İçindekiler

PyQt5 Belgelendirmesi

Bu belgelendirme Python3.4 sürümü üzerinde PyQt5 modülünü anlatmaktadır. Python'un ve PyQt'nin farklı sürümlerini kullanıyorsanız, ufak farklılıklar dışında bu belgelendirme diğer sürümler için de geçerlidir.

Bu belgelendirmeden faydalanabilmeniz için Python dilini yeterli düzeye kadar bilmeniz gerekiyor. Bu belgede sadece PyQt5 kütüphanesine değinilecektir.

Temel Bilgiler

Genel olarak programlama dilleri, yazdığınız programlarda kullanıcılarınızla iletişimi iki farklı yoldan sağlamanıza izin verir:

1. Komut satırı arayüzü (command-line interface)
2. Grafik kullanıcı arayüzü (grapfical user interface)

Bir programlama dilinin sadece kendi imkanlarını kullanarak genellikle sadece komut satırı üzerinden çalışan programlar yazabiliriz(İstisnalar mevcut). Programlama dilleri ile grafik arayüzler tasarlayabilmek için o dille uyumlu harici kütüphanelerden yararlanmamız gerekir.

Bu durum Python dili için de aynen geçerlidir. Python dilinin sadece kendi imkanlarını kullanarak yalnızca komut satırı üzerinden çalışan programlar yazabiliriz. Ama Python'la uyumlu harici kütüphaneleri kullanarak grafik arayüze sahip programlar yazma imkanına da sahibiz.

Python'da grafik arayüz tasarlayabilmek için elimizde pek çok ek kütüphane seçeneği var. Bu seçeneklerin en önemlilerini şöyle sıralayabiliriz:

1. Tkinter
2. PyQt
3. PyGObject + GTK3
4. wxPython

Dediğimiz gibi, yukarıda saydıklarımız Python üzerinde kullanabileceğimiz grafik arayüz geliştirme kütüphanelerinin en önemlileridir. Ama aslında bunların dışında başka seçenekler de bulunur. Eğer öteki seçeneklerin ne olduğunu görmek istiyorsanız bu bağlantıdan öğrenebilirsiniz.

Yukarıda saydığımız seçeneklerin en güçlülerinden biri, elinizin altındaki bu belgelendirmenin de konusunu oluşturan, PyQt5 adlı grafik arayüz kütüphanesidir. Bu bölüm de PyQt5 adlı grafik arayüz kütüphanesi hakkındaki bilgileri edinmeye çalışacağız. PyQt5'in temeli Qt adlı bir kütüphanedir. İsterseniz PyQt5 ve bunun temelini oluşturan Qt kütüphanelerinden biraz söz edelim.

Qt Hakkında

Qt; pek çok farklı işletim sistemi üzerinde çalışabilen, oldukça profesyonel ve son derece modern bir arayüz geliştirme kütüphanesi olmasının yanında ağ programlama, veri işleme gibi bir çok özelliği barındıran bir araç takımıdır. Bu arayüz kütüphanesini TrollTech şirketinden satın alan Nokia'dan da satın alan Digia şirketi tarafından C++ adlı programlama dili ile geliştiriliyor. Bu arayüz kütüphanesini kullanarak, en basitinden en karmaşığına kadar her türlü grafik arayüzlü programı rahatlıkla tasarlayabilirsiniz. Ayrıca son sürümlerinden itibaren Android platformu için de uygulama geliştirebilirsiniz.

PyQt Hakkında

Dediğimiz gibi Qt, C++ adlı programlama dili ile geliştiriliyor. O yüzden, normal şartlar altında Qt'yi ancak C++ ile birlikte kullanabiliriz. Qt adlı bu arayüz kütüphanesini Python ile birlikte de kullanabilmek için, PyQt adlı bir ara katmandan yararlanmamız gerekiyor.

Herhangi bir dille yazılmış bir kütüphanenin başka bir dille birlikte de kullanılabilmesini sağlayan bu tür ara katmanlara teknik dilde 'bağlayıcı' **binding** adı verilir. Mesela Tkinter; aslında Tcl adlı bir programlama dili ile geliştirilen Tk adlı arayüz kütüphanesinin Python ile birlikte de kullanılabilmesini sağlamak için yazılmış bir bağlayıcıdır. Yani, nasıl Tkinter, Tcl adlı programlama dili ile geliştirilen Tk adlı arayüz kütüphanesini Python'a bağlıyorsa(Python3'de tkinter), aynı şekilde PyQt de, C++ adlı programlama dili ile geliştirilen Qt adlı grafik arayüz kütüphanesini Python'a bağlar.

PyQt5 ise, yukarıda bahsettiğimiz bu PyQt adlı bağlayıcının 5 numaralı ve şu an ki en son sürümüdür(Qt'nin 5x sürümünü kapsar).

> Python programlama dili altında grafik arayüz geliştirebileceğimiz ana seçeneklerden Tkinter ve PyGObject-pygtk2 tamamen özgür lisanslar ile dağıtılan kütüphanelerdir. Bu kütüphaneleri kullanarak hem özgür, hem özgür olmayan; hem açık, hem de kapalı kaynaklı yazılımlar geliştirebiliriz. Ancak PyQt5 de bu geçerli değildir.
>
> Eğer PyQt5 ile geliştireceğiniz program GPL lisansı altında dağıtılan bir özgür yazılımsa PyQt5'i herhangi bir lisans ücreti ödemeden ücretsiz olarak indirip kullanabilirsiniz. Ama eğer geliştireceğiniz yazılım özgür değilse bu durumda PyQt5 için bir lisans satın almanız gerekiyor. Yani, PyQt5 için herhangi bir lisans ücreti ödeyip ödemeyeceğiniz, yazdığınız programın ücretli veya ücretsiz olmasına değil, özgür veya özgür olmamasına bağlıdır. Eğer özgür ama ücretli bir yazılım geliştiriyorsanız PyQt5 lisansı satın almanıza gerek yok.
>
> PyQt5'in lisansı hakkında daha ayrıntılı bilgi edinmek için burayı ziyaret edebilirsiniz.

Kurulum

PyQt5’in kurulumunu Windows ve Linux dağıtımları için ayrı ayrı anlatacağız(Mac Os kullanıcıları avuçlarını yalasın). İlk olarak Linux dağıtımları ile başlayalım.

Linux Dağıtımları

PyQt5'i Linux dağıtımlarına kurmanın en kolay ve zahmetsiz yolu, kullandığınız dağıtımın paket yönetim sistemini kullanmaktır. Dağıtımınızın deposunda yalnızca Python'un 2.x sürümleriyle uyumlu PyQt5 paketleri yer alıyor olabilir(Python 3.x sürümü için yoksa 2.x sürümüyle uyumlu PyQt5 paketini de kurabilirsiniz.). Ubuntu ve Ubuntu tabanlı dağıtımlara Python3 ile uyumlu PyQt5 paketini kurmak için şu komutu kullanabilirsiniz:

```
sudo apt-get install python3-pyqt5
```

Pardus tabanlı(Pardus 2011) Pisi Linux 1.2 dağıtımına PyQt5 kurmak için ise şu komutu kullanabilirsiniz:

```
sudo pisi it python3-qt5
```

Bu komutu verdiğinizde PyQt5 ve bununla ilgili bütün bağımlılıklar sadece Pisi Linux da sisteminize kurulacaktır. Ubuntu ve türevi dağıtımlarda QtMultimedia gibi modüller ayrı paketlere ayrılmıştır. Dağıtımın paket yöneticilerininde arama yaparak ek paketleri kodlarınızda kullanıyorsanız kurabilirsiniz.

Eğer kurulum sırasında herhangi bir hata almadıysanız PyQt5 ile grafik arayüz tasarlamaya hazırsınız demektir. Birazdan PyQt5'i sistemimize doğru bir şekilde kurup kuramadığımızı denetleyeceğiz.

Windows

PyQt5'in Windows üzerine kurulumu oldukça basittir. Windows kullanıcıları http://www.riverbankcomputing.com/software/pyqt/download5 adresini ziyaret ederek Binary Packages başlığı altındaki, 32 bitlik exe dosyasını bilgisayarlarına indirecekler ve indirilen dosyaya çift tıklayarak normal bir şekilde PyQt5'i sistemlerine kuracaklar.

> Pythno3.x sürümünü 64 bit dahi kullanıyor olsanız yukarıda yazıldığı gibi PyQt5'in 32 bitlik sürümünü kurmalısınız. PyQt5'in 64 bitlik sürümünü kurduğunuzda, kütüphaneyi import ettiğinizde hata alacaksınız. Hata kısaca "Bu bir win32 uygulaması değil!" Neden böyle bir hatayla karşılaşıldığını henüz bilmiyorum. Şimdilik 32 bit PyQt5'i kullanmalısınız.

Son Hazırlıklar

Yukarıda anlattığımız bütün kurulum işlemlerinden sonra PyQt5'i gerçekten doğru bir şekilde kurup kuramadığınızı test etmek için Python3'ün etkileşimli kabuğunda şu komutu verin:

```
import PyQt5
```

Eğer bu komutu verdikten sonra hiçbir şey olmadan alt satıra geçiliyorsa PyQt5'i başarıyla kurdunuz demektir. Ama eğer yukarıdaki komuttan **ImportError: No module named PyQt5** gibi bir çıktı aldıysanız PyQt5'i düzgün bir şekilde kuramadınız demektir. Bu durumda başa dönüp nerede hata yaptığınızı bulmaya çalışabilir veya ilgili forumlara uğrayarak yardım isteyebilirsiniz. Biz kurulumu doğru bir şekilde yapabildiğinizi varsayarak yolumuza devam edelim.

PyQt5 adlı grafik arayüz kütüphanesini başarıyla bilgisayarınıza kurdunuz. Artık bu kütüphaneyi kullanarak grafik arayüzler tasarlamaya hazırsınız. Biz bu süreçte, karşılaştığınız sorunları çözebilmeniz için, elinizdeki belgeler aracılığıyla size mümkün olduğunca yardımcı olmaya çalışacağız. Ancak PyQt5 ile grafik arayüz tasarlarken bu belgelerin yeterli gelmediği durumlarla da karşılaşabilirsiniz. Öyle bir durumda ya forumlardan yardım isteyeceksiniz ya da deneme yanılma ile doğru yolu bulacaksınız.

> Bir Qt sınıfı ile ilgili bir bilgi arıyorsanız Qt'nin kendi belgelerine bakarak aradığınızı bulma ihtimaliniz oldukça yüksektir. Qt ile PyQt'nin kullanıldığı diller farklı olsa da aralarında çok az fark vardır. PyQt5'i öğrenip takıldığınızda araştırma yaparken de fark edeceksiniz. Qt5 belgelerine buradan ulaşabilirsiniz.

PyQt5'e Giriş

Önceki bölümde PyQt5 adlı arayüz kütüphanesi hakkındaki en temel bilgileri edindik. Öğrendiğimiz bu bilgiler sayesinde Qt'nin ne olduğunu, PyQt5 ile Qt arasında nasıl bir ilişki bulunduğunu, bu kütüphaneleri sistemimize nasıl kuracağımızı ve herhangi bir sorunla karşılaşmamız halinde nerelerden yardım alabileceğimizi öğrenmiş olduk. Bu bölümde ise PyQt5 ile ilk programlarımızı yazmaya başlayacağız. Ayrıca bu bölümde PyQt5 ile ilk programlarımızı yazmanın yanısıra, PyQt5 hakkında bazı teknik bilgiler vererek bu grafik arayüz kütüphanesini daha yakından tanımaya çalışacağız.

Basit Pencere Oluşturma

PyQt5'i daha yakından tanıyabilmek ve ilk programlarımızı yazabilmek için, çalışmalarımıza boş da olsa bir pencere tasarlayarak başlayacağız. Dilerseniz, etkileşimli kabuk üzerinde bazı denemeler yaparak pencere oluşturma sürecinin nasıl işlediğini anlamaya çalışalım. Böylece PyQt5'de boş bir pencere oluşturmak için atılması gereken adımları daha net görür, yazacağımız asıl programlar için bir ön hazırlık yapmış oluruz. Bildiğiniz gibi, ufak tefek kod parçalarını test etmek için Python'ın etkileşimli kabuğu mükemmel bir ortamdır.

Şimdi hemen etkileşimli kabuk ortamını çalıştıralım ve sırasıyla şu kodları yazalım:

```
>>> from PyQt5 import QtWidgets
>>> import sys
```

PyQt5 ile geliştirme yaparken temel olarak iki modülü mutlaka içe aktarmamız gerekiyor. Bunlar PyQt5 modülü içindeki QtWidgets sınıfı ve gömülü modüllerden biri olan sys modülü. Bu modüllerin sunduğu işlevlerden programlarımız içinde yararlanacağımız için, bu modülleri içe aktarmayı unutmuyoruz.

Gerekli modülleri içe aktardıktan sonra yapmamız gereken ilk iş bir QApplication nesnesi oluşturmak olmalı. Dikkat:

```
>>> uygulama = QtWidgets.QApplication(sys.argv)
```

Bu satır ilk bakışta, yapmak istediğimiz pencere oluşturma işlemi açısından gereksiz bir satırmış gibi görünebilir. Ama aslında bu kod parçası arkaplanda son derece önemli işler çevirir. Programımızla ilgili ilk ve ana ayarlar bu kod ile yerine getirilir. İlerleyen sayfalarda bu satırın tam olarak ne işe yaradığını daha iyi anlayacağız. Biz şimdilik, yazdığımız programlarda bu satırı bu şekilde kullanmamız gerektiğini bilelim yeter.

Sıra geldi penceremizi oluşturmaya. Penceremizi şu kod ile oluşturuyoruz:

```
>>> pencere = QtWidgets.QWidget()
```

Böylece penceremizi oluşturmuş olduk. Bunun için QtWidgets modülünün QWidget() adlı sınıfını parametresiz bir şekilde kullandığımıza dikkat edin. Bu sınıfı, bu şekilde parametresiz olarak kullandığımızda bir pencere oluşturmuş oluyoruz.

Ancak penceremizin ekranda görünmesi için bu kod yeterli değil. Penceremizin ekranda görünebilmesi için, pencereyi ekranda göstermek istediğimizi açık açık belirtmemiz gerekiyor. Bunun için show() adlı bir metottan yararlanacağız:

```
>>> pencere.show()
```

Bu komutu verir vermez ekranda boş bir pencere açılacaktır. Böylece PyQt5 ile ilk penceremizi oluşturmuş olduk. Gördüğünüz gibi, oluşan bu pencere, bir pencerenin sahip olması gereken bütün özellikleri taşıyor. Bu pencereyi kapatmak için pencere üzerindeki çarpı işaretine basabilirsiniz.

