Widgets

WEEK 6

Instructor: Ph.D. Oleg Tymchuk

Overview

Display Widgets

Buttons

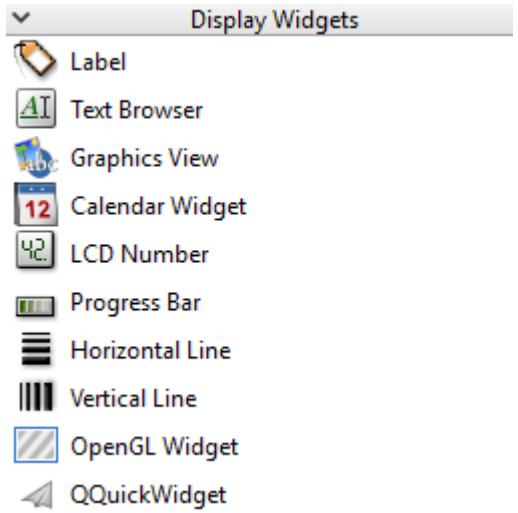
Input Widgets

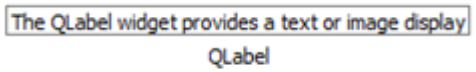
Containers

Item Views (Model Based)

Item Widgets (Item Based)

Display Widgets



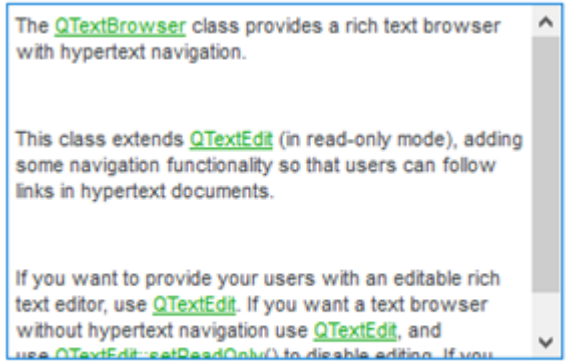


QLabel

QLCDNumber

QCalendarWidget

QLabel



QTextBrowser

QProgressBar

Display Widgets. QLabel

- The QLabel widget provides a text or image display
- No user interaction functionality is provided
- The visual appearance of the label can be configured in various ways, and it can be used for specifying a focus mnemonic key for another widget

A QLabel can contain any of the following content types:

- Plain text
- Rich text
- A pixmap
- A movie
- A number
- Nothing

Display Widgets. QLabel

Methods
QPicture picture() • returns the label's picture
QPixmap pixmap() • returns the label's pixmap
QMovie movie() • returns the label's movie
QString text() • returns the label's text

Slots
clear() • clears any label contents
setMovie (QMovie movie) • sets the label contents to movie
setNum(int num) • sets the label contents to plain text containing the textual representation of integer num
setNum(float num) • sets the label contents to plain text containing the textual representation of float num
setPicture (QPicture) • sets the label contents to picture
setPixmap (QPixmap) • sets the label contents to pixmap
setText (QString) • sets the label contents to text

Signals
linkActivated () • this signal is emitted when the user clicks a link
linkHovered () • this signal is emitted when the user hovers over a link

