

“detail” parametresi dışında, showerror() fonksiyonu için (ve tabii öteki bilgi mesajları için) kullanabileceğimiz başka parametreler de vardır. Mesela bu parametrelerden biri “type” parametresidir. Bu parametre ile, bilgi ekranında gösterilecek düğmelerin tipini belirleyebiliriz. Bu parametre ile birlikte kullanabileceğimiz altı farklı seçenek vardır:

**ABORTRETRYIGNORE:** Pencere üzerinde “vazgeç”, “tekrar dene”, “yoksay” düğmelerini gösterir.

**OK:** Pencere üzerinde “tamam” düğmesi gösterir.

**OKCANCEL:** Pencere üzerinde “tamam” ve “iptal” düğmelerini gösterir.

**RETRYCANCEL:** Pencere üzerinde “yeniden dene” ve “iptal” düğmelerini gösterir.

**YESNO:** Pencere üzerinde “evet” ve “hayır” düğmelerini gösterir.

**YESNOCANCEL:** Pencere üzerinde “evet”, “hayır” ve “iptal” düğmelerini gösterir.

Bu seçenekler şöyle kullanılır:

```
showerror(baslik, mesaj, detail=detay, type=ABORTRETRYIGNORE)
```

Burada “ABORTRETRYIGNORE” yerine başka herhangi bir seçeneği de koyabilirsiniz elbette...

Başka bir parametre ise “default” parametresidir. Bu parametre yardımıyla, hata penceresi ilk açıldığında hangi düğmenin seçili (etkin) olacağını belirleyebiliriz. Yani doğrudan enter tuşuna basıldığında hangi düğmenin çalışacağını bu “default” adlı parametre ile belirliyoruz.

“default” parametresi yedi farklı seçeneğe sahiptir:

- ABORT
- RETRY
- IGNORE
- OK

- CANCEL
- YES
- NO

Bu parametreyi de şöyle kullanıyoruz:

```
showerror(baslik, mesaj, detail=detay, type=ABORTRETRYIGNORE, default=RETRY)
```

En son parametremiz ise “icon” adlı olan... Bu parametre yardımıyla, bilgi penceresi üzerinde gördüğümüz simgenin türünü belirleyebiliriz. Bu parametre ile şu seçenekleri kullanabiliriz:

- ERROR (Hata simgesi)
- INFO (Bilgi simgesi)
- QUESTION (Soru simgesi)
- WARNING (Uyarı simgesi)

Bu parametreyi ise şöyle kullanacağız:

showerror(baslik, mesaj, detail=detay, type=ABORTRETRYIGNORE, default=RETRY, icon=WARNING)

Bu arada, mümkün olan yerlerde kod satırlarımızı yukarıda gibi uzun tutmamak iyi bir yaklaşım olacaktır. Dolayısıyla yukarıdaki uzun satırı şu şekle getirmek okunaklılığı artırır:

```
showerror(baslik,
          mesaj,
          detail = detay,
          type = ABORTRETRYIGNORE,
          default=RETRY,
          icon=WARNING)
```

Peki kullanıcı bu bilgi ekranındaki herhangi bir düğmeye bastığında hangi işlemin yapılacağını nasıl belirleyeceğiz? Yani mesela diyelim ki kullanıcı “OK” tuşuna bastı, işte böyle bir durumda hangi işlemin yapılması gerektiğini nasıl bulacağız? Bu işi yapmak oldukça basittir. Hemen şöyle bir örnek verelim:

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

from Tkinter import *
from tkMessageBox import *

pencere = Tk()

def soru():
    cevap = showerror("Uyarı",
                      "Bu işlem diskteki bütün verileri silecektir.",
                      type=OKCANCEL)

    if cevap == "ok":
        etiket["text"] = "disk biçimlendirildi!"

    if cevap == "cancel":
        etiket["text"] = "işlem durduruldu!"

