Bölüm 4

Button

4.1 Button Nedir?

Button (düğme), tkinter içinde bir sınıftır; başka bir deyişle bir widget'tir. Üstelik, *Button*, öteki GUI araç çantalarının hemen hepsinde aynı ad ile var olan standart bir widget'tir. Arayüzlerde biçemleri ve işlevleri farklı olan düğmeler yaratılabilir. Genellikle, Button nesnesi üzerine onun işlevini belirten bir etiket konulur. Bu durumda, düğme bir *Label* nesnesi taşıyor olur. Taşınan nesne *Label*'in özeliklerine sahip olur.

Button üzerine konulan etiket için, *Label* sınıfının özelikleri geçerli olacağına göre, button üzerine text, resim (ikon) ya da her ikisi birden konulabilir. Text birden çok satır içerebilir, ama bir button için bir tek font türü seçilebilir. *Label* için bildiğimiz gibi, düğme üzerine yazılan bir text içindeki bir ve yalnızca bir harfin altı çizgili olabilir. Bu özelik, kısa yol tuşları tanımlarken işe yarar.

Arayüzlerde düğmeler, çoğunlukla, olay yönetimi için kullanılır. Düğmeye tıklandığında bir python fonksiyonu çağrılır ve fonksiyonun belirlediği olay oluşur.

Arayüz üzerindeki bir düğmeden sonrakine geçmek için *Tab* tuşu kullanılabilir. Tabii, bu iş bir olay yönetimidir. Tab tuşuna basıldığında, sonraki düğmeyi etkin kılan fonksiyonun çağrılması gerekir.

Arayüzlerde, button widgetleri pencerenin başlığında, menü çubuğunda ya da araç kutusu (toolbox) gibi yerlerde çok kullanılır. Kullanıcıyla etkileşimi sağlayan *Checkbutton* ve *Radiobutton* widget'leri özel button tipleridir. Onları ayrıca inceleyeceğiz.

4.2 Button Örnekleri

İlk işimiz arayüzde kullanacağımız bütün öğeleri (widget, fonksiyon, öznitelik vb) programa çağırmak olmalıdır. Bunun için

```
>>> from tkinter import *
```

deyimini yazıyoruz. Bunu yapmazsak, kullanacağımız her aracın önüne, örneğin, `tkinter.Button` gibi *tkinteri* yazmalıyız. Ayrıca küçük arayüzler için, kodları IDLE ile yazabiliriz. Ama karmaşık arayüzler için, kodlarımızı *NotePad* gibi basit bir kelime işlemci ile yazıp `.py` uzantılı bir dosya olarak uygun bir dizine kaydetmeyi alışkanlık edinmeliyiz. Bunu yaparsak, yazdığımız programı her istediğimizde çağırıp yeniden koşturabiliriz.

İkinci iş olarak, arayüzün anataşıyıcısını yaratmalıyız. Anataşıyıcı, arayüzün monitörde kapladığı alandır; yani ekranda açılan bir penceredir. O pencere `Tk` sınıfının bir nesnesidir. O nesneyi yaratmak için `Tk()` kurucu metodunu kullanıyoruz. Yaratılan nesneye erişebilmek için, onu bir pointer ile göstermeliyiz. Alışkanlıklarımıza göre, bir nesne yaratınca onu işaret eden bir pointer tanımlanmalıdır. Bu pointeri bir değişken adı gibi göreceğiz.

Şimdi `Tk()` kurucusunun yarattığı nesneyi işaret eden pointere `win` diyelim. `win` pointeri, yalnızca yarattığımız anataşıyıcıyı gösteriyor. O nedenle ona, anataşıyıcı'nın adıymış gibi bakabiliriz. Anataşıyıcıyı yaratıp ona `win` adını vermek için;

```
>>> win=Tk()
```

deyimini yazıyoruz.

Şimdi iki düğme yaratıp, onları win üzerine koyacağız. Düğmeler Button sınıfından türetilen nesnelerdir.

```
>>> b1 = Button(win,text="Düğme1")
>>> b2 = Button(win,text="Düğme2")
```

deyimileri iki tane düğme yaratır.

Açıklamalar:

b1, Button sınıfından Button() kurucu metodu ile türetilen bir nesnedir. Bu nesneye button (düğme) diyoruz. Kurucu metodun iki parametresi var.

Birinci parametre: `win`. Bu parametre, düğmeyi taşıyacak olan nesneyi gösterir. Bu parametreye taşıyıcı widget denilir. İleride, düğmeleri başka taşıyıcılar üzerine koyacağız.

İkinci parametre: `text="Düğme1"`. Bu parametre düğmenin üzerine konulacak etikettir. Düğme üzerine `Düğme1` yazısını yazar.

b2 düğmesi de benzer biçimde yaratılmıştır.

