7 Dialoge

Dialoge sind eine weitere Form von Fenstern, welche besonders dazu geeignet sind, Daten zu präsentieren oder Eingaben abzufragen. In PyQt gibt es etliche Standarddialoge wie zum Beispiel für das Öffnen von Dateien. Neben den Standarddialogen können auch eigene Dialoge erstellt werden.

Auf Desktop-Systemen wird meist zwischen **modalen** und **nicht-modalen Dialogen** unterschieden: Modale Dialoge erzwingen den Eingabefokus und es kann mit der Anwendung erst weitergearbeitet werden wenn dieser geschlossen wird. Nicht-modale Dialoge erlauben das weiterarbeiten mit der Applikation und sollten auch nur verwendet werden, wenn dies möglich ist.

7.1 Nachrichten und Fragen

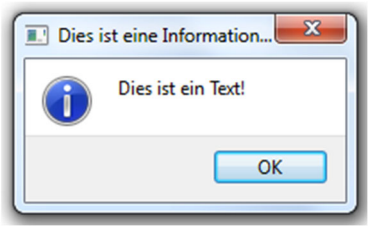
Nachrichtdialoge sind die einfachste Form von Dialogen. Diese Kategorie von Dialogen stellt eine Nachricht dar oder stellt eine Frage, welche verschiedene Antwortmöglichkeiten bietet. Diese Dialoge werden mit `QMessageBox` erstellt.

7.1.1 Informationsbox

Mit `QMessageBox.information(...)` wird das Informationsfenster erstellt. Der erste Parameter ist das parent Widget, also das Widget von dem das Informationsfenster gestartet wird, dies ist meist `self`, wenn das Informationsfenster innerhalb einer `QWidget`-Klasse erstellt wird.

```
titel = "Dies ist eine Informations-Box"
text = "Dies ist ein Text!"
QMessageBox.information(self, titel, text)
```

Der Text kann auch HTML-Tags enthalten und wird dann entsprechend formatiert.

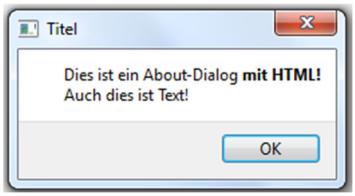


Beispiel der QMessageBox.information()

7.1.2 About und AboutQt Dialoge

Die meisten Applikationen haben einen "About" Dialog, welcher Informationen zur Applikation liefert. Unter MacOS kann ein About-Menu immer beim Applikations-Menu gefunden werden. Unter Windows ist dies nicht einheitlich. Um ein About-Fenster anzuzeigen kann `QMessageBox.about(...)` aufgerufen werden.

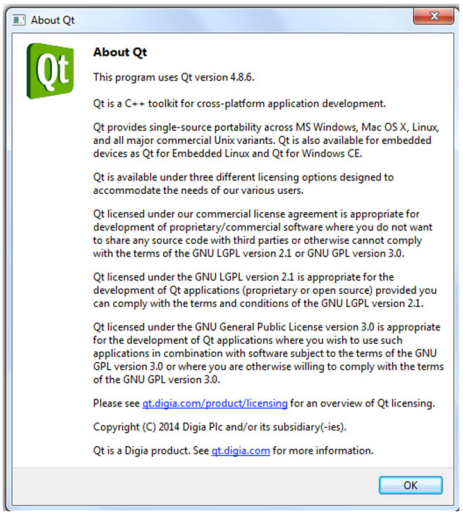
```
QMessageBox.about(self, titel, text)
```



Beispiel des QMessageBox.about()

Ein spezielles About-Fenster, welches informationen zu Qt anzeigt, kann mit `QMessageBox.aboutQt(...)` aufgerufen werden.

```
QMessageBox.aboutQt(self)  # Informationen über Qt
```