etiket = Label(text="işlem durumu: ")
etiket.pack()
```

```
dgm = Button(text="diski biçimlendir!", command=soru)
dgm.pack()

mainloop()
```

Gördüğünüz gibi, showerror() fonksiyonunu bir değişkene atadık. Burada, "type" parametresini kullanarak, içinde "OK" ve "CANCEL" gibi iki farklı düğme barındıran bir bilgi ekranı oluşturduğumuza dikkat edin. Eğer kullanıcı "OK" tuşuna basarsa "cevap" değişkeninin değeri "ok", yok eğer kullanıcı "CANCEL" tuşuna basarsa cevap değişkeninin değeri "cancel" olacaktır. İşte bu değerleri birer "if" deyimine bağlayarak her bir düğmeye ayrı bir işlev atayabiliriz...

Burada kendi kendinize, type parametresine farklı değerler vererek denemeler yapmanızı tavsiye ederim. Mesela OKCANCEL, RETRYCANCEL, ABORTRETRYIGNORE gibi değerlerin ne tür çıktılar verdiğini kontrol ederek çıktıya göre farklı fonksiyonlar yazabilirsiniz... Mesela şöyle bir test uygulaması ile her bir düğmenin nasıl bir çıktı verdiğini kontrol edebilirsiniz:

```
from Tkinter import *
from tkMessageBox import *

pencere = Tk()

def soru():
    cevap = showerror("Uyarı",
                      "Bu işlem diskteki bütün verileri silecektir.",
                      type=ABORTRETRYIGNORE)
    print cevap

dgm = Button(text="diski biçimlendir!", command=soru)
dgm.pack()

mainloop()
```

Elbette burada çıktıları konsola veriyoruz. Ama eğer siz isterseniz arayüz üzerindeki "Label()" aracının "text" seçeneğine de bu çıktıları yazdırabilirsiniz:

```
from Tkinter import *
from tkMessageBox import *

pencere = Tk()

def soru():
    cevap = showerror("Uyarı",
                      "Bu işlem diskteki bütün verileri silecektir.",
                      type=ABORTRETRYIGNORE)

    etiket["text"] = cevap

etiket = Label()
etiket.pack()

dgm = Button(text="diski biçimlendir!", command=soru)
dgm.pack()

mainloop()
```

Gördüğünüz gibi, showerror() fonksiyonunu kullanmak zor değil. Gelin isterseniz şimdi öteki bilgi pencerelerine de şöyle bir göz atalım...

## 7.2 Bilgi Mesajı Gösteren Pencere

Bir önceki bölümde hata mesajı gösteren pencereyi işlemiştik. Bu bölümde ise "bilgi mesajı gösteren pencere"yi öğreneceğiz.

Bilgi mesajı gösterne pencere de tıpkı hata mesajı gösteren pencere gibidir. Bu kez showerror() fonksiyonu yerine "showinfo()" fonksiyonunu kullanacağız. En basit şekliyle showinfo() fonksiyonu şöyle kullanılır:

showinfo("Bilgi", "İşlem başarıyla tamamlandı!")

Gelin isterseniz bu fonksiyonu kapsamlı bir örnek içinde inceleyelim:

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

from Tkinter import *
from tkMessageBox import *

pencere = Tk()

def kapat():
    if not entry.get():
        showerror("Hata!", "Ürün sayısı belirtmediniz!")
        return "break"
    try:
        int(entry.get())
        showinfo("Bilgi", "%s ürün sepete eklendi!"%entry.get())

    except ValueError:
        showerror("Hata!", "Lütfen bir sayı girin!")


etiket = Label()
etiket["text"] = "ürün adedi:"
etiket.pack()

entry = Entry()
entry.pack()

dgm = Button()
dgm["text"] = "Gönder"
dgm["command"] = kapat
dgm.pack()

pencere.mainloop()
```

Burada birkaç işlemi aynı anda yaptık. Programımızın amacı, kullanıcının kutucuğa bir sayı yazmasını sağlamak. Ayrıca kullanıcının, kutucuğa hiç bir şey yazmadan "Gönder" tuşuna basabileceğini de hesaba katmamız gerekir... Böyle bir duruma karşı, kodlarımız içinde şu satırları yazdık:

```
if not entry.get():
    showerror("Hata!", "Ürün sayısı belirtmediniz!")
    return "break"
```

Eğer kullanıcımız, kutucuğu boş bırakırsa kendisine bir hata mesajı gösteriyoruz. Burada return "break" ifadesini kullanmamızın nedeni, hata mesajı gösterildikten sonra programın çalışmaya devam etmesini engellemek. Eğer bu ifadeyi yazmazsak, programımız kullanıcıya hata mesajı

gösterdikten sonra alt satırdaki kodları da işletmeye devam edecektir... İsterseniz o satırı kaldırıp kendi kendinize birtakım denemeler yapabilirsiniz...