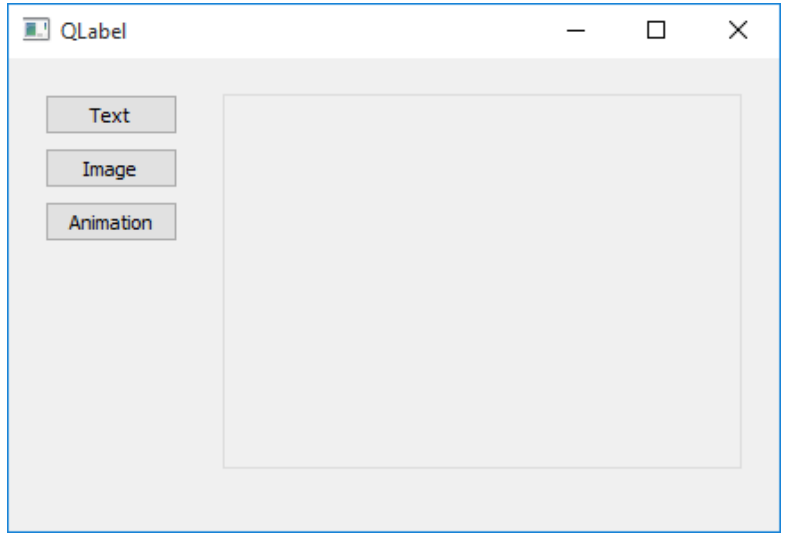
QLabel Class - http://doc.qt.io/qt-5.8/qlabel.html

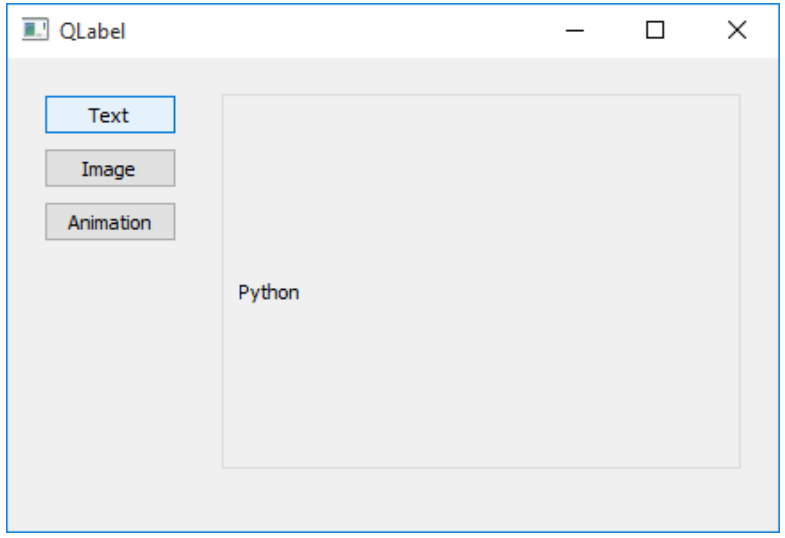
Display Widgets. QLabel

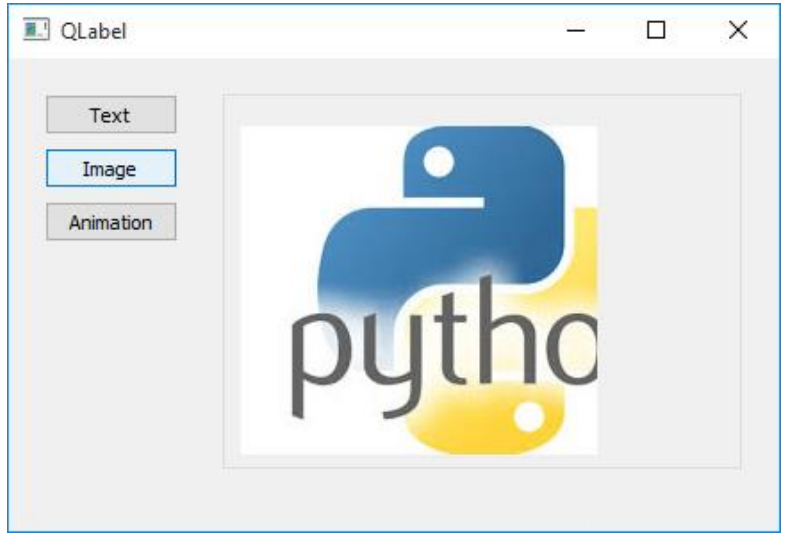
```
import sys
from PyQt5 import QtGui, QtWidgets, uic


class QLabelExample(QtWidgets.QMainWindow):

    def __init__(self, parent=None):
        super(QLabelExample, self).__init__(parent)
        # Set up the user interface from Designer.
        self.ui = uic.loadUi("qlabel_ui.ui")
        # Connect Btn_text clicked signal to slot
        self.ui.Btn_text.clicked.\
            connect(self.label_text)
        # Connect Btn_image clicked signal to slot
        self.ui.Btn_image.clicked.\
            connect(self.label_image)
        # Connect Btn_animation clicked signal to slot
        self.ui.Btn_animation.clicked.\
            connect(self.label_animation)
        self.ui.show()
```