Etkileşimli kabuk ortamında ilk denemelerimizi başarıyla tamamladığımıza göre artık çalışmalarımızı daha ciddi bir ortama taşıyabiliriz. Şimdi boş bir metin belgesi açalım ve içine şu satırları yazalım:

```
from PyQt5 import QtWidgets
import sys

uygulama = QtWidgets.QApplication(sys.argv)

pencere = QtWidgets.QWidget()
pencere.show()
```

Bu dosyayı masaüstüne deneme.py adıyla kaydedip, herhangi bir Python programını nasıl çalıştırıyorsak o şekilde çalıştıralım.

Ne oldu? Programı çalıştırdığınızda pencere çok hızlı bir şekilde ekranda belirip kayboldu, değil mi? Bunu engellemek için programımızın son satırına şu kodu ekleyelim:

```
uygulama.exec_()
```

Yani kodlarımız tam olarak şöyle görünsün:

```
from PyQt5 import QtWidgets
import sys

uygulama = QtWidgets.QApplication(sys.argv)

pencere = QtWidgets.QWidget()
pencere.show()

uygulama.exec_()
```

Artık programımızı çalıştırdığımızda, pencere açılacak ve kapanmak için bizim çarpı düğmesine basmamızı bekleyecektir.

Gördüğünüz gibi, QtWidgets adlı alt-modül büyüklü-küçüklü harflerden oluşuyor. Bu yüzden bu alt-modülü kodlarımızın her yerinde doğru bir şekilde yazmaya çalışmak ilave bir çaba gerektiriyor. İsterseniz işlerimizi kolaylaştırmak için bu sınıfı şu şekilde içe aktarabiliriz:

```
from PyQt5.QtWidgets import *
```

Böylece kodlarımızda **QtWidgets** önekini yazmak zorunda kalmayacağız.

Gerekli modülleri içe aktardığımıza göre, ikinci adıma geçebiliriz.

İkinci adımda bir Qt uygulaması oluşturmamız gerekiyor:

```
uygulama = QApplication(sys.argv)
```

Daha önce de söylediğimiz gibi, ilk bakışta pek belli olmasa da aslında çok önemli bir satırdır bu. Bu satır sayesinde PyQt5, programımızın başlangıcından sona erişine kadar olan bütün süreci takip edebilecek ve programımızın düzgün bir şekilde çalışıp sona ermesini sağlayabilecektir. Burada oluşturduğumuz QApplication nesnesi, programımızı çalıştırdığımız masaüstü ortamı ile programımız arasındaki ilişkiyi düzenlediği için, grafik arayüze dair başka herhangi bir kod yazmadan önce bu satırı yazmış olmamız gerekiyor. Eğer bu satırı yazmazsak, programımız zaten bizi bu satırı yazmamız gerektiği konusunda uyaracaktır:

```
QWidget: Must construct a QApplication before a QPaintDevice
```

Bu satırın en önemli görevlerinden biri de, program çalıştırılırken komut satırında verilen parametreleri tutmasıdır. QApplication() sınıfına sys.argv parametresini de zaten bu yüzden veriyoruz.

Bu durumu daha net anlayabilmek için şöyle bir program yazabilirsiniz:

```
from PyQt5.QtWidgets import *
import sys

uygulama = QApplication(sys.argv)

print(uygulama.argv())
```

Bu programı yazıp deneme.py adıyla kaydettikten sonra şu komutu vererek programı çalıştırın:

```
python3 deneme.py
```

Programı bu şekilde çalıştırınca şu çıktıyı alacaksınız:

```
['deneme.py']
```

Aynı programı şimdi bir de şöyle çalıştırın:

```
python3 deneme.py --yardim
```

Bu defa şöyle bir çıktı alacaksınız:

```
['deneme.py', '--yardim']
```

Gördüğünüz gibi, QApplication nesnesi, yazdığımız programın çalıştırılması sırasında programa verilen parametrelerin listesini de tutuyor. Bu listeye QApplication nesnesinin argv() metodu aracılığıyla ulaşabildiğimizi görüyorsunuz.

Pencere oluşturma sürecinin ikinci aşamasını da geride bıraktığımıza göre üçüncü aşamaya gelebiliriz. Bu aşamada penceremizi çizeceğiz:

```
pencere = QWidget()
```

QtWidgets modülünün QWidget() sınıfını kullanarak boş bir pencere oluşturuyoruz. Ancak bu kod pencerenin görüntülenebilmesi için yeterli değil. Penceremizi oluşturduktan sonra görüntüleyebilmek için show() adlı bir metottan yararlanacağız:

```
pencere.show()
```

Böylece dördüncü aşamayı da geride bırakıp son aşamaya gelmiş olduk. Son aşamada ana döngüyü başlatmamız gerekiyor. Bunu şu satırla hallediyoruz:

```
uygulama.exec_()
```

Bu satır sayesinde programımız hızlı bir şekilde açılıp kapanmak yerine, kullanıcıdan gelecek emirleri beklemeye başlıyor. Böylece biz de penceremizi ekranda görebiliyoruz.

Programımızı derli toplu bir şekilde tekrar görelim:

```
from PyQt5.QtWidgets import *
import sys

uygulama = QApplication(sys.argv)
pencere = QWidget()
pencere.show()
uygulama.exec_()
```

Gördüğünüz gibi, PyQt5 ile boş bir pencere oluşturabilmek için altı satırlık bir program yazmamız gerekiyor. Bu kodlar ilk bakışta biraz karmaşıkmış gibi görünse de, aşamaların mantığını kavradığınız zaman yazması oldukça kolaydır.

Nesle Tabanlı Geliştirme

Yukarıda örneklerini verdiğimiz şekilde, PyQt5 ile prosedür tabanlı (yordamsal) programlama ilkelerine uygun bir geliştirme sürecini takip edebilirsiniz. Ancak hem kullanışlılık açısından, hem de internet üzerinde bulacağınız örnek programların yapısı nedeniyle prosedür tabanlı programlama yerine nesneye yönelik programlama tarzını benimsemenizi tavsiye ederim.

Bir örnek verelim.

Bildiğiniz gibi, PyQt5'te boş bir pencereyi şu kodlarla oluşturuyoruz:

```
from PyQt5.QtWidgets import *
import sys

uygulama = QApplication(sys.argv)
pencere = QWidget()
pencere.show()

uygulama.exec_()
```

Burada programımız, karşısına çıkan kodları belli bir prosedürü takip ederek tek tek çalıştırıyor. Ancak internet üzerinde veya başka kaynaklarda bu şekilde yazılmış kod pek göremezsiniz. Hem daha kullanışlı olması, hem de kodların bakımını kolaylaştırması nedeniyle grafik arayüz tasarlanırken genellikle nesneye yönelik programlama tarzını takip etmek çok daha mantıklı olacaktır. Dolayısıyla yukarıdaki kodları şu şekilde yazabiliriz:

```
from PyQt5.QtWidgets import *
import sys

class YeniPencere(QWidget):
    def __init__(self):
        QWidget.__init__(self)

uygulama = QApplication(sys.argv)
pencere = YeniPencere()
pencere.show()
uygulama.exec_()
```

Eğer taban sınıfı (QWidget) iki kez yazmak istemiyorsanız super() fonksiyonundan yararlanabilirsiniz:

```
from PyQt5.QtWidgets import *
import sys

class YeniPencere(QWidget):
    def __init__(self):
        super(YeniPencere, self).__init__()

uygulama = QApplication(sys.argv)
pencere = YeniPencere()
pencere.show()
uygulama.exec_()
```

Hatta, Python3 ile gelen yeni bir özellikten yararlanarak, ne taban sınıfı, ne de sınıf adını iki kez belirtmeyi tercih edebilirsiniz:

```
from PyQt5.QtWidgets import *
import sys

class YeniPencere(QWidget):
    def __init__(self):
        super().__init__()

uygulama = QApplication(sys.argv)
pencere = YeniPencere()
pencere.show()
uygulama.exec_()
```

Bu şekilde, aynı şeyleri tekrar tekrar yazma zahmetinden kurtulmanın yanısıra, sınıf adında veya miras alınan taban sınıfta bir değişiklik yapmanız gerektiğinde **init**() metodunun içeriğini de uygun bir şekilde değiştirme derdini bertaraf etmiş olursunuz.

Ana Pencere Türleri

Biz şimdiye kadar verdiğimiz örneklerde bir pencere oluşturabilmek için QWidget() adlı bir sınıftan yararlandık. Ancak pencere oluşturmak için kullanabileceğimiz tek sınıf bu değil. PyQt5 ile farklı amaçlara hizmet eden, farklı görünüm ve işlevlere sahip pencereler oluşturma imkanına sahibiz. Bu bölümde PyQt5'ün bize hangi tür pencereleri sunduğunu inceleyeceğiz.

QDialog

QDialog sınıfı da, tıpkı QWidget sınıfı gibi pencere oluşturmak için kullanılabilir. Örneğin:

```
from PyQt5.QtWidgets import *
import sys

class YeniPencere(QDialog):
    def __init__(self):
        super().__init__()

uygulama = QApplication(sys.argv)
pencere = YeniPencere()
pencere.show()
uygulama.exec_()
```

Gördüğünüz gibi, bu defa QWidget sınıfını değil, QDialog sınıfını miras aldık. Bu kodlar ile oluşturulan pencerenin, görünüş olarak QWidget ile oluşturulan pencereden farklı olduğuna dikkat edin. Bu tür pencerelerde büyütme-küçültme-kapatma düğmeleri standart değildir. Mesela bazı işletim sistemlerinde bir kapatma düğmesi ve bir de ‘bu nedir?’ düğmesi yer alırken, bazı işletim sistemlerinde kapatma ve büyütme düğmeleri, başka işletim sistemlerinde ise sadece kapatma düğmesi bulunur.

PyQt5 de QDialog sınıfı miras alan kullanıma hazır bir takım sınıflar da mevcuttur. Sırası geldiği zaman bunlara da değineceğiz.

QWidget

QWidget sınıfı, öteki bütün pencere sınıflarının atasıdır. Öteki bütün pencere sınıfları QWidget sınıfını miras alır. Daha önce verdiğimiz örneklerden de bildiğimiz gibi, bu sınıf yardımıyla, bir pencerenin sahip olması gereken her türlü temel araca ve işleve sahip pencereler oluşturabiliyoruz:

```
from PyQt5.QtWidgets import *
import sys

class YeniPencere(QWidget):
    def __init__(self):
        super().__init__()

uygulama = QApplication(sys.argv)
pencere = YeniPencere()
pencere.show()
uygulama.exec_()
```

Bu sınıfın oluşturduğu pencereyi, içi bomboş bir dikdörtgen olarak düşünebilirsiniz. Gördüğünüz gibi, QWidget sınıfını miras alarak oluşturduğumuz pencere, bir pencerenin sahip olması gereken, büyütme küçültme düğmesi, kapatma düğmesi, başlık çubuğu gibi araçlara ve kenarından tutup çekildiğinde büyüyüp küçülme, ekran üzerinde fare ile sürüklenerek taşınabilme gibi işlevlere sahip.

QMainWindow

QMainWindow sınıfı, öteki iki sınıfa kıyasla daha ayrıntılı bir pencere yapısı oluşturmamızı sağlar. Bu sınıfı da şu şekilde kullanıyoruz:

```
from PyQt5.QtWidgets import *
import sys

class YeniPencere(QMainWindow):
    def __init__(self):
        super().__init__()

uygulama = QApplication(sys.argv)
pencere = YeniPencere()
pencere.show()
uygulama.exec_()
```

Boş haliyle belli olmasa da aslında QMainWindow şu yapıya sahip bir pencere oluşturur:

Gördüğünüz gibi, bu pencerede menü çubuğu **QMenuBar**, araç çubukları **QToolBar**, ayrılabilen pencere araçları **QDockWidget** orta pencere aracı **QWidget** ve durum çubuğu **QStatusBar** için önceden belirlenmiş alanlar var. Dolayısıyla mesela bu pencereye ait metotları kullanarak oluşturacağınız bir durum çubuğu pencerenin alt tarafında kendisine ayrılan konuma yerleşecektir. Tabii bu pencere araçlarını kullanmak sizin kararınıza bağlı olmaktadır.

Yazdığınız programlarda ihtiyacınıza göre bu üç pencereden birini veya birkaçını kullanabilirsiniz.

Temel PyQt5 bilgileri üzerine konuşurken, PyQt5'in mevcut kütüphaneler arasında en güçlü grafik arayüz geliştirme kütüphanelerinden biri olduğunu söylemiştik hatırlarsanız. Gerçekten de PyQt5 aklınıza gelebilecek her türlü grafik arayüzü tasarlamanızı sağlayacak araçlara ve metotlara sahiptir. Mesela biraz önce, farklı sınıfları kullanarak oluşturduğumuz boş pencereleri ele alalım. PyQt5 bize bu boş pencerelerin pek çok niteliği üzerinde değişiklik yapma imkanı verir. PyQt5 ile oluşturduğunuz bir pencere ile neler yapabileceğinizi görmek için bu pencerenin hangi metotlara sahip olduğunu Qt5 belgelerini inceleyerek kontrol edebilirsiniz.

Temel Pencere İşlemleri

Önceki bölümde PyQt5 ile nasıl boş bir pencere oluşturacağımızı öğrendik. Öğrendiğimiz bilgiler sayesinde, oluşturduğumuz pencerenin hangi metotlara sahip olduğunu da biliyoruz. Bu metotlar yardımıyla bir PyQt5 penceresinin pek çok özelliği üzerinde değişiklikler ve sorgulamalar yapabileceğiz.

Elbette biz henüz bu listede gördüğümüz bütün metotları incelemeyeceğiz. Henüz PyQt5 konusundaki bilgilerimiz, bütün bu metotları incelemeye izin vermiyor. Ama zamanla, daha çok şey öğrendikçe, daha çok metot tanıyacağız. Biz burada sadece en temel metotları ele alacağız.

Pencere Boyutunu Ayarlamak

PyQt5 ile boş bir pencerenin nasıl oluşturulduğunu öğrendiğimize göre artık bu pencerenin bazı özellikleri üzerinde oynamalar yaparak yolumuza devam edebiliriz. Mesela önceki konularda verdiğimiz kod örneğini çalıştırdığımızda pencerenin boyutunun ön tanımlı değerlere göre oluştuğunu görürüz. Biz bunu istediğimiz şekilde değiştirme imkanına sahibiz. Bakınız:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.resize(640, 480)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Pencere() adını verdiğimiz penceremizin boyutunu resize() methodu aracılığıyla 640px genişliğinde, 480px yüksekliğinde olacak şekilde ayarlıyoruz. Bu resize() methoduna örnekte olduğu gibi ya iki adet integer değer veriyoruz ya da QtCore paketinde bulunan QSize() sınıfını tek argüman olarak atıyoruz:

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import QSize
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.resize(QSize(640, 480))

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Oldukça basit... resize methoduna iki ayrı argüman yazmak yerine QSize() sınıfına bu değerleri atayarak QSize() sınıfını tek argüman olarak atayabiliriz. Hangisini tercih edeceğiniz size kalıyor.