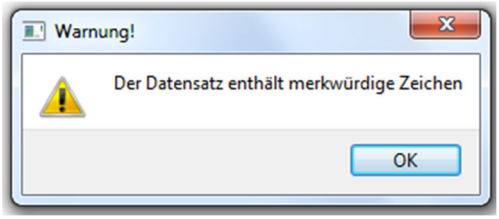


Screenshot aboutQt()-Dialog

7.1.3 Warnungen und kritische Fehler

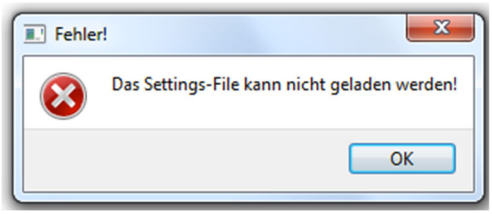
Warnungs-Dialoge können mit `QMessageBox.warning(...)` und Dialoge für kritische Fehler mit `QMessageBox.critical(...)` erstellt werden. Diese Dialoge werden in der Regel mit einem Ok Button bestätigt.

```
QMessageBox.warning(self, "Warnung!", "Der Datensatz enthält merkwürdige
Zeichen")
```



QMessageBox.warning(...)

```
QMessageBox.critical(self, "Fehler!", "Das Settings-File kann nicht geladen
werden!")
```

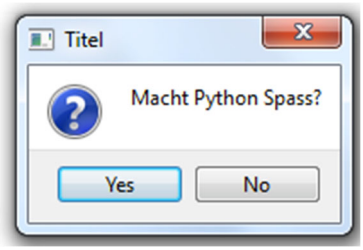


QMessageBox.critical(...)

7.1.4 Fragen

Ein Fragedialog stellt eine Frage, welche in der Regel mit Ja und Nein beantwortet werden kann. Der Fragedialog wird mit `QMessageBox.question(...)` realisiert. Die Antwort kann danach weiter verarbeitet werden.

```
antwort = QMessageBox.question(self, "Titel", "Macht Python Spass?",
                        QMessageBox.Yes, QMessageBox.No)
if antwort == QMessageBox.Yes:
   print("Sehr gut!")
elif anwort == QMessageBox.No:
   print("Sehr schade!")
```



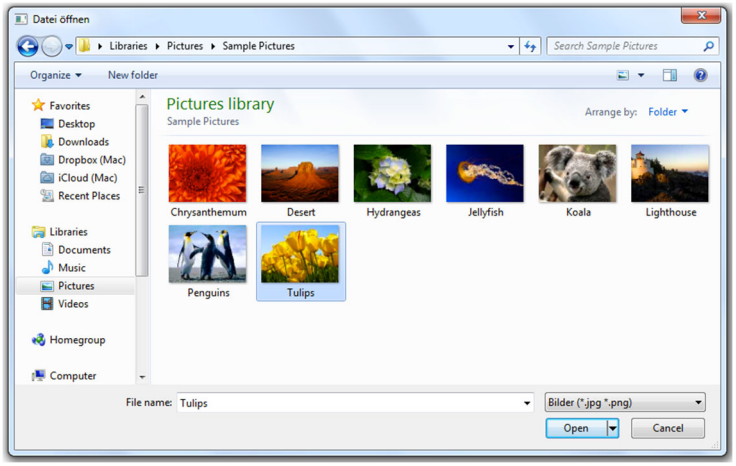
Dialog mit QMessageBox.question(...)

7.2 Datei-Dialoge

Datei-Dialoge für das Öffnen oder Speichern von Dateien sind häufig verwendete Dialoge. PyQt stellt dafür vier Arten von Datei- und Verzeichnisdialogen bereit. Diese Dialoge liefern eigentlich nur jeweils Filenamen resp. Verzeichnisnamen.

Die einfachste Art einen existierenden Dateinamen zu erhalten ist über die statische Funktion `QFileDialog.getOpenFileName(...)`. Der erste Parameter ist das Widget von dem es aufgerufen wird (meistens self), der zweite Parameter ist der Titel des Fensters, der dritte Parameter ist das Verzeichnis in dem gestartet wird und im

vierten Parameter werden die Unterstützten Filetypen angegeben. Sehen wir uns folgendes Beispiel an:



Dialog mit QFileDialog.getOpenFileName(...)

```
filename, filter = QFileDialog.getOpenFileName(self,
                                               "Datei öffnen",
                                               "C:/data/",
                                               "Bilder (*.jpg *.png)")
        if (filename != ""):
            print(filename)
            print(filter)
```

Falls "Abbrechen" gewählt wurde, so ist der Filename leer. Im Filter ist der gewählte Filter. Der Filter wird später noch detaillierter beschrieben.