Şimdi, programımızın sonuna `mainloop()` fonksiyonunu ekleyelim ve Liste 4.1 biçimini alan programımızı `basitButton.py` adıyla yazıp bir yere kaydedelim. Programa `mainloop()` fonksiyonu eklenmezse, arayüz monitörde görünmez.

Liste 4.1.

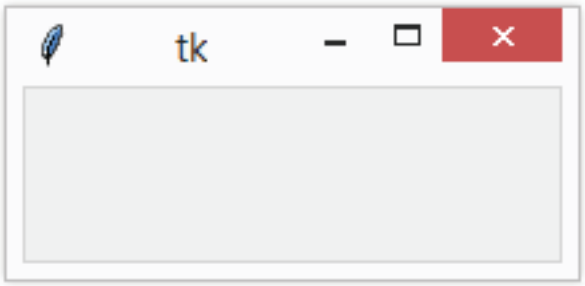
```
from tkinter import *
win = Tk()
b1 = Button(win, text="Düğme1")
b2 = Button(win, text="Düğme2")

mainloop()
```

Bu programı derleyip koşturmak için

```
python basitButton.py
```

yazmak yetecektir. Tabii, bu komutun çalışması için, ortam değişkenlerinin uygun biçimde ayarlandığını varsayıyoruz.

Ekrana gelen arayüz Şekil 4.1 biçiminde olacaktır.



Şekil 4.1: Pack Manageri Çalışmazsa

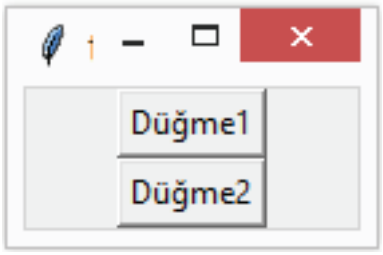
Bu şekilde, bizim yarattığımız b1 ve b2 düğmeleri görünmüyor. Düğmeleri görünür yapmak için, son satırdan önce `b1.pack()` ve `b2.pack()` konuşlandırıcı deyimlerini eklemeliyiz. `Pack` tkinter'de bir konuşlandırıcıdır (Layout Manager). İleride başka konuşlandırıcılar göreceğiz. Konuşlandırıcılar, yaratılan widgeti, taşıyıcı üzerine konuşlandırır. Konuşlandırıcı deyimlerini ekleyince, programımız Liste 4.2'deki gibi olur.

Liste 4.2.

```
from tkinter import *
win = Tk()
```

```
b1 = Button(win, text="Düğme1")
b2 = Button(win, text="Düğme2")

b1.pack()
b2.pack()

mainloop()
```

Bu pogramı derleyip koşturunca ekrana Şekil 4.2'deki arayüz çıkar.



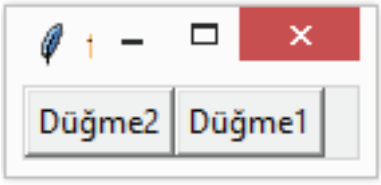
Şekil 4.2: Pack Manageri Çalışırsa

Şekil 4.2 arayüzü başlıktaki öğeler ve düğmelerin genişliğine ve yüksekliğine göre, kendi boyutlarını ayarlamıştır.

İstenirse, arayüzün boyutları ve düğmelerin konuşlanma biçemleri değiştirilebilir. Pack konuşlandırıcısı düğmeleri (TOP, LEFT, RIGHT, BOTTOM) seçeneklerine göre taşıyıcıya yerleştirir. Bu seçenekler widgeti ya taşıyıcıya göre ya da önce konuşlanan widgete göre yerleştirir. Örneğin, Liste 4.2'deki pack() konuşlandırıcılarını

```
>>> b2.pack(side=LEFT)
>>> b1.pack(side=LEFT)
```

biçiminde yazarsak, artayüzümüz



Şekil 4.3: Pack Manageri Çalışırsa

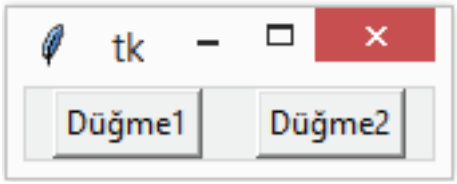
biçemini alır. Bu konuşlanma estetik görünmüyor. O zaman dümelerle taşıyıcının kenarları arasında bir yastıklama (padding) yapmak gerekebilir. Bunun için konuşlandırıcı deyimlere yatay yastıklama için `padx`, düşey yastıklama için `pady` ekleyelim. Bunların ölçü birimi px (pixel)'dir. Program

Liste 4.3 biçimini alır. Şekilden anlaşıldığı gibi, yastıklama taşıyıcının ve ilk düğmenin sol düşey kenarlarına göre yapılmıştır. Böyle olduğunu görmek için, fare ile arayüzün boyutlarını büyültünüz.