Pencere Konumunu Ayarlamak

Oluşturduğumuz uygulamayı her çalıştırdığımız da penceremizin ekranda farklı yerlerde çizildiğini fark etmiş olabilirsiniz. Yazdığınız bir uygulamanın ekrann neresinde açılacağını da belirlemek istiyorsanız move() methodundan faydalanabilirsiniz:

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import QPoint
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.resize(640, 480)
        self.move(300, 300)
        # veya
        self.move(QPoint(300, 300))

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Burada yazdığımız self.move(300, 300) satırı, pencerenin X düzleminde(soldan sağa) 300px, Y düzleminde ise (yukarıdan aşağıya) 400px den itibaren görüntülenmesini sağlıyor. move() methodu da resize() methodunda olduğu gibi iki ayrı integer değer ya da resize() methodunun aldığı QSize() sınıfı gibi bir sınıfı argüman olarak alabilir. Bu da örnek kodda görüldüğü üzere QPoint() sınıfıdır. Bu sınıf da QtCore paketinin içinde bulunmaktadır.

Pencere Başlığını Ayarlamak

Yazdığımız uygulamaya verdiğimiz adın ne olduğunu başlık çubuğunda mutlaka görmek isteriz değil mi? QDialog olsun, QWidget olsun, QMainWindow olsun, bu tarz pencere sınıflarında ortak olan bir method olan setWindowTitle() methoduna verdiğimiz karakter dizisinden oluşan argüman ile pencere başlığımızı ayarlayabiliriz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.resize(640, 480)
        self.move(300, 300)
        self.setWindowTitle("Pencere Uygulaması")

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Gayet basit değil mi? methodun adından da anlaşıldığı üzere(Window=Pencere, Title=Başlık) girdiğimiz karakter dizisi ile pencere başlığımızı belirledik.

Pencere Simgesini Ayarlamak

Şimdi de geliştirdiğimiz uygulamamıza pencere simgesi ekleyelim. Çok basit:

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.resize(640, 480)
        self.move(300, 300)
        self.setWindowTitle("Pencere Uygulaması")
        self.setWindowIcon(QIcon("pencere_simgesi.png"))

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Burada iki yeni bilgiyle karşılaştık. setWindowIcon() methodu ve QIcon() sınıfı. setWindowIcon() methodu argüman olarak QIcon() nesnesi alır. QIcon() sınıfı da adı argüman olarak verilen simge dosyasını yükler ve PyQt'nin kullanabileceği hale getirir. setWindowIcon() methodu da aldığı bu argümanla pencere simgesi olarak neyin belirlendiğini bu QIcon() nesnesi yardımıyla anlar ve pencere çubuğuna çizer. Eğer biz QIcon() sınıfını kullanmadan doğrudan setWindowIcon() methoduna simge dosyasının adını verirsek PyQt bize hata yaptığımızı gösterecektir.

> Qt 5x sürümüyle beraber pencere araçlarını QtWidgets paketi altında toplamıştır. QtGui paketinde ise QIcon gibi grafik arayüze etki eden sınıflar olduğu gibi bırakılmıştır.

> Kullandığınız GNU/Linux dağıtımının tema ayarları nedeniyle pencere simgelerini göremiyor olabilirsiniz. Örneğin Ubuntu GNU/Linux dağıtımında hiçbir programın simgesi pencere üzerinde görünmez.

Pencere Özelliklerini Sorgulamak

Yukarıda anlatılan belli başlı methodlarla bir pencerede yapılan en temel düzenlemeleri öğrendik. Şimdi de yeri geldiğinde kullanacağımız pencerenin özelliklerini sorgulayan belli başlı methodları görelim.

Bir ekranın boyutunu, ekrandaki konumunu nasıl ayarlayacağımızı öğrendik. Eğer istediğimiz bunları ayarlamaktan ziyade sorgulamaksa şu methodlardan yararlanırız:

```
height() #Yükseklik
width() #Genişlik
x() #X düzlemi
y() #Y düzlemi
```

Bu metotların nasıl kullanıldığını ve ne işe yaradığını göstermek için şöyle bir örnek verebiliriz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()

        print("Pencere genişliği: {}".format(self.width()))
        print("Pencere yüksekliği: {}".format(self.height()))
        print("Pencerenin x düzelemdeki yeri: {}".format(self.x()))
        print("Pencerenin y düzelemdeki yeri: {}".format(self.y()))

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Tabii siz PyQt5 de geliştikçe print() fonksiyonu dışında pencere ekranında da gösterebilecek bilgiye sahip olacaksınız.

Bir pencerenin başlığını ayarlamak için setWindowTitle() adlı bir methoddan yararlanabileceğimizi öğrenmiştik. Eğer amacımız başlık belirlemek değil de, o andaki mevcut başlığın ne olduğunu sorgulamak ise windowTitle() adlı bir metottan yararlanabiliriz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()

        print("Pencere başlığı: {}".format(self.windowTitle()))


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Böylece temel pencere metotlarını öğrenmiş olduk. En başta da dediğimiz gibi, bu metotlar bütün pencere sınıflarında ortaktır.

Temel Pencere Araçları

Şimdiye kadar PyQt5 ile sadece boş pencereler oluşturduk. Ancak grafik arayüz tasarımı sadece boş pencereler oluşturmaktan ibaret değildir elbette. Grafik arayüz tasarımı denince akla düğmeler, menüler, kutucuklar, etiketler ve buna benzer başka araçlar gelir. İşte bir grafik arayüz penceresinde bulunan bütün bu araçlara ‘pencere aracı’ **widget** adı veriyoruz. Bu bölümde, en temel pencere araçlarından biri olan QLabel adlı pencere aracından söz ederek pencere araçları konusuna giriş yapacağız.

QLabel

QLabel adlı pencere aracı, tasarladığımız grafik arayüzler üzerinde etiketler oluşturmamızı sağlar. Yani bu pencere aracını kullanarak, program penceremiz üzerinde kullanıcılarımıza programımızla ilgili mesajlar gösterebiliriz.

Bu pencere aracını şu şekilde oluşturuyoruz:

```
from PyQt5.QtWidgets import *
import sys

class EtiketliPencere(QWidget):
    def __init__(self):
        super().__init__()
        etiket = QLabel("Programa hoşgeldiniz!", self)

uygulama = QApplication(sys.argv)
pencere = EtiketliPencere()
pencere.show()
uygulama.exec_()
```

Burada QLabel pencere aracını oluşturmamızı sağlayan satır şu:

```
etiket = QLabel("Programa hoşgeldiniz!", self)
```

QLabel sınıfına verdiğimiz parametrelere dikkat edin. İlk parametre etiket içeriğinin ne olacağını, ikinci parametre ise, oluşturduğumuz bu etiketin nerede yer alacağını gösteriyor. Buna göre etiketimizin içeriği **Programa hoşgeldiniz!** adlı karakter dizisi olacak. Oluşturduğumuz bu etiket ise ana penceremiz **self** üzerinde yer alacak.

Bu noktada önemli bir konuya değinelim. PyQt5'te pencere araçlarını herhangi bir ana pencere üzerine yerleştirmek zorunda değilsiniz. Mesela şu kodları dikkatlice inceleyin:

```
from PyQt5.QtWidgets import *
import sys

class EtiketliPencere(QLabel):
    def __init__(self):
        super().__init__("Programa hoşgeldiniz!")

uygulama = QApplication(sys.argv)
pencere = EtiketliPencere()
pencere.show()
uygulama.exec_()
```

Burada QWidget sınıfı yerine QLabel sınıfını miras aldık. Bu sınıfın **init**() metodunu çağırırken de parametre olarak etiket içeriğini yazdık. Gördüğünüz gibi, biz QLabel aracını herhangi bir ana pencere üzerine yerleştirmemiş olsak da, etiketimiz kendi kendine bir pencere üzerine yerleşti. Çünkü QLabel diğer pencere araçları gibi QWidget'i miras alır. Ancak yazdığınız programlarda pencere araçlarını daha kolay kontrol edebilmek için bunları bir ana pencere üzerine yerleştirmek iyi bir fikirdir.

QLabel sınıfı yukarıdaki yazım tarzı dışında ilk argümanı **self** de alabilir. Etiket içeriğini QLabel sınıfına argüman olarak vermek zorunda değiliz. QLabel de bulunan setText() methodu etiketimizin içeriğini belirlemede bir diğer tercihimizdir:

```
from PyQt5.QtWidgets import *
import sys

class EtiketliPencere(QWidget):
    def __init__(self):
        super().__init__()
        etiket = QLabel(self)
        etiket.setText("Programa hoşgeldiniz!")

uygulama = QApplication(sys.argv)
pencere = EtiketliPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırdığınızda önceki örnekteki ile aynı sonucu elde edeceksiniz.

QLabel sınıfı sadece metin göstermek için kullanılmaz. QLabel pencere aracına html etiketleri ile biçimlendirebilirsiniz. Şu örneğe bakalım:

```
from PyQt5.QtWidgets import *
import sys

class EtiketliPencere(QWidget):
    def __init__(self):
        super().__init__()
        etiket = QLabel(self)
        etiket.setText("<p style='color:red; font-weight:bold; font-size:14pt>Programa
 hoşgeldiniz!</p>")

uygulama = QApplication(sys.argv)
pencere = EtiketliPencere()
pencere.show()
uygulama.exec_()
```

Eğer html kodlamasından anlıyorsanız setText() methoduna atadığımız argümanın ne yaptığını anlarsınız. Ancak html bilmeyenler için burada ne yaptığımızı açıklayalım. `<p>` etiketinin style özelliğine css kodu yazarak "Programa hoşgeldiniz!" metnini kırmızı renkte, kalın ve 14 punto büyüklüğünde olmasını sağladık. Eğer isterseniz `<img>` etiketiyle resim de gösterebilirsiniz:

```
etiket.setText("<img src='resimler/falanca.png'>")
```

Bunun yerine alternatif isterseniz şöyle yapabilirsiniz:

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QPixmap
import sys

class EtiketliPencere(QWidget):
    def __init__(self):
        super().__init__()
        etiket = QLabel(self)
        etiket.setPixmap(QPixmap("resimler/falanca.png"))

uygulama = QApplication(sys.argv)
pencere = EtiketliPencere()
pencere.show()
uygulama.exec_()
```

setPixmap methodu argüman olarak bir QPixmap sınıfı alır. QPixmap sınıfı da aynı QIcon gibi, resim dosyasını QLabel'in anlayacağı şekilde ayarlar. Bu method sayesinde QLabel sınıfıyla metin dışında resim dosyası da gösterebiliyoruz.

> setPixmap() methodu ile resim gösterdiğinizde setText() methodunu kullanarak karakter dizisi girseniz de pencerede sadece resim gösterilecektir.

QLineEdit

QLineEdit kısaca tek satırlık bir metin düzenleyici pencere aracıdır. Bu tek satırlık alanda girilen metni kesebilir, kopyalayabilir, silebilir ve hatta sürükle bırak işlemi gerçekleştirebilirsiniz. Kullanıcıdan veri almak için en temel pencere aracıdır. Kullanımı oldukça basittir:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        lineEdit = QLineEdit(self)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

QLineEdit pencere aracını kendi kullanım amacımıza göre özelleştirebiliriz. Örneğin kullanıcı giriş ekranı yazdığımızı varsayalım... Bize gereken şifre giriş kısmında her karakterin yerini yıldız(*) karakteriyle değiştirecek bir QLineEdit pencere aracı:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        lineEdit = QLineEdit(self)
        lineEdit.setEchoMode(QLineEdit.Password)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Böylelikle kullanıcının girdiği karakter dizisinin şifre olduğunu ve bu karakter dizisinin yerine yıldız(*) karakterinin yansıtılmasını sağladık. Zaten Echo da Türkçe'de yansıtmak anlamına gelmektedir.

Eğer biz QLineEdit'in düzenlenebilir özelliğini engellemek istersek tek satırlık bir kod yazmamız yeterli olacaktır:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        lineEdit = QLineEdit(self)
        lineEdit.setText("Bu metin düzenlenemez!")
        lineEdit.setReadOnly(True)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

setReadOnly() methodu sayesinde QLineEdit pencere aracının düzenleme özelliğini engellemiş olduk. Kodu çalıştırdığınızda QLineEdit içerisinde yazan "Bu metin düzenlenemez!" metnini hiç bir şekilde değiştiremeyiz. Tabii yazacağınız ufak bir kodla setReadOnly() methoduna **False** değerini argüman olarak vererek düzenlenebilir yapabilirsiniz.

Şimdi isterseniz QLineEdit pencere aracının önemli iki sinyaline değinelim... Uygulamamıza bir adet QLabel, bir adet de QPushButton ekleyelim:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout(self)
        self.lineEdit = QLineEdit(self)
        self.buton = QPushButton(self)
        self.buton.setText("Tamam")
        self.label = QLabel(self)
        self.label.setText("Burası değişecek!")

        layout.addWidget(self.lineEdit)
        layout.addWidget(self.buton)
        layout.addWidget(self.label)

        self.lineEdit.returnPressed.connect(self.rengiDegistir)
        self.lineEdit.textChanged.connect(self.labeleYaz)

    def rengiDegistir(self):
        self.buton.animateClick()
        self.label.setStyleSheet("color:red")

    def labeleYaz(self, metin):
        self.label.setText(metin)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Bu örnekte yeni bilgilerle karşılaşıyoruz. QVBoxLayout() pencere aracı, pencere araçlarını dikey olarak yerleştiren bir sınıftır. Bu pencere aracı her ne kadar QtWidgets paketinde olsa da gözle görünür değildir. Bu sınıfın addWidget() methodu ile QLineEdit'i, QPushButton'u ve QLabel'i alt alta sıralamış olduk. Pencereyi genişlettiğinizde orantılı bir şekilde pencere araçlarının da genişlediğini göreceksiniz.

> QVBoxLayout gibi pencere yerleştirme sınıflarını ileriki konularda işleyeceğiz.

QLineEdit, içerisindeki karakter dizisi her değiştiğinde textChanged() sinyali çalışır. bu sinyal QLineEdit pencere aracında değişim sonucunda oluşan metni sinyal aracılığı ile yayar. Yukarıdaki örnekte yuva olarak kullandığımız labeleYaz() methodunu yazarken de **metin** adında bir parametre belirledik. textChanged() sinyali ile yayılan karakter dizisi, labeleYaz() methoduna verdiğimiz parametreye argüman olarak arka planda atanır. Sonuç olarak

kullanıcı QLineEdit'in içeriğini her değiştirdiğinde labeleYaz() methodu-yuvası da çalışacak ve bu method sayesinde elde edilen karakter dizisi QLabel pencere aracına anlık olarak yazılacaktır.

Klavye tuşunda **Enter** tuşuna bastığınız da ise QLineEdit returnPressed() sinyalini yayar. Bu sinyal her hangi bir veri içermez... Yukarıdaki örnekte biz bu sinyali renginiDegistir() methoduna-yuvasına bağladık. Bu method QLabel içerisindeki metni kırmızı renge boyar. setStyeSheet() methoduna CSS kodu yazılmak suretiyle QLabel içerisindeki metnin özelliklerini düzenler.