Der dritte Parameter sollte eigentlich nie ein absoluter Pfad wie im Beispiel sein. Mit `QDir` und den `QStandardPaths` können bestimme Verzeichnisse ausgegeben werden. In folgender Tabelle sind die wichtigsten Verzeichnisse aufgelistet:

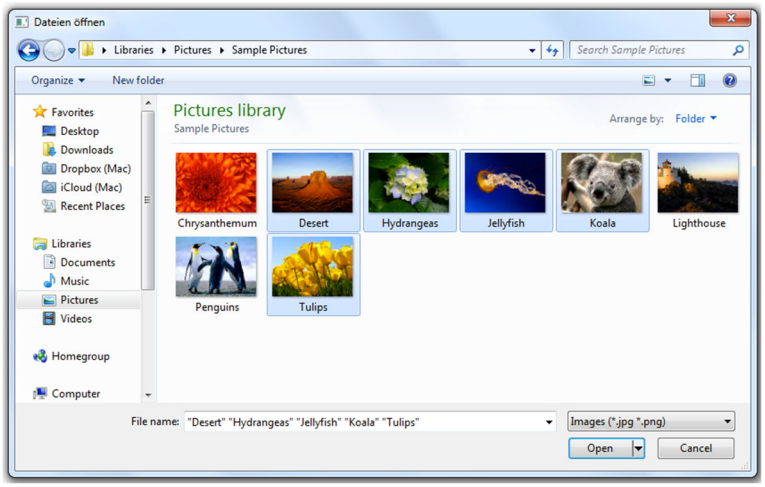
`QDir.currentPath()` `None`	Das aktuelle Verzeichnis
`QStandardPaths.storageLocation(QStandardPaths.DocumentsLocation)`	Dokument-Ordner
`QStandardPaths.storageLocation(QStandardPaths.FontsLocation)`	Font-Ordner
`QStandardPaths.storageLocation(QStandardPaths.ApplicationsLocation)`	Anwendungen
`QStandardPaths.storageLocation(QStandardPaths.MusicLocation)`	Musik-Ordner
`QStandardPaths.storageLocation(QStandardPaths.MoviesLocation)`	Filme-Ornder
`QStandardPaths.storageLocation(QStandardPaths.PicturesLocation)`	Bilder-Ordner
`QStandardPaths.storageLocation(QStandardPaths.TempLocation)`	Temporäre Daten
`QStandardPaths.storageLocation(QStandardPaths.HomeLocation)`	Home Verzeichnis
`QStandardPaths.storageLocation(QStandardPaths.DesktopLocation)`	Desktop

Siehe auch: https://doc.qt.io/qt-5/qstandardpaths.html

Der nächste File-Dialog ermöglicht das Wählen von **mehreren Dateien**. Dieser File-Dialog wird mit `QFileDialog.getOpenFileNames(...)` geöffnet. Diese Funktion funktioniert wie die vorherige mit dem Unterschied, dass neben dem Filter eine Liste von Filenamen zurückgegeben wird:

```
Location = QDesktopServices.storageLocation(QDesktopServices.PicturesLocation)
dateien, filter = QFileDialog.getOpenFileNames(self,
                                               "Dateien öffnen",
                                               Location,
                                               "Images (*.jpg *.png)")
if len(dateien)>0:
   for file in dateien:
      print(file)
   print(filter)
```

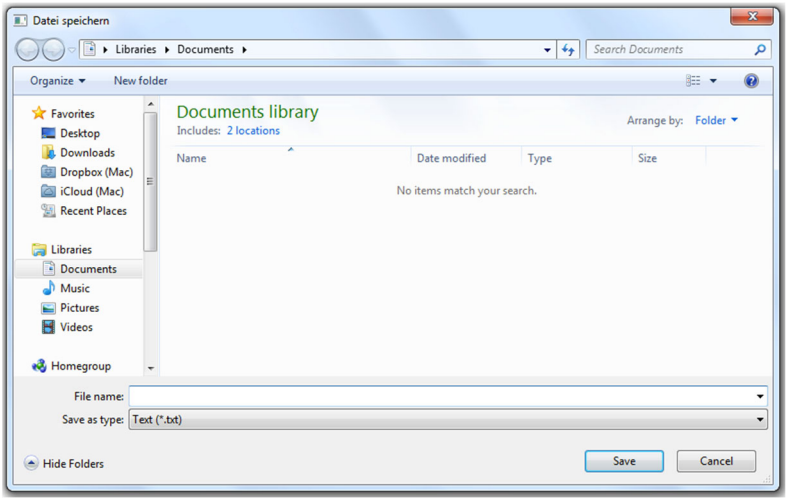
Wenn "Abbrechen" gedrückt wurde, so ist die Liste leer.