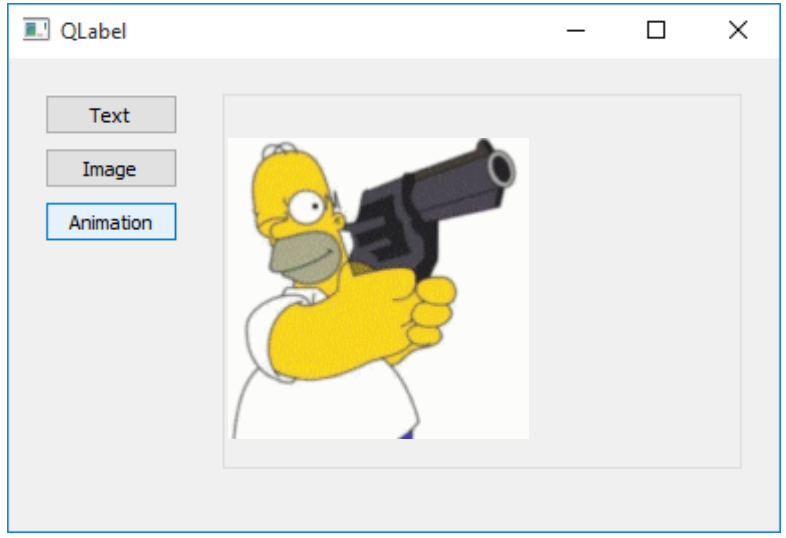
```
    # Set text in label widget
    def label_text(self):
        self.ui.label.setText('Python')
    # Set image in label widget
    def label_image(self):
        pixmap = QtGui.QPixmap('image.jpg')
        self.ui.label.setPixmap(pixmap)
    # Set GIF animation in label widget
    def label_animation(self):
        movie = QtGui.QMovie('anime.gif')
        self.ui.label.setMovie(movie)
        movie.start()

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = QLabelExample()
    sys.exit(app.exec())
```

Display Widgets. QLabel









Display Widgets. QLCDNumber

- The QLCDNumber widget displays a number with LCD-like digits
- It can display a number in just about any size
- It can display decimal, hexadecimal, octal or binary numbers
- It is easy to connect to data sources using the display() slot, which is overloaded to take any of five argument types
- These digits and other symbols can be shown: 0/O, 1, 2, 3, 4, 5/S, 6, 7, 8, 9/g, minus, decimal point, A, B, C, D, E, F, h, H, L, o, P, r, u, U, Y, colon, degree sign (which is specified as single quote in the string) and space. QLCDNumber substitutes spaces for illegal characters

Display Widgets. QLCDNumber

Methods
int digitCount() • returns the current number of digits displayed
int intValue() • returns the displayed value rounded to the nearest integer
Mode mode() • returns the current display mode (number base)
setDigitCount(int numDigits) • sets the number of displayed digits
setMode(Mode) • sets the display mode (number base)
float value() • returns the displayed value

Slots
display(QString s) • sets the LCDNumber contents to string
display(float num) • sets the LCDNumber contents to float num
display(int num) • sets the LCDNumber contents to integer num
setBinMode() • calls setMode(Bin)
setDecMode() • calls setMode(Dec)
setHexMode() • calls setMode(Hex)
setOctMode() • calls setMode(Oct)