QPushButton'un animateClick() methodu da bir slot-yuvadır. Bu method ile QLineEdit içindeyken **Enter** tuşuna basıldığında fare imleciyle bu butona tıklanmış gibi görsel bir olay gösterilmesini sağlarız.

QPushButton

Bir grafik arayüzünün olmazsa olmazlarından biri şüphesiz butonlardır. PyQt de penceremize buton yerleştirmek için QPushButton pencere aracını sağlar. Kullanımı oldukça basit:

```
from PyQt5.QtWidgets import *
import sys

class ButonluPencere(QWidget):
    def __init__(self):
        super().__init__()
        buton = QPushButton(self)
        buton.setText("Tıklayın!")

uygulama = QApplication(sys.argv)
pencere = ButonluPencere()
pencere.show()
uygulama.exec_()
```

Örnek kodda görüldüğü gibi QLabel pencere aracıyla aynı yazıma sahip. QLabel örneğinde olduğu gibi **self'i** ikinci argüman yapıp QPushButton'un içeriğini ilk parametreye de yazabiliriz. Ancak daha hoş yazım tarzı istiyorsak yukarıdaki gibi olanı tercih etmeniz daha iyi olacaktır:

```
buton = QPushButton("Tıklayın!", self)
```

Genellikle bir butonu kullanma amacımız butona tıklandığında istenilen bazı işlemleri yapmasıdır. QPushButton da bu işlemi gerçekleştirmenin yolu oldukça kolay ve yazım tarzı da Pythoniktir:

```
from PyQt5.QtWidgets import *
import sys

class ButonluPencere(QWidget):
    def __init__(self):
        super().__init__()
        buton = QPushButton(self)
        buton.setText("Tıklayın!")

        buton.clicked.connect(self.close)


uygulama = QApplication(sys.argv)
pencere = ButonluPencere()
pencere.show()
uygulama.exec_()
```

Örneği çalıştırıp butona tıklayınız. Butona tıkladığınızda pencerenin kapandığını göreceksiniz. Şimdi bu görevi yapan kodu inceleyelim:

```
buton.clicked.connect(self.close)
```

Bir PyQt uygulamasında kullanıcının eylemi ya da uygulama da gerçekleşen bir durum değişikliği durumunda pencere araçları çeşitli sinyaller yayar. Örnek kodumuzda kullandığımız QPushButton pencere aracına kullanıcı tıkladığında clicked() adında bir sinyal yayılır. Yayılan sinyali kullanabilmek için de **slot** yani **yuva**ya bağlamamız gerekir. Yuva dediğimiz terim ise bas bayağı bir methoddur. Bu methodu yukarıdaki örnekte olduğu gibi sinyalin connect() methoduna argüman olarak vererek bağlarız. close() methodu QWidget adlı pencere uygulamasını kapatmaya yarar. Siz butona tıkladığınızda bu method çalışacak ve uygulanızı kapatacaktır. Sinyal-yuva kavramı şimdilik kafanızı karıştırmasın. İleriki konularda fazlasıyla kullanacağız ve aklınızda daha iyi yer edecektir.

> Sinyal ve yuvayı yazarken parantezleri kulanmayacağımızı unutmayın!

QRadioButton

QRadioButton radyo butonları oluşturmanızı sağlar. Bu pencere aracını; özellikle internette kayıt yaparken, anket doldururken görürüz. QRadioButton'un bizim için faydası kullanıcıya sunulan seçeneklerden birini seçmesi sağlanır. Ancak tek başına bir QRadioButton bir işe yaramadığı gibi birden fazla QRadioButton ise kontrol edeceğimiz bir yapı olmadan bize sıkıntı yaratacaktır. Örneğin:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)
        self.radyobuton1 = QRadioButton(self)
        self.radyobuton1.setText("Kadın")
        self.radyobuton2  = QRadioButton(self)
        self.radyobuton2.setText("Erkek")
        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.radyobuton1)
        self.layout.addWidget(self.radyobuton2)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.hangiButon)

    def hangiButon(self):
        if self.radyobuton1.isChecked():
            print(self.radyobuton1.text())
        elif self.radyobuton2.isChecked():
            print(self.radyobuton2.text())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Bu kodla radyo butonlardan birini seçip **Tamam** butonuna tıkladığınızda seçili radyo butonunun metnini konsola yazdırır. Burada bu görevi hangiButon() methodu üstleniyor. QRadioButton'un isChecked() methodu radyo butonuna tıklanmış mı, tıklanmamış mı, True ya da False olarak değer döndürür. Yalnız sıkıntılı bir durum var! Fark ettiniz mi?.. Diğelim ki bir anket oluşturdunuz ve seçenek olarak beş, on ya da daha fazla QRadioButton kullandınız. Şimdi sıkıntıyı fark ettiniz mi? Eminim etmişsinizdir. Eğer böyle bir durumla

karşılaşırsak hangiButon() methodunda bir o kadar **if**, **elif** sorgusu kullanmak zorunda kalırdık. Kalırdık, çünkü öyle bir sorunu gidermek için PyQt bize bir sınıf sunar: QButtonGroup.

QButtonGroup, adından da anlaşılacağı gibi butonları gruplamamızı sağlar. Bu pencere aracına eklenen radyo butonlarından hangisinin seçili olduğunu tek method ile öğrenebiliriz. Bu da bize daha temiz ve kısa kod yazmamızı sağlar:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)
        self.radyobuton1 = QRadioButton(self)
        self.radyobuton1.setText("Kadın")
        self.radyobuton2  = QRadioButton(self)
        self.radyobuton2.setText("Erkek")
        self.radyobuton3  = QRadioButton(self)
        self.radyobuton3.setText("Yaşlı")
        self.radyobuton4  = QRadioButton(self)
        self.radyobuton4.setText("Çocuk")
        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.butongrup = QButtonGroup(self)
        self.butongrup.addButton(self.radyobuton1)
        self.butongrup.addButton(self.radyobuton2)
        self.butongrup.addButton(self.radyobuton3)
        self.butongrup.addButton(self.radyobuton4)

        self.layout.addWidget(self.radyobuton1)
        self.layout.addWidget(self.radyobuton2)
        self.layout.addWidget(self.radyobuton3)
        self.layout.addWidget(self.radyobuton4)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.hangiButon)

    def hangiButon(self):
        if self.butongrup.checkedButton():
            print(self.butongrup.checkedButton().text())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Daha iyi bir örnek olması için radyo butonların sayısını artırdık. QButtonGroup sınıfını tanımladık ve bu sınıfın addButton() methodu yardımıyla QRadiButton'ları grupladık. hangiButon() methodunu da iki satırlık kodla düzenledik. QButtonGroup sınıfının checkedButton() methodu tıklanmış radyo butonun kendisini döndürür. Biz burada **if** ile seçilmiş radyo buton var mı diye sorgulayıp, varsa print() ile radyo butonunun metninin çıktısını yazdırmasını sağladık. Eğer **if** ile denetlemeseydik seçim yok iken **Tamam** butonuna tıklayınca hata çıktısıyla karşılaşırdık:

```
Traceback (most recent call last):
  File "/home/metehan/deneme.py", line 34, in hangiButon
    print(self.butongrup.checkedButton().text())
AttributeError: 'NoneType' object has no attribute 'text'
```

QCheckBox

Onay butonu, yani QCheckBox pencere aracı, QRadioButton ile hemen hemen aynı bir sınıftır. QRadioButton'dan ayrılan tarafı şekli karedir ve birden fazla seçim yapılacak durumlarda kullanılır. Yalnız bu pencere aracını gruplayan bir sınıf yoktur. Yani QButtonGroup ile seçili QRadioButton'u aldığınız gibi QCheckBox(lar)ı alamazsınız. Tabii ki QButtonGroup'u QCheckBox ile kullanabilirsiniz, ama seçim yapmaya kalktığınızda sadece bir tanesinin seçilebileceğini göreceksiniz. Eğer bir çok QCheckBox pencere aracı kullanırsanız hangilerinin seçili olduğunu tek tek kontrol ederek öğrenmek zorundasınız. Kullanımına gelince:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)
        self.checkbox1 = QCheckBox(self)
        self.checkbox1.setText("Ev")
        self.checkbox2  = QCheckBox(self)
        self.checkbox2.setText("Araba")
        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.checkbox1)
        self.layout.addWidget(self.checkbox2)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.hangiButon)

    def hangiButon(self):
        if self.checkbox1.isChecked():
            print(self.checkbox1.text())
        if self.checkbox2.isChecked():
            print(self.checkbox2.text())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Oldukça basit...

QComboBox

Bu pencere aracında liste halinde verilen karakter dizilerinin grafik arayüz üzerinde kullanabilmenize imkan sağlar. Girdiğiniz veriye göre kullanıcı bu pencere aracına tıklayarak seçimini yapar ve buna göre uygulamanızın ne yapacağını ayarlarsınız. Ufak bir örnekle QComboBox pencere aracını giriş yapalım:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)
        self.comboBox = QComboBox(self)
        self.comboBox.addItem("Metehan")
        self.comboBox.addItems(["Ahmet", "Mehmet", "Banu", "Konuray"])

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.comboBox)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.hangiButon)

    def hangiButon(self):
        print(self.comboBox.currentIndex(), self.comboBox.currentText())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

self.comboBox adında bir QComboBox nesnesi tanımladık. QComboBox'un addItem() methodu ile **Metehan** adlı karakter dizisini ekledik. Eğer istersek addItems() methodu ile liste halinde vereceğimiz karakter dizilerini de çoklu olarak ekleyebiliriz. Örnekte; Ahmet, Mehmet, Banu, Konuray isimlerini de Metehan'dan sonrasına eklemiş olduk.

Yine butonumuzun clicked() sinyaline bağladığımız hangisi() yuvasıyla bazı işlemler gerçekleştiriyoruz. currentIndex() methodu seçili olan karakter dizisinin sırasını, currentText() methodu ise seçili olan karakter dizisinin kendisini verir. Uygulamayı çalıştırıp QComboBox pencere arasından **Mehmet**'i seçip **Tamam** butonuna tıkladığınızda:

```
2 Mehmet
```

çıktısını alacaksınız. Python'da listeleri bildiğinizi varsayarak bu işlemin de aynı listeden seçilen ögeyi getirmek gibi olduğunu anlayacaksınızdır.

Mesela uygulamanız için tema özelliği getirdiniz ve seçili olan temanın QComboBox da varsayılan olarak gösterilmesini istiyorsunuz. Normalde QComboBox pencere aracı ilk sırada hangi veri varsa onu gösterir, ama biz istediğimizin gösterilmesini basitçe ayarlayabiliriz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)
        self.comboBox = QComboBox(self)
        self.comboBox.addItem("Metehan")
        self.comboBox.addItems(["Ahmet", "Mehmet", "Banu", "Konuray"])

        self.comboBox.setCurrentText("Ahmet")
        # ya da
        self.comboBox.setCurrentIndex(1)

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.comboBox)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.hangiButon)

    def hangiButon(self):
        print(self.comboBox.currentIndex(), self.comboBox.currentText())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

İki basit method ile istersek karakter dizisi ile ya da karakter dizisinin sırası ile başlangıçta seçili gelecek karakter dizisini ayarlıyoruz. Bizim için bunu setCurrentText() ve setCurrentIndex() methodu sağlıyor.

Daha görsel bir kullanıma ihtiyacınız olursa, her liste öğesinin başına simge resmi koyabilirsiniz:

```
self.comboBox.setItemIcon(1, QIcon("falanca.png"))
```

> QIcon sınıfının QtGui paketinde olduğunu unutmayın.

setItemIcon() methodu ilk parametreye listedeki elemanın sırasını alır ve ikinci parametrede belirtilen simge resmini bu elemanın başında uygun çözünürlükte gösterir. İkinci parametre yukarıda görüldüğü gibi bir QIcon sınıfı alıyor. Bu sınıfı daha önceki konularda da nasıl kullandığımızı gördük.

Şimdi de son olarak öğrenmesi basit, ama kullanımı görsel olarak oldukça şıklık katan bir sınıfa değineceğiz. QCompleter! Bu anlatacağımız QCompleter sınıfını QLineEdit pencere aracında da kullanabileceğinizi de söyleyelim...

Bazı uygulamalarda QLineEdit gibi pencere araçlarına karakter dizisi yazarken aşağı doğru açılan, yazdığınız metne göre size sonuç listelendiğini görmüşsünüzdür. Örneğin; artık internet tarayıcılarının adres çubuğuna aramak istediğimiz sözcükleri yazarken bize "bu mu?" der gibi ilgili arama seçenekleri sunar. Biz de basitçe buna benzer bir uygulama geliştireceğiz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        isim_listesi = ["Ahmet", "Mehmet", "Banu", "Konuray", "Metehan", "Aleyna", "Me
lahat", "İlhan", "İdris"]
        self.tamamlayici = QCompleter(isim_listesi, self)

        self.comboBox = QComboBox(self)
        self.comboBox.setEditable(True)
        self.comboBox.addItems(isim_listesi)
        self.comboBox.setCompleter(self.tamamlayici)

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.comboBox)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.hangiButon)

    def hangiButon(self):
        print(self.comboBox.currentIndex(), self.comboBox.currentText())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Burada isim_listesi adında bir liste oluşturduk ve bir kaç tane isim girdik. Tamamlayıcı sınıfımız QCompleter'e ilk parametre olarak bu listeyi girdik ve QComboBox sınıfının setCompleter() methodu ile bu QCompleter sınıfını QComboBox'a tanıttık. Tamamlayıcının çalıştığını görebilmek için de QComboBox pencere aracını düzenlenebilir olmasını sağlayan setEditable() methodunu kullandık.

Uygulamayı çalıştırdığınızda QComboBox pencere aracında Ahmet yazacaktır. Bu ismi silip listeye göre yeni isimler girdiğinizde girdiğiniz her karaktere göre isimleri listeleyecektir. Misal ilk iki harfi **Me** girerseniz tamamlayıcı bize **Mehmet**, **Metehan** ve **Melahat** isimlerini gösterecektir. Eğer üçüncü harf olarak **h** yazarsanız sadece **Mehmet** görünecektir. Uygulamayı çalıştırarak test ettiğinizde, bu kadar basit bir uygulama ile nasıl görsellik kattığınızı görüp memnun olacaksınız.

> QLineEdit ile kullanmak için, QComboBox'daki gibi QLineEdit'in setCompleter() methodunu kullanmalısınız.

QTextEdit

QTextEdit pencere aracı, hem düz, hem zengin metin görüntülemeye yarar. Şöyle ki; saf düz yazı dışında html içerik de görüntüleyebilmektedir. QLabel konusunda nasıl html kullandıysak QTextEdit pencere aracı ile de biraz daha fazlasını kullanabiliriz. En basitinden QTextEdit kullanarak html düzenleyici yapabiliriz. Yeteri kadar kendimizi geliştirirsek kod renklendirmeli bir Python düzenleyicisi de yapabilirsiniz. Bu sizin hayal gücünüze bağlı bir durum.