Bildiğimiz gibi, entry.get() metodundan elde edilen verinin tipi karakter dizisidir. Kullanıcının bir sayı mı yoksa bir harf mi girdiğini tespit edebilmek için öncelikle, kutucuğa kullanıcı tarafından yazılan değerin sayıya dönüştürülebilen bir veri olup olmadığına bakıyoruz. Bunun için şu kodları yazdık:

```
try:
    int(entry.get())
    showinfo("Bilgi", "%s ürün sepete eklendi!"%entry.get())

except ValueError:
    showerror("Hata!", "Lütfen bir sayı girin!")
```

Burada eğer kullanıcının yazdığı şey sayıya dönüştürülebilen bir veri ise kendisine “showinfo()” fonksiyonunu kullanarak kaç ürünün sepete eklendiği bilgisini veriyoruz. Ama eğer mesela kullanıcı sayı girmek yerine “fsdfd” gibi bir şey yazarsa, bu defa “showerror()” fonksiyonunu kullanarak ona, sayı girmediğine dair bir hata mesajı gösteriyoruz. Burada “try except” bloklarını nasıl kullandığımıza dikkat edin...

Gördüğünüz gibi, showinfo() fonksiyonunun kullanımı showerror() fonksiyonunun kullanımına çok benziyor. Hatta hemen hemen aynı bile diyebiliriz...

Geçen bölümde showerror() fonksiyonunu işlerken, bu fonksiyonun birtakım parametreler de alabildiğini söylemiştik. Orada söylediklerimiz showinfo() fonksiyonu için de geçerlidir. Mesela burada da “detail” parametresini kullanarak, kullanıcıya konuyla ilgili ek bilgi verebiliriz:

```
showinfo("Bilgi", "%s ürün sepete eklendi!"%entry.get(),
         detail="Ama bence bu kadar ürün az!\nÜrünlerimizden biraz daha alın!")
```

showerror() ile showinfo() arasındaki benzerlikler bununla da sınırlı değildir. Mesela geçen bölümde öğrendiğimiz “type” parametresini burada da kullanabiliriz:

```
showinfo("Bilgi", "%s ürün sepete eklendi!"%entry.get(),
         detail="Ama bence bu kadar ürün az!\nÜrünlerimizden biraz daha alın!",
         type=OKCANCEL)
```

Gördüğünüz gibi, showerror() konusunda ne öğrendiysek burada da uygulayabiliyoruz. İsterseniz geçen bölümde öğrendiğimiz bilgileri showinfo() fonksiyonuna da uygulayarak kendi kendinize el alıştırmaları yapabilirsiniz...

Sıra geldi başka bir bilgi mesajı penceresini incelemeye...

## 7.3 Uyarı Mesajı Gösteren Pencere

Bu bölümde kullanıcılarımıza bir uyarı mesajı gösteren pencerenin nasıl oluşturulacağına bakacağız... Burada da yeni bir fonksiyonla karşılaşacağız. Ama endişelenmeye gerek yok. Bu fonksiyon da tıpkı showerror() ve showinfo() fonksiyonlarına benzer. Adı ve tabii ki üzerindeki ünlem simgesi dışında her yönüyle onlarla aynıdır. Bu fonksiyonumuzun adı showwarning(). İsterseniz bununla ilgili de basit bir örnek yapıp bu konuyu kapatalım:

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

from Tkinter import *
from tkMessageBox import *
```

```
pencere = Tk()

def kapat():
      showwarning("Bağlantı yok!", "İşleminizi gerçekleştiremiyorum!")

etiket = Label()
etiket["text"] = "ürün adedi:"
etiket.pack()

entry = Entry()
entry.pack()

dgm = Button()
dgm["text"] = "Gönder"
dgm["command"] = kapat
dgm.pack()

pencere.mainloop()
```

**Tkinter Uygulamaları:** HARMAN

**Özel Bölüm:** Canvas ve PIL

‘Canvas ve PIL’ adlı belge Sayın Yasin Meydan tarafından hazırlanmıştır.