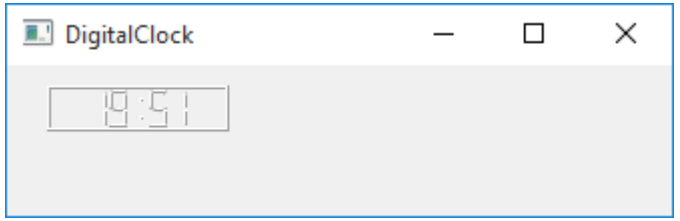
Signals
overflow() • This signal is emitted whenever the QLCDNumber is asked to display a too-large number or a too-long string

QLCDNumber Class - http://doc.qt.io/qt-5.8/qlcdnumber.html

Display Widgets. QLCDNumber

```
import sys
from PyQt5 import QtWidgets, QtCore, uic

class DigitalClock(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(DigitalClock, self).__init__(parent)
        # Set up the user interface from Designer.
        self.ui = uic.loadUi('qlcdnumber_ui.ui')
        # Set up a one-second timer.
        timer = QtCore.QTimer(self)
        timer.start(1000)
        # Connect timer timeout signal to the slot.
        timer.timeout.connect(self.showTime)
        # Call the showTime() slot
        self.showTime()
        self.ui.show()

    # Update the clock display.
    def showTime(self):
        time = QtCore.QTime.currentTime()
        # Convert time into a 'hh:mm' string.
        text = time.toString('hh:mm')
        self.ui.lcdNumber.display(text)
```

```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = DigitalClock()
    sys.exit(app.exec())
```



Display Widgets. QProgressBar

- The QProgressBar widget provides a horizontal or vertical progress bar

- A progress bar is used to give the user an indication of the progress of an operation and to reassure them that the application is still running

- The progress bar uses the concept of steps.
  You set it up by specifying the minimum and maximum possible step values, and it will display the percentage of steps that have been completed when you later give it the current step value.
  The percentage is calculated by dividing the progress (value() - minimum()) divided by maximum() - minimum()

- If minimum and maximum both are set to 0, the bar shows a busy indicator instead of a percentage of steps

Display Widgets. QProgressBar

Methods
int maximum() • returns the progress bar's maximum value
int minimum() • returns the progress bar's minimum value
QString text() • returns the descriptive text shown with the progress bar
int value() • returns the progress bar's current value

Slots
reset() • resets the progress bar
setMaximum(int maximum) • sets the progress bar's maximum value
setMinimum(int minimum) • sets the progress bar's minimum value
setOrientation(Qt::Orientation) • sets the orientation of the progress bar (Qt::Horizontal (the default) or Qt::Vertical)
setRange(int minimum, int maximum) • sets the progress bar's minimum and maximum values to minimum and maximum respectively
setValue(int value) • sets the progress bar's value

Signals
valueChanged(int value) • this signal is emitted when the value shown in the progress bar changes