QTextEdit sınıfı, burada açıklayamayacağımız kadar fazla method, özellik içerir. Bu konuda en temel işlevlere değinip bu konuyu tamamlayacağız.

> Eğer zengin metin içeriği kullanmayacaksanız QPlainTextEdit pencere aracını kullanabilirsiniz.

QTextEdit pencere aracını olduğu gibi uygulamaya koyduğumuzda sadece düz yazı yazabiliriz. Metin renklendirmesi, font biçimlendirmeleri gibi bir çok özelliği yazacağımız kodlarla kullanılabilir yaparız. İlk olarak QLineEdit ile yazacağımız html kodunu QTextEdit üzerinde biçimlendirilmiş halini gösterelim:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.textedit = QTextEdit(self)
        self.lineedit = QLineEdit(self)
        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.textedit)
        self.layout.addWidget(self.lineedit)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.htmlYaz)

    def htmlYaz(self):
        self.textedit.setHtml(self.lineedit.text())

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştıralım ve QLineEdit üzerine `<h1>PyQt5 Belgelendirmesi</h1>` yazıp **Tamam** butonuna tıklayalım. QTextEdit üzerinde göreceğiniz gibi boyutu büyük harflerle **PyQt5 Belgelendirmesi** yazacaktır.

`<h1 style='color:red'>PyQt5 Belgelendirmesi</h1>` yazıp tekrar butona tıklarsak bu sefer kırmızı renkte **PyQt5 Belgelendirmesi** görünecektir.

Yazdığımız kodda göreceğiniz gibi butonun clicked() sinyalini htmlYaz() yuvasına bağladık. Bu yuvada QTextEdit sınıfının setHtml() methoduna QLineEdit'in text() methodunu argüman olarak verdik. Biz butona tıkladığımızda o an QLineEdit üzerinde ne yazıyorsa QTextEdit üzerine html olarak biçimlendirilerek yazılır. Eğer hatalı bir html kodu ise doğal olarak doğru bir biçimlendirme olmayacaktır.

Uygulamayı denedikçe fark edeceksiniz ki, QLineEdit üzerindeki metni değiştirmemize rağmen QTextEdit üzerinde ekleme yapmak yerine sıfırdan yazacaktır. Eğer biz her yazdığımızı buton ile QTextEdit'e yolladığımızda sonrasına eklenmesini istersek append() methodunu kullanacağız:

```
def htmlYaz(self):
    self.textedit.append(self.lineedit.text())
```

Aynı şekilde insertHtml() methodunu da kullanabilirsiniz, ama göreceksiniz ki biçimlenen metin yazdığınız koda aykırı olarak bir öncekinin hemen sonrasına yazılacaktır.

QLineEdit ile QTextEdit içerisine girdiğiniz html kodlarını QTextEdit'in toHtml() methodu ile alabilirsiniz. Ancak aldığınız çıktıdan da anlayacağınız gibi yazdığınız kodlarla alakasız bir html kodu alacaksınız. Örneğin QTextEdit'e sadece `<h1>PyQt5 Belgelendirmesi</h1>` yazıp çıktısına baktığımızda şöyle bir html kodu görürüz:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-html40/stri
ct.dtd">
<html><head><meta name="qrichtext" content="1" /><style type="text/css">
p, li { white-space: pre-wrap; }
</style></head><body style=" font-family:'Oxygen-Sans'; font-size:9pt; font-weight:400
; font-style:normal;">
<p style=" margin-top:18px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt
-block-indent:0; text-indent:0px;"><span style=" font-size:xx-large; font-weight:600;">
PyQt5 Belgelendirmesi</span></p></body></html>
```

Yazdığımız dışında kalan gereksiz html kodlarını bir tarafa bırakırsak bizi ilgilendiren html kodu şudur:

```
<span style=" font-size:xx-large; font-weight:600;">PyQt5 Belgelendirmesi</span>
```

Gördüğünüz gibi biz `<h1>` etiketini kullanmış olmamıza rağmen QTextEdit aracı `<span>` etiketi kullanmış. Neden böyle olduğunu henüz bilmiyorum...

Gelelim sinyallere... QTextEdit sınıfının iki önemli sinyali vardır. Diğer sinyallerin kullanımı konumuzun dışında kalıyor. Bu sinyallerden ilki QLineEdit sınıfında da olan textChanged() sinyalidir. QLineEdit konusunu okuduğunuzu varsayarak bu sinyalin ne işe yaradığını da bildiğinizi varsayıyorum.

İkinci sinyalimiz olan cursorPositionChanged(), QTextEdit üzerindeyken kaybolup, beliren imlecimizin konumu her değiştiğinde çalışarak sinyal yayar. textChanged() ile benzer tarafı ise siz karakter dizisi girdikçe imlecinde ilerlemesidir. Bu durumda her iki sinyalde çalışır. Tabii biz klavye tuşlarıyla ya da fare ile imlecin yerini değiştirirsek sadece cursorPositionChanged() sinyali, sinyal yayar.

Bu sinyalin ne işimize yarayacağını da siz düşünün :)

QListWidget

Bu pencere aracı verilerinizi listelemeye yarayan bir sınıftır. Klasik liste mantığında olduğu gibi ekleme, çıkarma yapabilirsiniz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Bu kod pencereye boş bir QListWidget ve QPushButton çizer. Bir QListWidget'e öge eklemek istersek addItem() methodunu kullanırız:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)
        self.listwidget.addItem("Python")
        self.listwidget.addItem("Ruby")
        self.listwidget.addItem("Go")
        self.listwidget.addItem("Perl")

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Yukarı da dört adet öge ekledik. Kodu çalıştırdığınızda QListWidget'de bu ögelerin listelendiğini görürsünüz. Eğer istersek bu dört ögeyi tek seferde de ekleyebiliriz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)
        self.listwidget.addItems(["Python","Ruby","Go","Perl"])


        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

addItems() methodu liste olarak verilen karakter dizilerini QListWidget'de sırayla sıralayacaktır.

Bir QListWidget deki öge sayısını öğrenmek istersek bu sınıfın count() methodunu kullanırız. Bu method bize kaç adet öge olduğunu integer olarak verir.

QListWidget'deki ögelerden tıklanmış olan ögenin sırasını öğrenmek istersek currentRow() methodunu kullanırız. currentRow() methodu tıklanmış ögenin integer olarak sırasını verir. Yalnız unutmayın ki her zamanki gibi sıranın başında olan öge her zaman sıfırıncı sıradadır...

Eğer ön tanımlı olarak seçili olmasını istediğimiz bir öge varsa da setCurrentRow() methodu kullanılır. Bu method integer değer alır. Bu integer değer de seçili olmasını istediğiniz ögenin QListWidget'teki sırasıdır.

Listelerde bir listenin her hangi bir yerine nasıl veri eklediğimizi biliyorsunuzdur. QListWidget de bu işlem için insertItem() methodu kullanır. İlk parametreye eklenecek ögenin sırası, ikinci parametreye ise ögenin kendisini yazarız:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)
        self.listwidget.addItem("Python")
        self.listwidget.addItem("Ruby")
        self.listwidget.addItem("Go")
        self.listwidget.addItem("Perl")


        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.ekle)

    def ekle(self):
        self.listwidget.insertItem(1, "JavaScript")


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Örnek kodda **Tamam** butonuna basıldığında birinci sıraya **JavaScript** ögesi ekleniyor. Uygulama çalıştığında liste sıralaması şöyle iken:

```
Ruby
Go
Perl
```

Tamam butonuna tıklanınca şöyle olur:

```
JavaScript
Go
Perl
```

Aynı şekilde birden fazla veriyi liste arasına eklemek isterseniz insertItems() methodunu kullanabilirsiniz.

Öge eklemenin yanı sıra yeri geldiğinde öge de silmemiz gerekir. QListWidget ile öge silmenin yolu takeItem() methodudur. Kullanımı da gayet basit:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)
        self.listwidget.addItem("Python")
        self.listwidget.addItem("Ruby")
        self.listwidget.addItem("Go")
        self.listwidget.addItem("Perl")


        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)

        self.buton.clicked.connect(self.sil)


    def sil(self):
        self.listwidget.takeItem(self.listwidget.currentRow())


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

takeItem() methodu parametre olarak verilen ögenin sırasını alır ve bu method çalıştığında sırası belirtilen ögeyi listeden çıkarır. sil() yuvasında ise QListWidget pencere aracında tıklanmış bir öge varsa o ögeyi parametre olarak takeItem() methoduna veriyoruz ve **Tamam** butonuna basılmasıyla seçili ögeyi listeden çıkarıyoruz. Oldukça basit değil mi?

QListWidgetItem

QListWidget'i anlatırken öge olarak hep karakter dizisi ekledik ve bir Python listesi gibi sırasına göre işlem yaptık. PyQt bize QListWidget ile kullanmamız için QListWidgetItem sınıfı sağlar. Bu sınıf sayesinde hem listede görünürde karakter verisi sıralarız, hem de bu sınıf ile üreteceğimiz nesne sayesinde her QListWidgetItem ögesine farklı özellikler kazandırabiliriz.

Bir QListWidgetItem nesnesini şöyle tanımlarız:

```
item = QListWidgetItem("Python", self.listwidget)
```

İstersek ilk parametre olan öge adını QListWidgetItem'in setText() methodu ile tanımlayıp sadece self.listwidget'i parametre olarak verebiliriz:

```
item = QListWidgetItem(self.listwidget)
item.setText("Python")
```

Bir önceki konuda QListWidget'e nasıl öge eklediğimizi öğrendik. Şimdi basit bir örnekle bu sefer karakter dizisi değil de QListWidgetItem nesnesi ekleyelim:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)
        self.listwidget.addItem(QListWidgetItem("Python"))
        self.listwidget.addItem(QListWidgetItem("Ruby"))
        self.listwidget.addItem(QListWidgetItem("Go"))
        self.listwidget.addItem(QListWidgetItem("Perl"))

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)
        self.buton.clicked.connect(self.sil)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırdığınızda bir önceki konuda gördüğünüz aynı pencere ile karşılaşacaksınız. QListWidget sınıfının addItem() methodu parametre olarak QListWidgetItem sınıfını da alır. Burada dikkat edilmesi gereken QListWidgetItem'e argüman olarak sadece öge ismi verdik. İkinci parametreye verdiğimiz ebeveyn parametresini burada kullanmaya gerek duymadık. Çünkü addItem() methodu aldığı QListWidgetItem nesnesine ebeveynini tayin eder. Eğer siz addItem ile QListWidgetItem ögesini QListWidget'e eklemek istemiyorsanız konunun başında kullanımına dair verdiğimiz örneğe göre de ekleme işlemini yapabilirsiniz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.listwidget = QListWidget(self)
        item1 = QListWidgetItem("Python", self.listwidget)
        item2 = QListWidgetItem("Ruby", self.listwidget)
        item3 = QListWidgetItem("Go", self.listwidget)
        item4 = QListWidgetItem("Perl", self.listwidget)

        self.buton = QPushButton(self)
        self.buton.setText("Tamam")

        self.layout.addWidget(self.listwidget)
        self.layout.addWidget(self.buton)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

QListWidget'e bu şekilde öge eklediğinizde seçili olan ögeyi almak için currentRow() methodu yerine currentItem() methodunu kullanmak isteyebilirsiniz. Bu method size öge sırası yerine bir QListWidgetItem nesnesi döndürür. Aynı şekilde seçili olmasını istediğiniz ögenin nesnesini biliyorsanız setCurrentItem() methodunu kullanabilirsiniz.

Aynı şekilde insertItem() methodunu da QListWidgetItem sınıfı ile kullanabilirsiniz. Ayrıca bu sınıfın setIcon() methodu ile her ögenin başına gelecek şekilde bir simge resmi koyabilirsiniz:

```
item1.setIcon(QIcon("falanca.png"))
```

> QIcon sınıfı QtGui paketindedir.

QListWidgetItem sınıfının kullanımını gördükten sonra şimdi tekrar QListWidget pencere aracına dönelim...

QListWidget de sıralanmış ögelere **Shift** tuşuna basılı olarak tıkladığınızda birden fazla seçim yapamadığınızı fark edeceksiniz. Çünkü ön tanımlı seçim modu tekli seçimdir. QListWidget pencere aracı için tanımlı dört adet seçim modu vardır. Ayrıca seçim modu sayarsanız bir de seçilememe modu da mevcuttur.

Bir QListWidget'in öge seçim modunu düzenlemek için setSelectionMode() methodu kullanılır. Kullanımını bir örnek ile görelim:

```
self.listwidget.setSelectionMode(QAbstractItemView.ContiguousSelection)
self.listwidget.setSelectionMode(QAbstractItemView.ExtendedSelection)
self.listwidget.setSelectionMode(QAbstractItemView.SingleSelection)
self.listwidget.setSelectionMode(QAbstractItemView.MultiSelection)
```

QAbstractItemView bir sanal sınıftır. PyQt de buna benzer bir çok sanal sınıf vardır. Bu sınıflar hangi sınıf için yazıldıysa, o sınıfa bazı özelliklerin kazandırılmasını sağlar. Bu örnekte de QListWidget pencere aracına özellik kazandırılıyor.

ContiguousSelection özelliği ile QListWidget de bir ögeye tıklanmış ise ve sonraki ögeye basarken **Shift** tuşu basılıysa, iki öge dahil arasında kalan ögelerde seçilir.

ExtendedSelection özelliği ile QListWidget de bir ögeye **Ctrl** tuşu ile basılırsa seçili ise seçilme kalkar, seçili değilse seçilir. Bu şekilde tıkladığınız her öge ya seçilir, ya seçimi kalkar. **Shift** tuşuna basılı olarak tıkladığınızda ise ilk seçimle ikinci seçim arası seçilir ve **Shift** tuşuna basılıyken farklı bir öge seçerseniz ilk tıkladığınız ile bu seferki tıkladığınız dahil arasında kalan ögeler de seçilir.

SingleSelection özelliği ön tanımlı olarak gelir. Sadece tek seçim yapabilirsiniz.

MultiSelection özelliği ile her tıkladığınız öge seçili değilse seçilir, seçili ise seçilmemiş hale döner. Bu seçim modu seçildiğinde **Shift** ya da **Ctrl** tuşuna basmadan çoklu seçim yapabilirsiniz.

> QAbstractItemView sınıfı QtWidgets paketi içindedir.

Pencere Araçlarının Yerleşimi

Bir PyQt penceresinde yer alan pencere araçlarını yerleştirmek için bir çok alternatif vardır. Şimdiye kadar işlediğimiz konularda pencere aracı yerleşimi olarak bir tek QVBoxLayout sınıfını pek değinmeden kullandık. Ancak pencere aracını, pencerenin istediğimiz her hangi bir yerine yerleştirebiliriz. Bunu sağlayan her pencere aracının setGeometry() methodudur. Bu method tercihinize göre ya QRect sınıfını ya da dört adet integer veriyi parametre alır:

```
self.buton = QPushButton(self)
self.buton.setGeometry(QRect(10, 10, 100, 25))
self.buton.setGeometry(10, 10, 100, 25)
```

> QRect sınıfı QtCore paketindedir.

Gördüğünüz gibi oldukça kolay. Burada ilk parametre pencerenin ne kadar sağında yer alacağını, ikinci parametre ne kadar aşağıda kalacağını belirler. Diğer iki parametre ise pencere aracının boyutunu ayarlar. Tabii isterseniz pencere aracının sadece konumunu da belirtebilirsiniz. daha önceki konularda gördüğümüz move() methodu bu işe yarar. Bu şekilde bir kullanım tercih edilirse pencere aracının boyutları ön tanımlı olarak gelen boyutta pencereye çizilir. Ayrıca boyutunu ayarlamak isterseniz de resize() methodunu kullanabilirsiniz. Bu methodun kullanımını da daha önceki konularda öğrenmiştik.

Bir çok programa baktığımızda; pencere boyutunu fare kullanarak büyültüp küçülttüğümüzde pencere araçlarınında duruma göre konum ve boyutlarının değiştiğini görürüz. Eğer biz uygulamamızı bu şekilde olmasını istiyorsak setGeometry() methodunu kullanmamalıyız. Eğer bu şekilde penceremize pencere araçlarını konumlandırırsak ekranı büyülttüğümüzde ya da küçülttüğümüzde pencere araçlarının olduğu gibi kaldığını görürüz. İyi tasarlanmış ve gelişmiş uygulamalar bu tarz bir tasarımdan kaçınırlar. Ayrıca kullandığımız grafik arayüz kütüphaneleri bunun için gerekli pencere aracı yerleştirme sınıflarını da sağlarlar.

Kullanımı oldukça basit olan bu yerleştirme araçlarını görelim...

QVBoxLayout

Bu yerleştirme aracı eklediğiniz pencere araçlarını dikey olarak yerleştirir:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout(self)

        self.buton1 = QPushButton(self)
        self.buton1.setText("Bir")
        self.buton2 = QPushButton(self)
        self.buton2.setText("İki")
        self.buton3 = QPushButton(self)
        self.buton3.setText("Üç")
        self.buton4 = QPushButton(self)
        self.buton4.setText("Dört")

        self.layout.addWidget(self.buton1)
        self.layout.addWidget(self.buton2)
        self.layout.addWidget(self.buton3)
        self.layout.addWidget(self.buton4)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Örnek kodda dört adet butonu dikey olarak yerleştirdik. Bunu sağlayan QVBoxLayout sınıfının addWidget() methodudur. Gördüğünüz gibi oldukça basit bir kullanımı var.

Biz yukarıdaki örnekte QVBoxLayout sınıfının ilk parametresini **self** yaparak ebeveyninin QWidget olduğunu belirttik. Bunun yerine QWidget'in setLayout() methodu ile QVBoxLayout sınıfına hiç parametre girmeden de tanıtabiliriz. Ayrıca bu method ile self.layout ile örneğini aldığımız QVBoxLayout yerleştirme aracını da QWidget'e bu pencerenin yerleştirme yöneticisi olduğunu belirtmiş oluyoruz:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QVBoxLayout()

        self.buton1 = QPushButton(self)
        self.buton1.setText("Bir")
        self.buton2 = QPushButton(self)
        self.buton2.setText("İki")
        self.buton3 = QPushButton(self)
        self.buton3.setText("Üç")
        self.buton4 = QPushButton(self)
        self.buton4.setText("Dört")

        self.layout.addWidget(self.buton1)
        self.layout.addWidget(self.buton2)
        self.layout.addWidget(self.buton3)
        self.layout.addWidget(self.buton4)

        self.setLayout(self.layout)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

QHBoxLayout

Bu yerleştirme aracı ise eklenen pencere araçlarını yatay olarak yerleştirir. Yukarıdaki koddan örnek olarak verelim:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QHBoxLayout()

        self.buton1 = QPushButton(self)
        self.buton1.setText("Bir")
        self.buton2 = QPushButton(self)
        self.buton2.setText("İki")
        self.buton3 = QPushButton(self)
        self.buton3.setText("Üç")
        self.buton4 = QPushButton(self)
        self.buton4.setText("Dört")

        self.layout.addWidget(self.buton1)
        self.layout.addWidget(self.buton2)
        self.layout.addWidget(self.buton3)
        self.layout.addWidget(self.buton4)

        self.setLayout(self.layout)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Gördüğünüz gibi sadece sınıf adını değiştirdik ve pencere araçları yatay olarak yerleştirildi.

QFormLayout

QFormLayout, adından da anlaşılacağı gibi formlarınız için yerleştirme işini yapar. Mesela üye kayıt formu tasarladınız: Bu tasarladığınız formda kullandığınız pencere araçlarını nasıl uygun şekilde yerleştireceksiniz? Yan yana iki adet QVBoxLayout ile mi? Tabii ki hayır! Tabii isterseniz QGridLayout ile de yapabilirsiniz, ama pencere aracı ekleme işi biraz daha zahmetli gelebilir.

QFormLayout sadece iki sütun alacak şekilde ayarlanmıştır. Yani formunuzda yer alabilecek **İsim:** yazan etiket ile sağındaki QLineEdit pencere aracı iki sütunu dolduracaktır. Her satır için iki sütun. Basit bir form ile QFormLayout sınıfını görelim:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QFormLayout(self)

        self.lineedit1 = QLineEdit(self)
        self.lineedit2 = QLineEdit(self)
        self.lineedit3 = QLineEdit(self)
        self.lineedit3.setEchoMode(QLineEdit.Password)

        self.layout.addRow("İsim:", self.lineedit1)
        self.layout.addRow("Soyad:", self.lineedit2)
        self.layout.addRow("Şifre:", self.lineedit3)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Gördüğünüz gibi diğer pencere aracı yerleştirme sınıfları gibi oldukça basit bir kullanımı var. QFormLayout'un addRow() methodu ilki etiket adı, ikincisi pencere aracı olmak üzere iki tane parametre alır. Her kullandığınız addRow() methodu formunuza bir satır daha ekler.

Pencere aracı yerleşimi için QFormLayout bize başka bir seçenek daha sunar. Bu seçeneği siz daha çok **Qt Designer** uygulaması ile QFormLayout pencere aracı yerleştiricisini kullandığınızda göreceksiniz. Çünkü **.ui** uzantılı tasarım dosyasını Python betiğine dönüştürdüğünüzde bu kullanım çeşidiyle karşılaşacaksınız.

Şimdi diğer seçeneğimizi görelim:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.layout = QFormLayout(self)

        self.lineedit1 = QLineEdit(self)
        self.lineedit2 = QLineEdit(self)
        self.lineedit3 = QLineEdit(self)
        self.lineedit3.setEchoMode(QLineEdit.Password)

        self.label1 = QLabel(self)
        self.label1.setText("İsim:")
        self.label2 = QLabel(self)
        self.label2.setText("Soyad:")
        self.label3 = QLabel(self)
        self.label3.setText("Şifre:")

        self.layout.setWidget(0, QFormLayout.LabelRole, self.label1)
        self.layout.setWidget(0, QFormLayout.FieldRole, self.lineedit1)
        self.layout.setWidget(1, QFormLayout.LabelRole, self.label2)
        self.layout.setWidget(1, QFormLayout.FieldRole, self.lineedit2)
        self.layout.setWidget(2, QFormLayout.LabelRole, self.label3)
        self.layout.setWidget(2, QFormLayout.FieldRole, self.lineedit3)


uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Kodumuz oldukça uzadı değil mi?..

QFormLayout sınıfının setWidget() methodu bir diğer seçenektir. Az önce dediğimiz gibi bu kullanım tarzını **Qt Designer** uygulaması ile tasarım yaptığınızda karşılaşacaksınız.

setWidget() methodunun aldığı ilk parametre sizin de tahmin edeceğiniz gibi form ögelerinin sırasını belirtir.

İkinci parametrede bulunan QFormLayout sınıfının LabelRole özelliği, eklenen pencere aracının etiket özelliğinde olduğunu belirtir. Zaten son parametrelere verdiğimiz QLabel nesnelerinden de anlaşılıyordur.

FieldRole özelliği ise Türkçe anlamındaki gibi üçüncü parametreye atadığımız pencere aracının rolünü belirtiyor. QLineEdit pencere araçları formun alanında yer alırlar.

Form yerleşimi için sunulan iki seçeneği de gördünüz. Siz QFormLayout sınıfını kullanırsanız, sizin için hangisi uygunsa onu kullanacaksınız.

QGridLayout

Grid, Türkçe de ızgara anlamına gelir. QGridLayout'a pencere aracı yerleşiminde de mantık ızgara üzerine et yerleştirmek gibidir. İster köfte gibi tek bir noktayı işgal etsin, ister pirzola gibi birden fazla köftenin işgal ettiği yeri işgal etsin. Yerleştireceğiniz pencere araçlarını isteğinize göre tek satır ve sütuna koyabileceğiniz gibi birden fazla sütuna ve/veya satıra sığacak şekilde de koyabilirsiniz. Örnek olarak şöyle bir penceremiz olduğunu düşünelim:

Kırmızı çerçeveli alan bizim ızgaramız oluyor. Bu ızgarada birer adet QTextEdit, QPushButton, QLineEdit ve QLabel var. Dikkatli bakarsanız her pencere aracını birbirinden ayıran yeşil çizgiler var. Bu yeşil çizgilerden anlayacağımız şey, ızgaramızın üç satırdan ve iki sütundan oluştuğudur. QTextEdit sıfırıncı satırda bulunur ve iki sütunu da kaplar. Aynı şekilde QLabelde ikinci satırda bulunur ve iki sütunu da kaplar. QPushButton ve QLineEdit ise birinci satırda birer sütun işgal ederler.

> Python'un saymaya sıfırdan başladığını unutmayalım.

Dikkatinizi sütunlardaki eşitsizlik çekmiş olabilir. Bunun sebebi de pencere aracı olarak yerleştirdiklerimizin kapladıkları alana göre şekil almasıdır.

Şimdi de kod olarak nasıl olduğunu görelim:

```
gridLayout = QGridLayout(self)

gridLayout.addWidget(textEdit, 0, 0, 1, 2)
gridLayout.addWidget(pushButton, 1, 1, 1, 1)
gridLayout.addWidget(lineEdit, 1, 0, 1, 1)
gridLayout.addWidget(label, 2, 0, 1, 2)
```

Diğer yerleşim sınıflarında olduğu gibi QGridLayout sınıfının da addWidget() methodunu pencere aracı yerleştirmek için kullanırız. Yalnız diğer methodlardan farkı aldığı parametrelerdir. Hatırlayın!.. QTextEdit sıfırıncı satırda birinci ve ikinci sütunu kaplıyordu. Yukarıdaki kodda göreceğiniz gibi ilk iki sıfır rakamı sıfırıncı satır ve sıfırıncı sütunu temsil ediyor. Sonraki "1" rakamı kaplayacağı satır sayısını, "2" rakamı ise kaplayacağı sütun sayısını temsil ediyor.

Aynı şekilde QLabel de ikinci satırın sıfırıncı sütununda bulunuyor ve tek satır ile iki sütunu kaplıyor.

QLineEdit birinci satırda ve sıfırıncı sütunda bulunup, tek satır ve tek sütunluk bir yer kaplıyor. QPushButton ise birinci satır ve sütunda yer alıp QLineEdit gibi tek satır ve sütunluk yer kaplıyor.

Anlamakta güçlük çekiyor olabilirsiniz. Ben bile ızgara yerleşimini kullanırken kafam karışabiliyor. Siz de bu sınıfı ne kadar sık kullanır ve yanılıp hatanızı düzeltirseniz o kadar iyi kavrarsınız.

QGridLayout, diğer sınıfların aksine daha fazla method barındırır. Yalnız fazla bilgiyle kafa karışıklığı yaratmamak için kullanacağımız en sık methodu anlattık.

Bunun yanı sıra bu konu altında anlatılan her yerleşim sınıfında olan bir methodu da açıklayalım:

Bir zaman gelir ve birden fazla yerleşim sınıfını başka bir yerleşim sınıfı altında birleştirmek isteyebilirsiniz. Nasıl bir pencere aracını addWidget() ile ekliyorsak bir yerleşim sınıfını da bu şekilde ekleyebiliriz. Tabii bu işi addWidget() methodu yerine addLayout() methodu ile yaparız.

addLayout() methodu her sınıfın addWidget() methodu ile aynı işlevselliği barındırır. Tek fark addLayout ile bir yerleşim sınıfı ekliyoruz. Küçük bir örnek ile konuyu kapatalım:

```
from PyQt5.QtWidgets import *
import sys

class Pencere(QWidget):
    def __init__(self):
        super().__init__()
        self.vlayout = QVBoxLayout(self)

        self.hlayout = QHBoxLayout(self)
        buton1 = QPushButton("Sol üst", self)
        buton2 = QPushButton("Sağ üst", self)
        self.hlayout.addWidget(buton1)
        self.hlayout.addWidget(buton2)

        self.hlayout2 = QHBoxLayout(self)
        buton3 = QPushButton("Sol alt", self)
        buton4 = QPushButton("Sağ alt", self)
        self.hlayout2.addWidget(buton3)
        self.hlayout2.addWidget(buton4)

        self.vlayout.addLayout(self.hlayout)
        self.vlayout.addLayout(self.hlayout2)

uygulama = QApplication(sys.argv)
pencere = Pencere()
pencere.show()
uygulama.exec_()
```

Ana Pencere Araçları

Ana pencere aracı olan QMainWindow penceresi bu konuda öğrendiğimiz gibi sadece kendi içerisine ekleyebileceğimiz pencere araçlarını kullanmamıza olanak sağlar. Bu bölümde, bu pencere araçlarının kullanımını öğreneceğiz.

QMenuBar

QMenuBar ile menülerden oluşan bir menü çubuğu oluştururuz. Menüler QMenu sınıfı ile oluşturulur. Menülerde bulunan ögeler de Aksiyon adı verilen QAction sınıfı ile oluşturulur. İsterseniz öncelikle QMenuBar pencere aracımızı, penceremize yerleştirelim:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menuBar = QMenuBar(self)
        self.setMenuBar(self.menuBar)


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

En sade şekliyle bir QMainWindow'a QMenuBar bu şekilde eklenir. Unutmayın ki; QMainWindow, özelliği nedeniyle kendi içerisinde bazı pencere araçları için yer ayrılmış bir pencere aracıdır. Bir QMainWindow'a QPushButton gibi pencere araçları eklemek istiyorsanız öncelikle bir QWidget pencere aracını tanımlayıp setCentralWidget() methodu ile QMainWindow'a tanıtmalısınız. Aynı Şekilde QMenuBar da QMainWindow'un üst kısmı için ayarlandığı için eklediğimiz QMenuBar üst kısımda görülecektir. Görüldüğü üzere bu pencere aracını da QMainWindow'un setMenuBar() methodu ile tanıttık. Tabii bu haliyle menü çubuğunu görmek mümkün değil. Bu yüzden menu çubuğuna QMenu ya da QAction eklememiz gerekir.

QMenu

Menü çubuğuna bir menü eklemek için yapmamız gereken şey çok basit:

```
self.menu = QMenu(self.menuBar)
self.menu.setTitle("Dosya")
self.menuBar.addMenu(self.menu)
```

self.menu adında bir QMenu nesnesi tanımladık. Menünün adını setTitle() methodu ile "Dosya" olarak belirttik ve QMenuBar'ın addMenu() methodu ile menü çubuğumuza menümüzü ekledik. addMenu() methoduna isterseniz QMenu'yu isterseniz karakter dizisini argüman olarak girerek menü ekleyebilirsiniz:

```
self.menuBar.addMenu(self.menu)
#ya da
self.menuBar.addMenu("Dosya")
```

Tabii kullanım amacınıza göre menülerin başına simge resmi ekleyebilirsiniz:

```
self.menuBar.addMenu(QIcon("simge.png"), "Dosya")
```

Alternatif olarak da QMenu sınıfının setIcon() methodunu kullanabiliriz:

```
self.menu.setIcon(QIcon("simge.png"))
```

QMenu aracında eklediğimiz QAction ögelerinin arasına çizgi ekleyip bir QAction grubunu diğer bir gruptan ayırmak istersek de QMenu sınıfının addSeperator() methodunu gerekli yere yazmamız yeterlidir:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menuBar = QMenuBar(self)
        self.setMenuBar(self.menuBar)

        self.menu = QMenu(self.menuBar)
        self.menu.setTitle("Dosya")
        self.menuBar.addMenu(self.menu)
        self.action1 = QAction(self.menu)
        self.action1.setText("Aç")
        self.menu.addAction(self.action1)
        self.menu.addSeparator()
        self.action2 = QAction(self.menu)
        self.action2.setText("Kaydet")
        self.menu.addAction(self.action2)
        self.action3 = QAction(self.menu)
        self.action3.setText("Farklı Kaydet")
        self.menu.addAction(self.action3)

uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Örnek kodu çalıştırdığınızda "Dosya" menüsünde bulunan "Aç" adlı QAction ögesinden sonra diğer QActionlardan ayıran bir çizgi eklendiğini göreceksiniz.

QAction

QAction pencere araçlarına eklenebilen soyut bir kullanıcı arabirimi eylemi sağlar. Uygulamalarda menülerde, araç çubuğunda kullanılabilir ve klavye kısayolları ile çağırılabilirler.

Kullanımı oldukça basittir... Bir QAction nesnesini üç farklı şekilde tanımlayabilirsiniz:

```
self.action = QAction(self)
self.action = QAction("Aç", self)
self.action = QAction(QIcon("simge.png"), "Aç", self)
```

İsterseniz ilk seçenekteki gibi nesneyi yaratıp adı için setText() methodundan, ikon resmi için ise setIcon() methodundan faydalanabilirsiniz ya da diğer seçeneklerdeki gibi bir tanımlama yaparak fazladan method kullanmaktan kaçınabilirsiniz.

Genelde QAction nesnesi eklendiği menü örneğini argüman alır. Yani:

```
self.menu = QMenu(self.menuBar)
self.action = QAction(self.menu)
```

Ayrıca bir QAction'ı QCheckBox gibi kullanabilmeniz için setCheckable() methodu barındırır. Bu methoda **True** değeri girilirse QAction nesnesinin eklendiği menü de isminin baş kısmında kutucuk belirir.

Bir QAction nesnesini bir menüye, menünün addAction() methodu ile ekleyebileceğiniz gibi QAction sınıfında bulunan setMenu() methodu ile bu methoda argüman olarak gireceğiniz QMenu örneği ile QAction nesneniz ilgili menüye dahil olacaktır.

Her işletim sisteminde ya da Linux dağıtımında çalışmasa da bir QAction'a kısayol atayarak, bu kısayol sayesinde QAction'a tıklanma olayı gerçekleştirilebilir:

```
self.action.setShortcut("Ctrl+O")
```

Görüldüğü gibi karakter dizisi ile kısayol atamasını yaptık. Artık uygulamanız açıkken Ctrl + O tuşuna bastığınızda **Aç** isimli QAction'a fare ile tıklanmış gibi işlem yapılacak. Yalnız Ubuntu üzerinden bu kısayol olayı çalışmıyor...

Şimdi basit bir örnekle kısayol olayını pekiştirelim:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menuBar = QMenuBar(self)
        self.setMenuBar(self.menuBar)

        self.menu = QMenu(self.menuBar)
        self.menu.setTitle("Dosya")
        self.menuBar.addMenu(self.menu)
        self.action = QAction(self.menu)
        self.action.setText("Aç")
        self.action.setShortcut("Ctrl+O")
        self.menu.addAction(self.action)

        self.action.triggered.connect(self.dialogAc)

    def dialogAc(self):
        dosya = QFileDialog.getOpenFileName(self, "Dosya Dialogu", "/home", "Text dosy
ası (*.txt)")

uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

QAction arayüzüne tıklandığında ya da kısayol atadıysak ve bu kısayol tuşlarına bastiysak triggered() sinyali yayılır. Örnekte de görüldüğü gibi bu sinyali dialogAc() methoduna bağladık. Eğer bu kodu kısayolu çalıştıran bir işletim sistemi ya da masaüstü ortamında çalıştırıp denerseniz QAction arayüzüne tıkladığınızda ya da kısayol tuşlarına bastığınız da bir dosya seçim penceresi açılacaktır.

QAction'ın triggered sinyali ile bu arayüze tıklanıldığında yapılmasını istediğiniz her türlü işlemi halledebilirsiniz. Oldukça basit bir kullanımı var değil mi?

QToolBar

Bu pencere aracı adından da anlaşılacağı üzere araç çubuğu oluşturmamıza yarar. Ofis programramlarında gördüğünüz "Kaydet", "Aç", "Yazdır", gibi butonların bulunduğu şerit şeklindeki alan PyQt5 de QToolBar ile yapılır.

QToolBar pencere aracına isterseniz QAction, QMenu ya da diğer pencere araçlarını ekleyebilirsiniz:

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.toolBar = QToolBar(self)
        self.addToolBar(Qt.TopToolBarArea, self.toolBar)

        self.pushButton1 = QPushButton()
        self.pushButton1.setText("Aç")
        self.toolBar.addWidget(self.pushButton1)

        self.toolBar.addSeparator()
        self.pushButton2 = QPushButton()
        self.pushButton2.setText("Kaydet")
        self.toolBar.addWidget(self.pushButton2)

        self.pushButton3 = QPushButton()
        self.pushButton3.setText("Farklı Kaydet")
        self.toolBar.addWidget(self.pushButton3)

uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Bu örnekte QToolBar'ın addWidget() methodu ile üç adet QPushButton pencere aracı ekledik. **Aç** butonundan sonraki butonları ayrı tutmak için QToolBar'ın addSeparator() methodunu kullandık. Bu method QMenu de olduğu gibi kullanıldığı yerden önceki ve sonraki pencere araçları arasına çizgi çeker.

Bir QToolBar'ı QMainWindow'a yerleştirmek için QMainWindow'un addToolBar() methodunu kullanılır. Bu methodun ilk parametresi QToolBar'ın pencere üzerindeki konumunu(sağ-sol-yukarı-aşağı) belirtir, ikinci parametreye ise QToolBar sınıfının örneği yazılır. Yukarıdaki örnek kodda biz QToolBar'ı pencerenin üst kısmına yerleştirdik. Eğer farklı bir konuma yerleştirmek isterseniz:

```
Qt.LeftToolBarArea
Qt.RightToolBarArea
Qt.BottomToolBarArea
```

> Qt sınıfı QtCore paketinde bulunur.

Genelde QToolBar pencere aracında QPushButton yerine QAction kullanılır. QMenu de tanımlanan bu sınıfları fazladan QAction oluşturmadan QToolBar'da da kullanabiliriz. Bir QAction sınıfı QToolBar'ın addAction() methodu ile eklenir:

```
self.action = QAction()
self.action.setText("Aç")
self.toolBar.addAction(self.action)
```

Yukarıda QToolBar pencere aracının hangi konuma yerleştirileceğini öğrendik. Varsayılan olarak bir QToolBar pencere aracı fare sürüklemesi ile farklı konumlara taşınabildiği gibi ayrı çerçevesiz bir pencerede duracak şekilde ayarlıdır. QToolBar'ın taşınabilmesine olanak sağlayan bu sınıfın setMovable() methodudur. Bu methodun varsayılan parametresi **True** dur. Eğer siz bu methoda **False** parametresini verirseniz QToolBar pencere aracı taşınamaz olacaktır. Eğer taşınabilir olmasında bir sakınca yoksa, ama uygun yere koyulmama durumunda ayrı bir pencere gibi durmasını engellemek istiyorsak setFloatable() methodunu kullanacağız. Bu method da varsayılan olarak **True** dur ve siz bu methoda **False** parametresi girerseniz QToolBar pencere aracı sadece QMainWindow içerisinde barınabilecektir.

QStatusBar

QStatusBar, QMainWindow penceresinin en altında yatay şekilde konumlanan bir pencere aracıdır. Bu pencere aracı ile geçici bilgi mesajlarının yanı sıra durum çubuğunun bir kısmını kaplayan pencere araçları içerebilir. Bunlar isteğe göre gizlenebilecek şekilde ayarlanabildiği gibi gizlenemeyen, kalıcı durum mesajları da koyulabilir.

Bir QStatusBar sınıfını QMainWindow'a tanıtmak için bu sınıfın setStatusBar() methodu kullanılır:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)
        self.statusBar.showMessage("Merhaba!", 3000)

uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırdığınızda basit bir pencere açılacaktır. Alt kısmında belli-belirsiz bir durum çubuğu olacak ve **Merhaba!** yazısı bu durum çubuğunda 3 saniye kadar gözükecektir. Evet, QStatusBar'ın showMessage() methodu ile durum çubuğumuzda mesaj gösteriyoruz. Gösterilecek süreyi de milisaniye cinsinden giriyoruz. Tabii süre belirtmek zorunda değilsiniz.

Geliştirilen yazılımlarda fare ile bir pencere aracının üzerine gelindiğinde durum çubuğunda geliştiricilerin istediği bir mesaj görünür. Biz de bunu yapmak istersek eğer çok basit bir yolu var. Böylece fazladan kod yazmamız veya takla atmamız gerekmez:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.buton = QPushButton(self.widget)
        self.buton.setText("Buton")
        self.buton.setStatusTip("Fare butonun üzerinde.")

        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)

uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Hemen her pencere aracında bulunan setStatusTip() methodu ile QMainWindow'a tanıttığımız QStatusBar pencere aracımıza bilgi mesajı gösterebiliyoruz. Kullanıcı ilgili pencere aracına fare imlecini üstünde tuttuğu sürece gösterilecek bir mesaj belirtmiş olduk.

Konunun başında gizlenebilen ya da kalıcı pencere araçlarını da durum çubuğunda konumlandırabileceğimizi söylemiştik. Eğer bir pencere aracının sabit olarak durum çubuğunda kalmasını istiyorsak addParmanentWidget() methodunu kullanırız. Eğer durum mesajı geldiğinde var olan pencere aracının bu mesajı engellemesini istemiyorsak addWidget() methodunu kullanırız.

addWidget() ile eklenen pencere araçları durum çubuğunun solunda konumlanır. addParmanentWidget() ile eklenen pencere araçları sağ tarafa yaslanacak şekilde konumlanır. Bir örnek ile pekiştirelim:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.buton = QPushButton()
        self.buton.setText("Buton")
        self.buton.setStatusTip("Fare butonun üzerinde.")

        self.buton2 = QPushButton()
        self.buton2.setText("Buton2")

        self.statusBar = QStatusBar(self)
        self.setStatusBar(self.statusBar)

        self.statusBar.addWidget(self.buton2)
        self.statusBar.addPermanentWidget(self.buton)

uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırdığınızda iki adet butonun durum çubuğunda göründüğünü göreceksiniz. Eğer sağdaki butona fare imlecini getirirseniz durum mesajının solda göründüğünü ve ikinci butonun gizlendiğini göreceksiniz. Durum mesajı ne kadar uzun olursa olsun, sağdaki buton hep görülür olarak kalacaktır.

addWidget() ve addParmanentWidget() methodları ikinci bir parametre daha alır. Bu parametreler varsayılan olarak 0(sıfır)'dır. Eğer bu parametreyi 1 olarak girersek eklediğimiz pencere aracı durum çubuğunu kaplayacaktır. Tabii başka pencere araçları var ise onların minimum boyutlara düşürüp kalan boşluğu kendine alacaktır.

Mesaj Kutuları

Mesaj Kutuları, kullanıcıyı bilgilendiren veya kullanıcıya bir soru soran ve cevap almak için kalıcı bir iletişim kutusu sağlar. Bu pencereleri QMessageBox sınıfı ile oluştururuz.

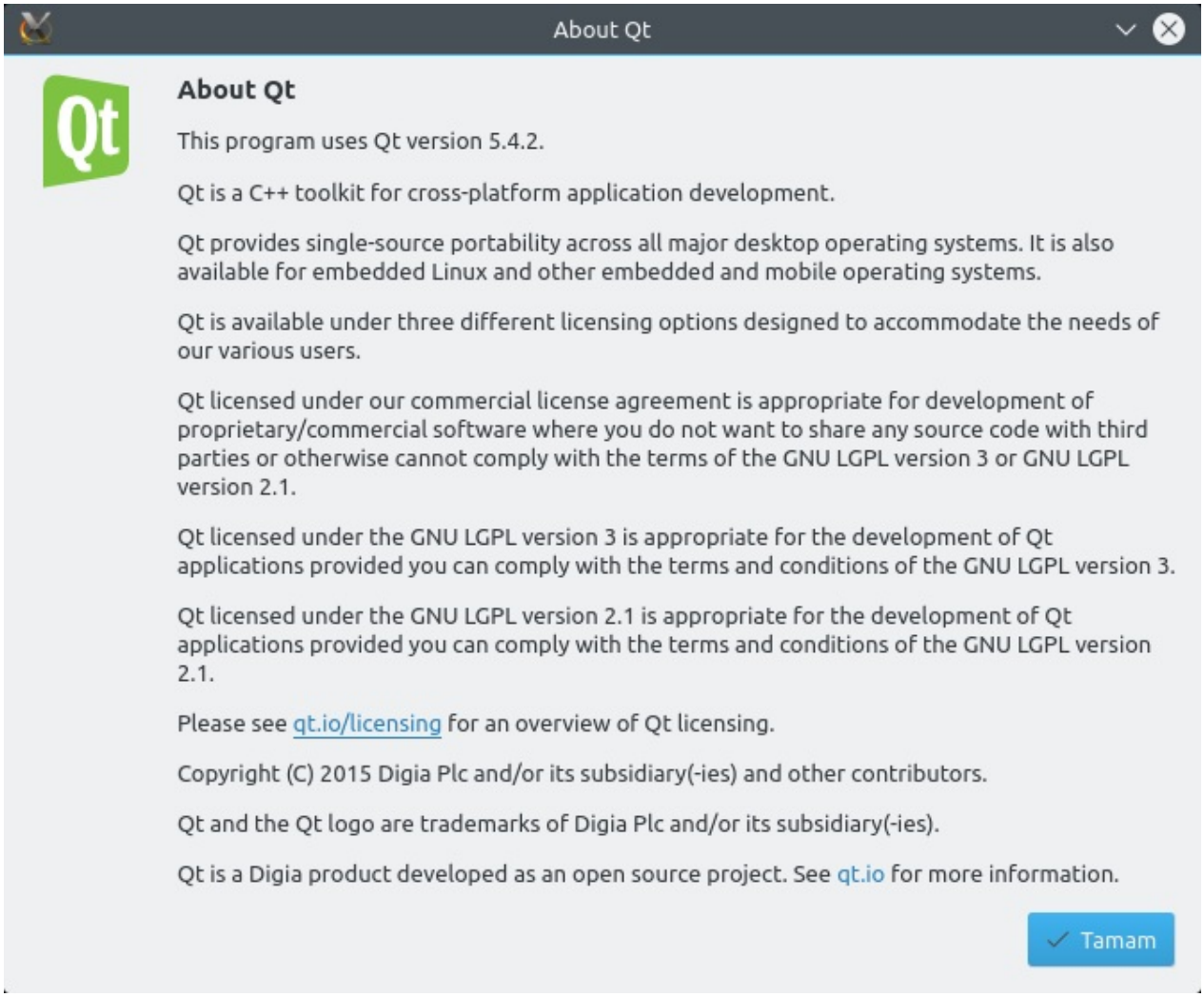
Bir mesaj kutusu duruma göre kullanıcıyı uyarmak için bir metin görüntüler. Bilgilendirici metin, her hangi bir uyarıyı açıklar, kullanıcıya soru sorar veya kullanıcı isterse daha detaylı bir açıklamayı görüntülemek için işlev sunar. Bir mesaj kutusu aynı zamanda özelliğine göre bir simge ve butonlar barındırır.

QMessageBox kullanıcıya mesaj kutusu göstermek için hali hazırda altı adet statik method sunar. Tabii biz yazdığımız programa göre kendi özelleştirilmiş mesaj kutularımızı da kodlayabileceğiz...

Hazır olarak sunulan mesaj kutularından birisi Qt hakkındadır. Qt ile yazılmış uygulamaların **Hakkında** menüsünde **Qt Hakkında** diye kesin bir aksiyonu bulunur. Buna tıkladığımızda ise Qt hakkında geniş bir bilgi içeren mesaj kutusu açılır. Bunu biz QMessageBox'un aboutQt() statik methoduyla yaparız. Kısa bir örnek ile bu kısmı geçelim ve daha çok kullanacağımız mesaj kutularına bakalım:

```
from PyQt5.QtWidgets import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menubar = QMenuBar(self)
        self.setMenuBar(self.menubar)

        self.hakkindaMenusu = QMenu(self)
        self.hakkindaMenusu.setTitle("Hakkında")
        self.menubar.addMenu(self.hakkindaMenusu)

        self.qtHakkindaAksiyonu = QAction(self)
        self.qtHakkindaAksiyonu.setText("Qt Hakkında")
        self.hakkindaMenusu.addAction(self.qtHakkindaAksiyonu)

        self.qtHakkindaAksiyonu.triggered.connect(self.qtHakkindaKutusuGoster)

    def qtHakkindaKutusuGoster(self):
        QMessageBox.aboutQt(self)


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Daha önceki konularda gördüğümüz sınıfları örneğimiz gereği bu kodda tekrar işledik. Kodu çalıştırdığınızda küçük bir pencerede, en üstte **Hakkında** adında bir menü göreceksiniz. Ona tıklayıp **Qt Hakkında** aksiyonuna tıkladığınızda ise karşınıza Qt hakkında bilgi veren bir mesaj kutusu göreceksiniz:



İlgili kod satırı oldukça basit. Gerekli sınıfı ve statik methosu yazıyoruz ve methoda parametre olarak ana pencereyi belirten bir **self** yazıyoruz.

Hakkında Kutusu

Hazır olarak gelen mesaj kutularından olan hakkında kutusu QMessageBox sınıfının about() statik methoduyla oluşturulur. about() methoduna parametre olarak aboutQt()'ye ek olarak başlık ve açıklayıcı metin girilir.

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menubar = QMenuBar(self)
        self.setMenuBar(self.menubar)

        self.hakkindaMenusu = QMenu(self)
        self.hakkindaMenusu.setTitle("Hakkında")
        self.menubar.addMenu(self.hakkindaMenusu)

        self.hakkindaAksiyonu = QAction(self)
        self.hakkindaAksiyonu.setText("Hakkında")
        self.hakkindaMenusu.addAction(self.hakkindaAksiyonu)

        self.hakkindaAksiyonu.triggered.connect(self.hakkindaKutusuGoster)

    def hakkindaKutusuGoster(self):
        QMessageBox.about(self, "Başlık", "Programımız hakkında bilgi içeren mesaj kut
usu.")


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırıp denediğinizde "Başlık" adında başlığa sahip bir mesaj kutusu açılacaktır. Çok basit ve açıklayıcı metin uzunluğuna göre yer kaplayan bir arayüz göreceksiniz. Görünüş size kötü geliyorsa programınıza bir pencere ikonu ekleyerek hakkında mesaj kutusunda görülmesini de sağlarsınız

```
self.setWindowIcon(QIcon("logo.png"))
```

Ana penceremize ekleyeceğimiz ikon ile daha naif bir hakkında mesaj kutusu elde edebilirsiniz.

Bilgi Mesajı Kutusu

Hazır olarak gelen mesaj kutularından olan bilgi mesajı kutusu QMessageBox sınıfının information() statik methoduyla oluşturulur. information() methoduna parametre olarak about() daki ile aynı parametreler girilir.

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menubar = QMenuBar(self)
        self.setMenuBar(self.menubar)

        self.bilgiMenusu = QMenu(self)
        self.bilgiMenusu.setTitle("Bilgi")
        self.menubar.addMenu(self.bilgiMenusu)

        self.bilgiAksiyonu = QAction(self)
        self.bilgiAksiyonu.setText("Bilgi")
        self.bilgiMenusu.addAction(self.bilgiAksiyonu)

        self.bilgiAksiyonu.triggered.connect(self.bilgiKutusuGoster)

    def bilgiKutusuGoster(self):
        QMessageBox.information(self, "Başlık", "Kullanıcı için bilgi içeren mesaj kut
usu.")


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırıp denediğinizde "Başlık" adında başlığa sahip bir mesaj kutusu açılacaktır. Çok basit ve açıklayıcı metin uzunluğuna göre yer kaplayan bir arayüz göreceksiniz. about()'un aksie information() ek parametre alarak mesaj kutunuzda arzunuza göre ek butonlar göstermenize olanak tanır. Bu adresden edineceğiniz bilgiyle hangi butonları koyabileceğinizi öğrenebilirsiniz. Python da yazarken de :: karakterini nokta (.) ile

değiştirmeniz gerekiyor. NoButton ifadesi herhangi bir buton göstermemesi gerekirken benim bilgisayarımda **Tamam** butonu yine de görünmektedir. Ekleyeceğiniz birden fazla buton varsa her birini | karakteriyle ayırmalısınız. Örnek:

```
QMessageBox.information(self, "Başlık", "Kullanıcı için bilgi içeren mesaj kutusu.",
                  QMessageBox.Close|QMessageBox.Help)
```

Uyarı Mesajı Kutusu

Hazır olarak gelen mesaj kutularından olan uyarı mesajı kutusu QMessageBox sınıfının warning() statik methoduyla oluşturulur. information() ile aynı parametreler girilir.

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menubar = QMenuBar(self)
        self.setMenuBar(self.menubar)

        self.uyariMenusu = QMenu(self)
        self.uyariMenusu.setTitle("Uyarı")
        self.menubar.addMenu(self.uyariMenusu)

        self.uyariAksiyonu = QAction(self)
        self.uyariAksiyonu.setText("Uyarı")
        self.uyariMenusu.addAction(self.uyariAksiyonu)

        self.uyariAksiyonu.triggered.connect(self.uyariKutusuGoster)

    def uyariKutusuGoster(self):
        QMessageBox.warning(self, "Başlık", "Kullanıcı için uyarı içeren mesaj kutusu."
)


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırıp denediğinizde "Başlık" adında başlığa sahip bir uyarı mesaj kutusu açılacaktır. Çok basit ve açıklayıcı metin uzunluğuna göre yer kaplayan bir arayüz göreceksiniz. information() da olduğu gibi warning() de ek parametre alarak mesaj kutunuzda arzunuza göre ek butonlar göstermenize olanak tanır. Bu adresden edineceğiniz bilgiyle hangi butonları koyabileceğinizi öğrenebilirsiniz. Python da yazarken de :: karakterini

nokta (.) ile değiştirmeniz gerekiyor. NoButton ifadesi herhangi bir buton göstermemesi gerekirken benim bilgisayarımda **Tamam** butonu yine de görünmektedir. Ekleyeceğiniz birden fazla buton varsa her birini | karakteriyle ayırmalısınız. Örnek:

```
QMessageBox.warning(self, "Başlık", "Kullanıcı için uyarı içeren mesaj kutusu.",
                  QMessageBox.Close|QMessageBox.Help)
```

Kritik Hata Mesajı Kutusu

Hazır olarak gelen mesaj kutularından olan kritik hata mesajı kutusu QMessageBox sınıfının critical() statik methoduyla oluşturulur. warning() ile aynı parametreler girilir.

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menubar = QMenuBar(self)
        self.setMenuBar(self.menubar)

        self.kritikMenusu = QMenu(self)
        self.kritikMenusu.setTitle("Kritik")
        self.menubar.addMenu(self.kritikMenusu)

        self.kritikAksiyonu = QAction(self)
        self.kritikAksiyonu.setText("Kritik Hata")
        self.kritikMenusu.addAction(self.kritikAksiyonu)

        self.kritikAksiyonu.triggered.connect(self.kritikKutusuGoster)

    def kritikKutusuGoster(self):
        QMessageBox.critical(self, "Başlık", "Kullanıcı için kritik hata içeren mesaj
kutusu.")


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırıp denediğinizde "Başlık" adında başlığa sahip bir kritik hata mesaj kutusu açılacaktır. Çok basit ve açıklayıcı metin uzunluğuna göre yer kaplayan bir arayüz göreceksiniz. warning() de ek parametre alarak mesaj kutunuzda arzunuza göre ek butonlar göstermenize olanak tanır. Bu adresden edineceğiniz bilgiyle hangi butonları koyabileceğinizi öğrenebilirsiniz. Python da yazarken de :: karakterini nokta (.) ile

değiştirmeniz gerekiyor. NoButton ifadesi herhangi bir buton göstermemesi gerekirken benim bilgisayarımda **Tamam** butonu yine de görünmektedir. Ekleyeceğiniz birden fazla buton varsa her birini | karakteriyle ayırmalısınız. Örnek:

```
QMessageBox.critical(self, "Başlık", "Kullanıcı için kritik hata içeren mesaj kutusu."
,
                  QMessageBox.Close|QMessageBox.Help)
```

Sorun Mesajı Kutusu

Hazır olarak gelen mesaj kutularından olan sorun mesajı kutusu QMessageBox sınıfının question() statik methoduyla oluşturulur. critical() ile aynı parametreler girilir.

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class AnaPencere(QMainWindow):
    def __init__(self):
        super().__init__()
        self.widget = QWidget(self)
        self.setCentralWidget(self.widget)

        self.menubar = QMenuBar(self)
        self.setMenuBar(self.menubar)

        self.sorunMenusu = QMenu(self)
        self.sorunMenusu.setTitle("Sorun")
        self.menubar.addMenu(self.sorunMenusu)

        self.sorunAksiyonu = QAction(self)
        self.sorunAksiyonu.setText("Sorun")
        self.sorunMenusu.addAction(self.sorunAksiyonu)

        self.sorunAksiyonu.triggered.connect(self.sorunKutusuGoster)

    def sorunKutusuGoster(self):
        QMessageBox.question(self, "Başlık", "Kullanıcı için sorun mesajı içeren mesaj
 kutusu.")


uygulama = QApplication(sys.argv)
pencere = AnaPencere()
pencere.show()
uygulama.exec_()
```

Kodu çalıştırıp denediğinizde "Başlık" adında başlığa sahip bir sorun mesajı, mesaj kutusu açılacaktır. Çok basit ve açıklayıcı metin uzunluğuna göre yer kaplayan bir arayüz göreceksiniz. Bu adresden edineceğiniz bilgiyle hangi butonları koyabileceğinizi öğrenebilirsiniz. Python da yazarken de :: karakterini nokta (.) ile değiştirmeniz gerekiyor. NoButton ifadesi herhangi bir buton göstermemesi gerekirken benim bilgisayarımda **Tamam** butonu yine de görünmektedir. Ekleyeceğiniz birden fazla buton varsa her birini | karakteriyle ayırmalısınız. Örnek:

```
QMessageBox.question(self, "Başlık", "Kullanıcı için sorun mesajı içeren mesaj kutusu."
,
                  QMessageBox.Close|QMessageBox.Help)
```

Sorun mesajı mesaj kutusunda diğerlerinin aksine **Tamam** Butonu yerine **Evet** ve **Hayır** butonları mevcuttur. Özel buton tanımladığınızda bunlar görünmez. Bu durumu göz alarak uygulamanızı geliştiriniz.

Göreceğiniz üzere statik methodların kullanım açısından birbirinden pek bir farkı yok. Gözle görülen tek fark kutumuzun solunda çıkan simgedir. Arka planda ise birbirleri arasında fark yoktur. Ama siz gerekli gördüğünüzde koyduğunuz butonlara ilgili slotları bağlayarak işlev kazandırabilirsiniz. Böylece farkı daha da belirginleştirmiş olursunuz.

Standart Dialoglar

Standart dialoglar PyQt5 ile gelen ve uygulamalarınızda kullanmanız için hazır gelen widgetlerdir. Bu dialoglar yardımı ile içlerinden kullanmak istediğiniz için sıfırdan üretmek yükünden kurtulursunuz. Beraberinde kullanımı kolaylaştıran methodlarda mevcuttur. Şimdi isterseniz sırayla bu dialogların kullanımını öğrenelim...

Renk Dialogu

Dosya Dialogu

Yazı Tipi Dialogu

Girdi Dialogu

Yazdırma Dialogu

Süreç Dialogu

Nasıl Yapılır?