Mehrere Files mit QFileDialog.getOpenFileNames(...)

Der nächste Filedialog-Typ ist für das Speichern von Dateien. Es ist dasselbe Prinzip wie beim Laden einer Datei.
Der Speicher-File-Dialog wird mit `QFileDialog.getSaveFileName(...)` aufgerufen.

```
Location = QDesktopServices.storageLocation(QDesktopServices.DocumentsLocation)
filename, filter = QFileDialog.getSaveFileName(self,
                                               "Datei speichern",
                                               Location,
                                               "Text (*.txt)")
```



Dialog für das Speichern einer Datei: QFileDialog.getSaveFileName(...)

Wir haben bei den File-Dialogen die Filter gesehen, z.B. "`Text (*.txt)`". Wird der Filter weggelassen, so werden alle Dateien "`All Files (*.*)`" unterstützt. Sollen verschiedene Filter-Typen unterstützt werden können diese durch zwei Semikolon getrennt angegeben werden: "`Text (*.txt);;CSV (*.csv)`". Es können auch mehrere Einträge pro Filter-Kategorie gemacht werden. Diese werden dann einfach durch ein Leerzeichen getrennt: "`Images (*.png *.jpg *.tif *.bmp)`"

Beim Speichern-Dialog werden zwei Werte zurückgegeben: Der erste Wert ist der Dateiname und der zweite ist der Filter. Die File-Extension wird automatisch hinzugefügt, sollte diese fehlen - bei mehreren Filtern wird der Erste genommen.

Wird beim Speichern-Dialog auf "Abbrechen" geklickt, so ist der Filename leer ("").

7.3 Eingaben abfragen

Eine spezielle Form für die Eingabe von Werten geht über die Eingabedialoge der `QInputDialog` Klasse. Diese Eingabemethode kann aber ab und zu ganz nützlich sein. Es gibt verschiedene Typen dieser Eingabedialoge, welche hier nun vorgestellt werden.

7.3.1 Zahlen abfragen

Mit QInputDialog.getInt(...) kann eine ganze Zahl eingelesen werden. Der Funktionsaufruf sieht folgendermassen aus:

```
ergebnis, ok = QInputDialog.getInt(self,
                                   Titel,
                                   Text,
                                   Anfangswert,
                                   Minimalwert,
```

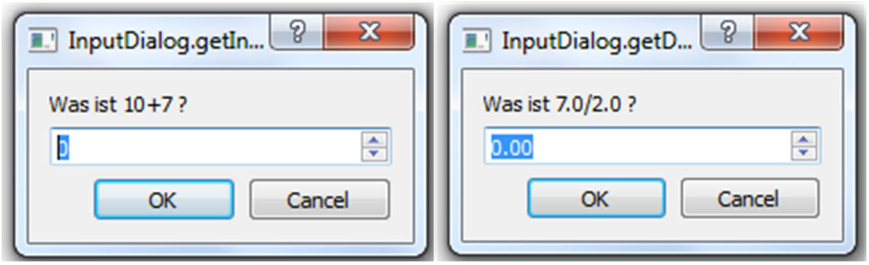
```
                                Maximalwert,
                                Schrittweite)
```

Ein konkretes Beispiel dazu wäre folgendes Ratespiel:

```
ergebnis, ok = QInputDialog.getInt(self,
                                "InputDialog.getInt()",
                                "Was ist 10+7 ? ",
                                0, 0, 20, 1)
        if ok:
            if ergebnis == 17:
                print("Richtig!")
```

Eine alternative Form für das Einlesen von Zahlen ist `QInputDialog.getDouble(...)`. Die Funktion ist wie `getInt(...)` aufgebaut, mit dem Unterschied, dass der letzte Parameter die Anzahl Nachkommastellen beinhaltet.

```
ergebnis, ok = QInputDialog.getDouble(self,
                                "InputDialog.getDouble()",
                                "Was ist 7.0/2.0 ? ",
                                0, 0, 10, 2)
        if ok:
            if ergebnis == 3.5:
                print("Richtig!")
```



Input-Dialoge für Zahlen

7.3.2 Aus Werten auswählen

Mit der Funktion `QInputDialog.getItem(...)` kann aus einer Liste von Werten ausgewählt werden. Dazu wird einfach eine Liste mit Zeichenketten angelegt.