QProgressBar Class - http://doc.qt.io/qt-5.8/qprogressbar.html

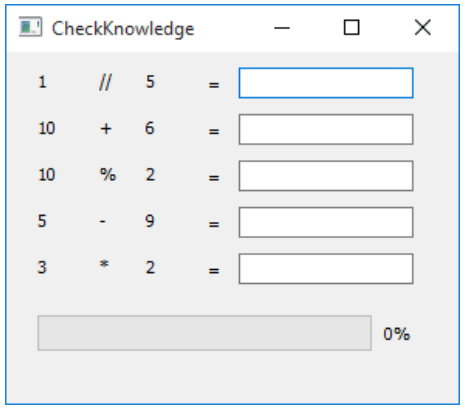
Display Widgets. QProgressBar

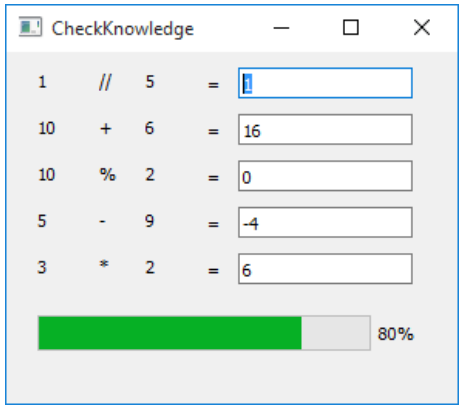
```
import sys
from PyQt5 import QtWidgets, uic
import random as rnd

class CheckKnowledge(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(CheckKnowledge, self).__init__(parent)
        # Set up the user interface from Designer.
        self.ui = uic.loadUi('qprogressbar_ui.ui')
        # Create list of QLabels (first operands in expressions)
        self.A = [self.ui.a1, self.ui.a2, self.ui.a3, self.ui.a4, self.ui.a5]
        # Create list of QLabels (second operands in expressions)
        self.B = [self.ui.b1, self.ui.b2, self.ui.b3, self.ui.b4, self.ui.b5]
        # Create list of QLabels (operators in expressions)
        self.C = [self.ui.op1, self.ui.op2, self.ui.op3, self.ui.op4, self.ui.op5]
        # Create list of QLineEdits (results of expressions)
        self.res_ans = [self.ui.res1, self.ui.res2, self.ui.res3, self.ui.res4, self.ui.res5]
        # Connect textChanged signal of QLineEdits to the check_ans() slot.
        for i in range(len(self.res_ans)):
            self.res_ans[i].textChanged.connect(self.check_ans)
        self.operations = ['+', '-', '*', '//', '%']
        self.res = [0] * len(self.operations)
        self.data_generator()
        self.ui.show()
```

Display Widgets. QProgressBar

```
# Generate data for expressions
def data_generator(self):
    rnd.shuffle(self.operations)
    for i in range(len(self.operations)):
        a, b = rnd.randint(1, 10), rnd.randint(1, 10)
        self.A[i].setText(str(a))
        self.B[i].setText(str(b))
        self.C[i].setText(self.operations[i])
        self.res[i] = {'+': lambda x, y: x + y,
                       '-':lambda x, y: x - y,
                       '*':lambda x, y: x * y,
                       '//':lambda x, y: x // y,
                       '%':lambda x, y: x % y}
                       [self.operations[i]](a, b)
# Check user's answers
def check_ans(self):
    correct_ans = 0
    for i in range(len(self.res)):
        try:
            if self.res_ans[i].text() != '' and
                    int(self.res_ans[i].text()) == self.res[i]:
                correct_ans += 1
        except: pass
    self.ui.progressBar.setValue(100 / len(self.res)*correct_ans)
```

```
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = CheckKnowledge()
    sys.exit(app.exec())
```





Display Widgets. QTextBrowser

- The QTextBrowser class provides a rich text browser with hypertext navigation
- This class extends QTextEdit (in read-only mode), adding some navigation functionality so that users can follow links in hypertext documents
- If you want to provide your users with an editable rich text editor, use QTextEdit
- If you want a text browser without hypertext navigation use QTextEdit, and use QTextEdit.setReadOnly() to disable editing
- If you just need to display a small piece of rich text use QLabel

Display Widgets. QTextBrowser

Methods

int backwardHistoryCount()
- returns the number of locations backward in the history

int forwardHistoryCount()
- returns the number of locations forward in the history

clearHistory()
- clears the history of visited documents and disables the forward and backward navigation