Liste 4.3.

```
from tkinter import *
win = Tk()
b1 = Button(win, text="Düğme1")
b2 = Button(win, text="Düğme2")

b1.pack(side=LEFT, padx=10)
b2.pack(side=LEFT, padx=10)

mainloop()
```

Bu programı koşturursak, ekrana çıkan arayüzümüz Şekil 4.4 biçemini alır.



Şekil 4.4: Pack Manageri Çalışırsa

biçemini alır.

Liste 4.4.

```
from tkinter import *

anaPencere = Tk()

def callback():
    print ("Tıkla!")

b = Button(anaPencere, text="Evet", command=callback)
b.pack()

mainloop()
```

Açıklamalar:

1.satır tkinter modülündeki bütün öğeleri çağırıyor.

3.satır Tk() kurucu metodu ile Tk sınıfından bir nesne yaratıyor. Bu nesnenin adı *anaPencere*'dir ve arayüzün anataşıyıcısıdır. *anaPencere*, yaratılan nesneyi işaret eden pointerdir. Ama, alışkanlıklarımıza uyarak, *anaPencere* pointerine *anaPencere* nesnesi diyoruz. Böyle demekte bir sakınca

yoktur; çünkü, *anaPencere* pointeri bir tek nesneyi işaret ediyor ve işaret edilen nesneyi, işaret eden pointer ile adlandırıyoruz.

5.satır *callback()* adlı bir fonksiyon tanımlıyor. Bu fonksiyon, düğmeye tıklanınca ekrana "Tıkla" sözcüğünü yazıyor.

8.satır, `Button` sınıfından b adlı bir nesne (düğme) yaratıyor (b pointerinin işaret ettiği nesne). Burada *Button()* kurucu metodunun üç tane parametresi var. Birinci parametre *anaPencere* adındaki nesnedir. Bu parametre, kurucu tarafından yaratılan button'un anataşıyıcı üzerine konulmasını istiyor. İkinci parametre *text="Evet"* ifadesidir. Bu parametre yaratılan düğmenin üzerine `Evet` yazılı label'i (etiket) koyuyor. Üçüncü *command=callback* parametresi, *Evet* düğmesine basılınca *callback()* fonksiyonunu çağırıyor; yani onu yürütüyor. Çağrılan fonksiyon ekrana `Tıkla` yazıyor. Böylece, `Evet` düğmesine her tıklayışta *callback()* fonksiyonu çağrılıyor ve o da ekrana "Tıkla" sözcüğünü yazdırıyor. Bu satır, aslında bir tür döngü oluşturuyor.

9.satırdaki *pack()* fonksiyonu, anataşıyıcının boyutunu içeriğine uyacak biçimde ayarlıyor.

command=callback yazılmazsa, düğmeye tıklayınca hiç bir şey olmaz. Buradaki `command` terimi düğmeye işlevsellik kazandırıyor. Bir olay oluyor ve o olay yönetiliyor.

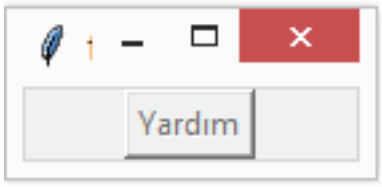
Şekil 4.5: Button Yaratma

Düğme üzerine yazılan *command=callback* olmasaydı, düğmeye tıklayınca hiç bir şey olmazdı. Buradaki *command* terimi düğmeye işlev kazandırıyor. Bir olay oluşuyor.

Bazen arayüze konulan bir düğmenin işlevsiz kalması istenebilir. Onun için, `Button()` kurucu metodundaki üçüncü parametre değeri olarak state= DISABLED yazılır. Bu durumda Liste 4.4'in 8.satırı şu biçimi alır:

```
b = Button(anaPencer, text="Yardım", state=DISABLED)
```

Bu değişikliği yaptıktan sonra Liste 4.4 programını koşturursak, ekrana Şekil 4.6'de görülen

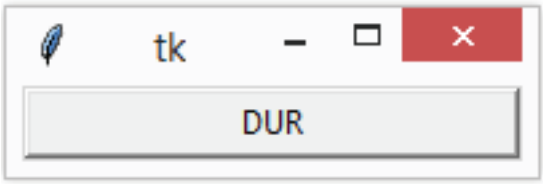


Şekil 4.6: İşlevsiz Button

arayüz gelir. Bu arayüzde Yardım düğmesine tıkladığımızda hiç bir eylem olmaz.

Düğmenin boyutlarını `height` ve `width` öznitelikleriyle ayarlayabiliriz. Örneğin, Liste 4.4'in 8.satırını şöyle değiştirirsek, ekrana Şekil 4.7'de görülen arayüz gelir.

```
b = Button(anaPencere, text='DUR', width=25, command=callback)
```



Şekil 4.7: Button Boyutunu Ayarlama