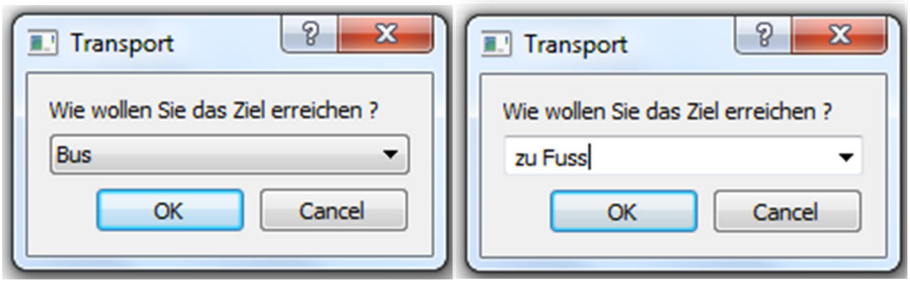
```
ergebnis, ok = QInputDialog.getItem(self,
                                Titel,
                                Text,
                                Werte,
                                AktuellesElement,
```

```
                                    Editierbar)
```

Werte ist eine Liste von Elementen, wie z.B. `["Glas", "Metall", "Holz"]`. *AktuellesElement* ist das zuerst angezeigte Element, beginnend bei 0. *Edtierbar* ist `True`, falls ein eigener Wert hinzugeführt werden kann, ansonsten `False`.
Ok ist ein bool, welcher True ist, falls "Ok" gedrückt wurde. Das Ergebnis enthält die ausgewählte Zeichenkette (oder auch den eingegebenen Text).

Ein konkretes Beispiel dazu wäre:

```
ergebnis, ok = QInputDialog.getItem(self, "Transport",
                    "Wie wollen Sie das Ziel erreichen ?",
                    ["Velo", "Bus", "Eisenbahn", "Auto"],
                    1, False)
if ok:
   print(ergebnis)
```



QInputDialog.getItem(...) mit Editierbarkeit ein- und ausgeschalten

7.3.3 Eine Zeichenkette einlesen

Mit `QInputDialog.getText(...)` kann eine Zeichenkette eingelesen werden. Dies geschieht mit folgendem Aufruf:

```
eingabe, ok = QInputDialog.getText(self,
                                   Titel,
                                   Text,
                                   Modus,
                                   Inhalt)
```

Titel und *Text* sind wie bei den vorherigen Dialogen.

Für den *Modus* gibt es drei Möglichkeiten:

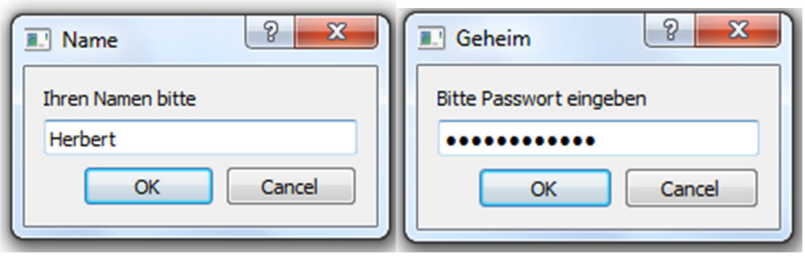
- Text normal einlesen (`QLineEdit.Normal`)
- Text als Passwort einlesen (`QLineEdit.Passwort`)
- Text als Passwort einlesen ohne sichtbare Zeichen (`QLineEdit.NoEcho`)

Inhalt füllt die Texteingabe mit einem vordefinierten Wert. Normalerweise ist diese leer und hat damit den Wert `""`.

Sehen wir und folgendes konkretes Beispiel an:

```
eingabe, ok = QInputDialog.getText(self, "Name",
                                   "Ihren Namen bitte",
                                   QLineEdit.Normal, "")

if ok:
   print("Eingabe:", eingabe)
```



QInputDialog.getText(...) mit verschiedenen Modi

Der Rückgabewert ist jeweils der eingegebene Text als Zeichenkette, sofern ok `True` ist.