QString historyTitle(int i)
- returns the documentTitle() of the HistoryItem

QUrl historyUrl(int i)
- returns the url of the HistoryItem

QUrl source()
- return the url of the displayed document

Slots

backward()
- changes the document displayed to the previous document in the list of documents built by navigating links

forward()
- changes the document displayed to the next document in the list of documents built by navigating links

home()
- changes the document displayed to be the first document from the history

reload()
- reloads the current set source

setSource(QUrl name)
- sets the name of the displayed document

Signals

anchorClicked(QUrl link)
- this signal is emitted when the user clicks an anchor

historyChanged()
- this signal is emitted when the history changes

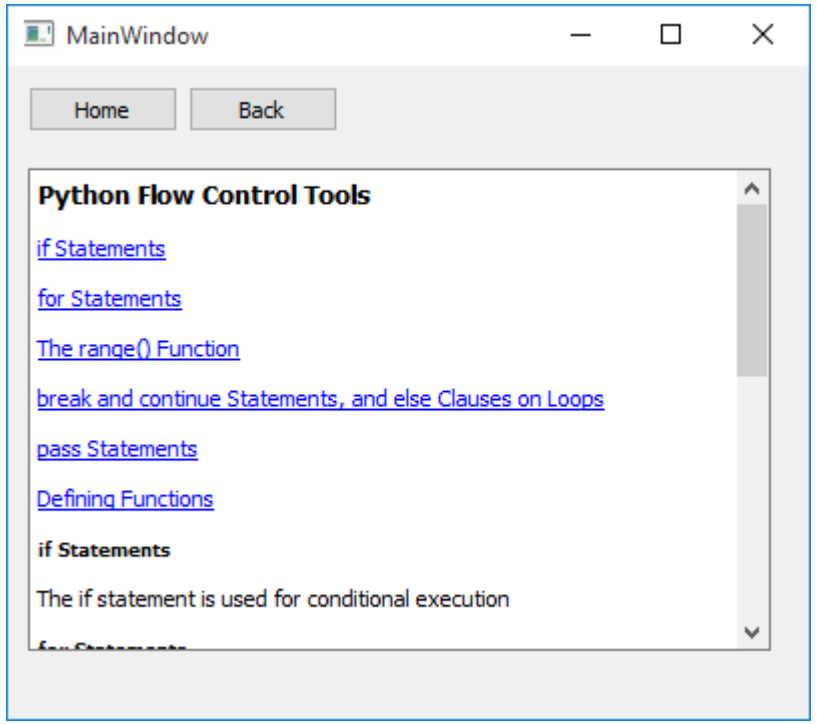
sourceChanged(QUrl src)
- this signal is emitted when the source has changed, src being the new source.

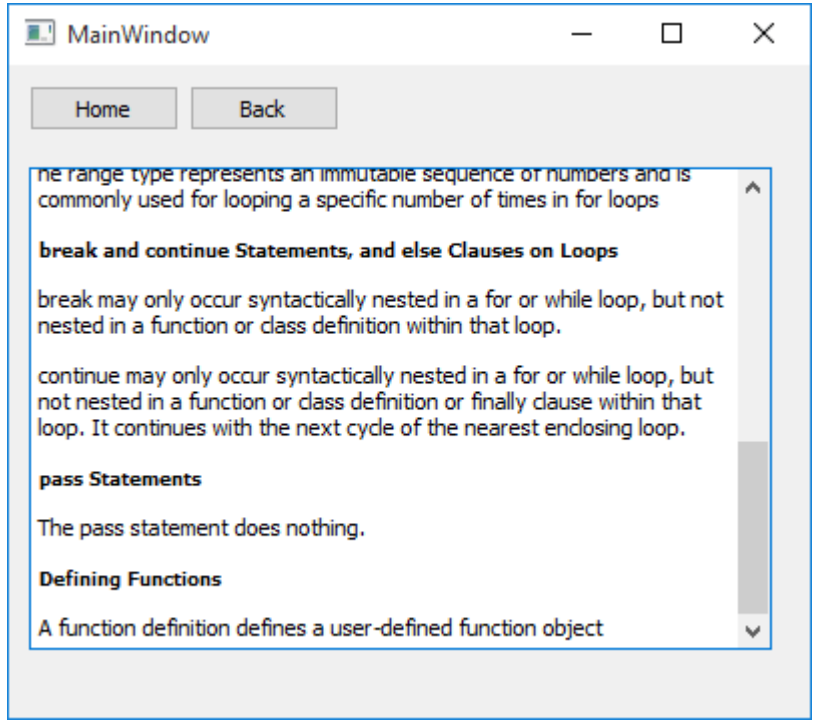
QTextBrowser Class - http://doc.qt.io/qt-5.8/qtextbrowser.html