7.4 Farbdialog

Ein weiterer wichtiger Dialog ist das auswählen einer Farbe. Der Aufruf eines Farbdialoges ist sehr einfach:

```
color = QColorDialog.getColor([initial=QColor])
```

color ist vom Typ "`QColor`" und hat unzählige Attribute und Methoden. Einige davon sind:

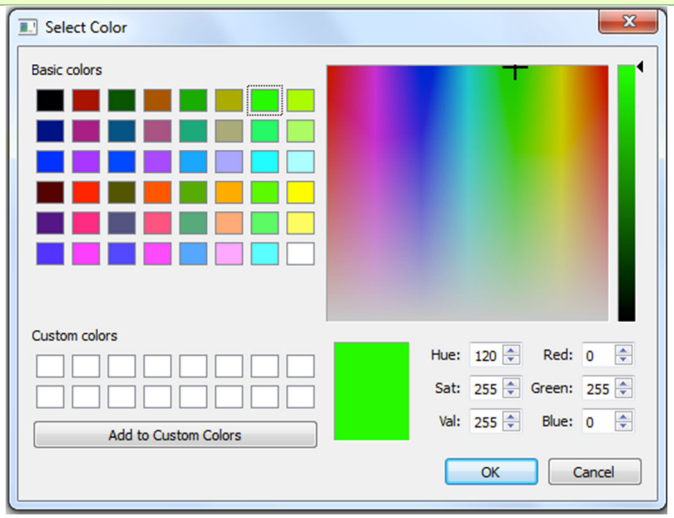
`QColor.red()`	Gibt den Rot-Wert als ganze Zahl [0,255] zurück
`QColor.green()`	Gibt den Grün-Wert als ganze Zahl [0,255] zurück
`QColor.blue()`	Gibt den Blau-Wert als ganze Zahl [0,255] zurück
`QColor.redF()`	Gibt dem Rot-Wert als floating point im Bereich [0,1] zurück
`QColor.greenF()`	Gibt dem Grün-Wert als floating point im Bereich [0,1] zurück
`QColor.blueF()`	Gibt dem Blau-Wert als floating point im Bereich [0,1] zurück

Ein konkretes Beispiel wäre:

```
color = QColorDialog.getColor()
print(color.red(), color.green(), color.blue())
```

Soll eine Farbe zu Beginn schon im Farbdialog gewählt werden, so geht das mit dem keyword Parameter "`initial`":

```
color = QColorDialog.getColor(initial=QColor(255,0,0))
```

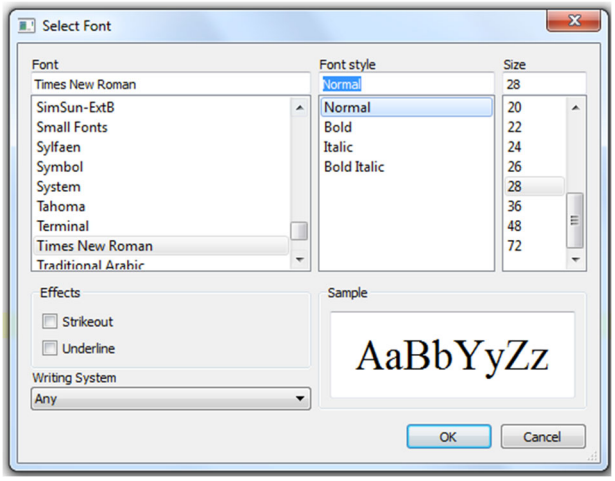


Beispiel eines Farbdialogs

7.5 Fontdialog

Zur Auswahl einer Schriftart kann der Font-Dialog nützlich sein. Mit dem Fontdialog kann eine im System vorhandene Schrift und deren Attribute gewählt werden:

```
font = QFontDialog.getFont([initial=QFont])
```



Der Rückgabewert ist vom Typ `QFont`.
Ein `QWidget` kann eine Font erhalten mit `widget.setFont(font)`.

Weitere Informationen zu Font-Dialogen kann in der PyQt oder Qt Dokumentation nachgeschlagen werden.

7.6 Druck- und Druckvorschau-Dialog

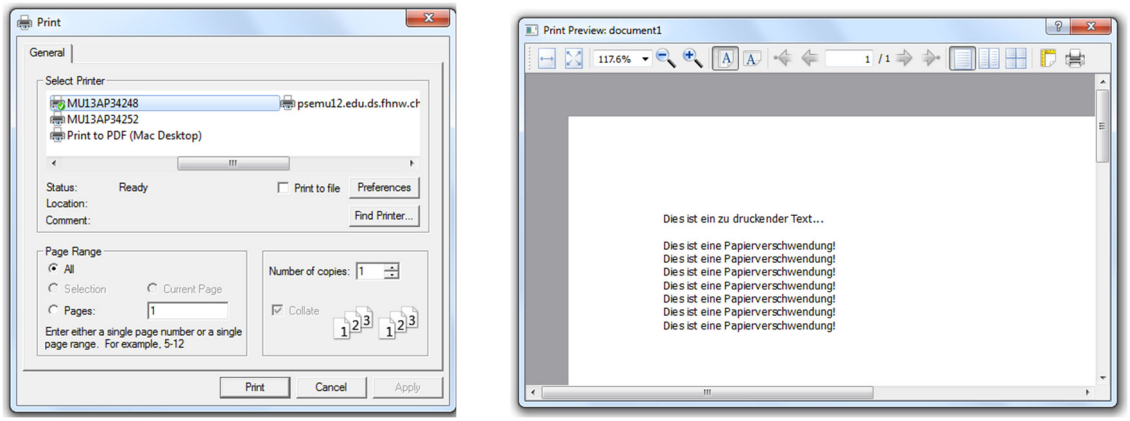
Drucken ist in PyQt denkbar einfach. Es können Dokumente aus Widgets gedruckt werden, wie zum Beispiel der Text aus einem `QTextEdit`. In Kapitel 0 werden wir komplexere Vektor-Grafiken erstellen, die auch ausgedruckt werden können. Dies geschieht über den `QPrintDialog()`.

Ein Druckdialog und Ausdruck aus einem `QTextEdit` (im Beispiel `self.textEdit`) kann folgendermassen erstellt werden:

```
dialog = QPrintDialog()
if dialog.exec() == QDialog.Accepted:
   doc = self.textEdit.document()
   doc.print_(dialog.printer())
```

Manchmal ist es auch sinnvoll eine Vorschau eines zu druckenden Elements darzustellen, dies kann im Falle unseres Text-Edits folgendermassen mithilfe von `QPrintPreviewDialog()` erstellt werden:

```
dialog = QPrintPreviewDialog()
dialog.paintRequested.connect(self.textEdit.print_)
dialog.exec()
```



QPrintDialog() und QPrintPreviewDialog()

7.7 Eigene Dialoge erstellen

Einen eigenen Dialog kann über die Klasse `QDialog` erstellt werden. Dies geschieht sehr ähnlich wie wir es bereits vom `QMainWindow` kennen. Dazu wird eine neue Klasse erstellt, welche von `QDialog` vererbt wird. Sehen wir uns folgendes Beispiel an:

```
class MyDialog(QDialog):
   def __init__(self, parent):
      super().__init__(parent)
      label = QLabel("Bitte Button klicken!")
      button = QPushButton("PushButton")
      layout = QHBoxLayout()
      layout.addWidget(label)
      layout.addWidget(button)
      self.setLayout(layout)
      button.clicked.connect(self.close)


# Dialog öffnen (innerhalb einer anderen QWidget-Klasse)
dialog = MyDialog(self)   # Dialog erstellen
dialog.exec()             # Dialog loop-starten
print("Dialog ende!")     # wird aufgerufen, wenn Dialog geschlossen
```

Der Unterschied zum bisher bekannten `QMainWidget` ist, dass ein Dialog über `exec()` ausgeführt werden muss. Das Programm fährt erst im Programm weiter, wenn der Dialog geschlossen ist. Hier sehen wir das Prinzip eines **modalen Dialogs**.

Gibt es in einem Projekt mehrere Fenster und Dialoge, so können diese in verschiedenen Python-Files stehen und mittels `import` dazugeladen werden. So können dieselben Dialoge in mehreren Projekten verwendet werden.