Display Widgets. QTextBrowser

```
import sys
from PyQt5 import QtWidgets, uic, QtCore

# Simple Help Engine.
class HelpSystem(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(HelpSystem, self).__init__(parent)
        # Set up the user interface from Designer.
        self.ui = uic.loadUi('qtextbrowser_ui.ui')
        # Connect clicked signal of homeButton to the QTextBrowser.home() slot.
        self.ui.homeButton.clicked.connect(self.ui.textBrowser.home)
        # Connect clicked signal of backButton to the QTextBrowser.backward() slot.
        self.ui.backButton.clicked.connect(self.ui.textBrowser.backward)
        # Connect anchorClicked signal of textBrowser to the update_browser() slot.
        self.ui.textBrowser.anchorClicked.connect(self.update_browser)
        # Loads help.html as the source text.
        self.ui.textBrowser.setSource(QtCore.QUrl('help.html'))
        self.ui.show()

    def update_browser(self, link):
        self.ui.textBrowser.setSource(link)

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = HelpSystem()
    sys.exit(app.exec())
```

Display Widgets. QTextBrowser





Display Widgets. QCalendarWidget

- The QCalendarWidget class provides a monthly based calendar widget allowing the user to select a date

- The widget is initialized with the current month and year, but QCalendarWidget provides several public slots to change the year and month that is shown

- The user can select a date using both mouse and keyboard

- A newly created calendar widget uses abbreviated day names, and both Saturdays and Sundays are marked in red. The calendar grid is not visible. The week numbers are displayed, and the first column day is the first day of the week for the calendar's locale

Display Widgets. QCalendarWidget

Methods	Slots	Signals
setFirstDayOfWeek(dayOfWeek) • sets the value identifying the day displayed in the first column	setCurrentPage(int year, int month) • displays the given month of the given year without changing the selected date	clicked(QDate date) • this signal is emitted when a mouse button is clicked
QDate selectedDate() • returns the currently selected date	setSelectedDate(QDate date) • sets the selected date	selectionChanged() • this signal is emitted when the currently selected date is changed
setMaximumDate(QDate date) • sets the maximum date of the currently specified date range	showNextMonth() • shows the next month relative to the currently displayed month	
setMinimumDate(QDate date) • sets the maximum date of the currently specified date range	showPreviousMonth() • shows the previous month relative to the currently displayed month	
int monthShown() • returns the currently displayed month	showSelectedDate() • shows the month of the selected date	
int yearShown() • returns the year of the currently displayed month	showToday() • shows the month of the today's date	

QCalendarWidget Class - http://doc.qt.io/qt-5.8/qcalendarwidget.html

Display Widgets. QCalendarWidget

```
import sys
from PyQt5 import uic, QtCore, QtWidgets

class CalendarExample(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(CalendarExample, self).__init__(parent)
        # Set up the user interface from Designer.
        self.ui = uic.loadUi('qcalendar_ui.ui')
        self.ui.calendarWidget.setFirstDayOfWeek(QtCore.Qt.Monday)
        # Connect clicked signal of QCalendarWidget to the slot.
        self.ui.calendarWidget.clicked.connect(self.show_date)
        self.ui.show()

    def show_date(self, date):
        self.ui.label.setText(date.toString())

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = CalendarExample()
    sys.exit(app.exec())
```