```
# The PEP 484 type hints stub file for the QtWidgets module.
#
# Generated by SIP 6.7.12
#
# Copyright (c) 2023 Riverbank Computing Limited <info@riverbankcomputing.com>
#
# This file is part of PyQt5.
#
# This file may be used under the terms of the GNU General Public License
# version 3.0 as published by the Free Software Foundation and appearing in
# the file LICENSE included in the packaging of this file.  Please review the
# following information to ensure the GNU General Public License version 3.0
# requirements will be met: http://www.gnu.org/copyleft/gpl.html.
#
# If you do not wish to use this file under the terms of the GPL version 3.0
# then you may purchase a commercial license.  For more information contact
# info@riverbankcomputing.com.
#
# This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
# WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.


import typing

import PyQt5.sip

from PyQt5 import QtCore
from PyQt5 import QtGui

# Support for QDate, QDateTime and QTime.
import datetime

# Convenient type aliases.
PYQT_SIGNAL = typing.Union[QtCore.pyqtSignal, QtCore.pyqtBoundSignal]
PYQT_SLOT = typing.Union[typing.Callable[..., Any], QtCore.pyqtBoundSignal]

# Convenient aliases for complicated OpenGL types.
PYQT_OPENGL_ARRAY = typing.Union[typing.Sequence[int], typing.Sequence[float],
        PyQt5.sip.Buffer, None]
PYQT_OPENGL_BOUND_ARRAY = typing.Union[typing.Sequence[int],
        typing.Sequence[float], PyQt5.sip.Buffer, int, None]


class QWidget(QtCore.QObject, QtGui.QPaintDevice):

    class RenderFlag(int):
        DrawWindowBackground = ... # type: QWidget.RenderFlag
        DrawChildren = ... # type: QWidget.RenderFlag
        IgnoreMask = ... # type: QWidget.RenderFlag

    class RenderFlags(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> 'QWidget.RenderFlags': ...
        def __xor__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> 'QWidget.RenderFlags': ...
        def __ior__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> 'QWidget.RenderFlags': ...
        def __or__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> 'QWidget.RenderFlags': ...
        def __iand__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> 'QWidget.RenderFlags': ...
        def __and__(self, f: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag']) -> 'QWidget.RenderFlags': ...
        def __invert__(self) -> 'QWidget.RenderFlags': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...
    def __init__(self, parent: typing.Optional['QWidget'] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def screen(self) -> typing.Optional[QtGui.QScreen]: ...
    def setWindowFlag(self, a0: QtCore.Qt.WindowType, on: bool = ...) -> None: ...
    def hasTabletTracking(self) -> bool: ...
    def setTabletTracking(self, enable: bool) -> None: ...
    windowIconTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    windowIconChanged: typing.ClassVar[QtCore.pyqtSignal]
    windowTitleChanged: typing.ClassVar[QtCore.pyqtSignal]
    def toolTipDuration(self) -> int: ...
    def setToolTipDuration(self, msec: int) -> None: ...
    def initPainter(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    def sharedPainter(self) -> typing.Optional[QtGui.QPainter]: ...
    def nativeEvent(self, eventType: typing.Union[QtCore.QByteArray, bytes, bytearray], message:
typing.Optional[PyQt5.sip.voidptr]) -> typing.Tuple[bool, typing.Optional[int]]: ...
    def windowHandle(self) -> typing.Optional[QtGui.QWindow]: ...
    @staticmethod
    def createWindowContainer(window: typing.Optional[QtGui.QWindow], parent: typing.Optional['QWidget'] = ..., flags:
typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> 'QWidget': ...
    def grab(self, rectangle: QtCore.QRect = ...) -> QtGui.QPixmap: ...
    def hasHeightForWidth(self) -> bool: ...
    def setInputMethodHints(self, hints: typing.Union[QtCore.Qt.InputMethodHints, QtCore.Qt.InputMethodHint]) -> None: ...
    def inputMethodHints(self) -> QtCore.Qt.InputMethodHints: ...
    def previousInFocusChain(self) -> typing.Optional['QWidget']: ...
    def contentsMargins(self) -> QtCore.QMargins: ...
    def ungrabGesture(self, type: QtCore.Qt.GestureType) -> None: ...
    def grabGesture(self, type: QtCore.Qt.GestureType, flags: typing.Union[QtCore.Qt.GestureFlags, QtCore.Qt.GestureFlag] =
...) -> None: ...
    def setGraphicsEffect(self, effect: typing.Optional['QGraphicsEffect']) -> None: ...
    def graphicsEffect(self) -> typing.Optional['QGraphicsEffect']: ...
    def graphicsProxyWidget(self) -> typing.Optional['QGraphicsProxyWidget']: ...
    def windowFilePath(self) -> str: ...
    def setWindowFilePath(self, filePath: typing.Optional[str]) -> None: ...
    def nativeParentWidget(self) -> typing.Optional['QWidget']: ...
    def effectiveWinId(self) -> PyQt5.sip.voidptr: ...
    def unsetLocale(self) -> None: ...
    def locale(self) -> QtCore.QLocale: ...
    def setLocale(self, locale: QtCore.QLocale) -> None: ...
    @typing.overload
    def render(self, target: typing.Optional[QtGui.QPaintDevice], targetOffset: QtCore.QPoint = ..., sourceRegion:
QtGui.QRegion = ..., flags: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag'] = ...) -> None: ...
    @typing.overload
    def render(self, painter: typing.Optional[QtGui.QPainter], targetOffset: QtCore.QPoint = ..., sourceRegion: QtGui.QRegion
= ..., flags: typing.Union['QWidget.RenderFlags', 'QWidget.RenderFlag'] = ...) -> None: ...
    def restoreGeometry(self, geometry: typing.Union[QtCore.QByteArray, bytes, bytearray]) -> bool: ...
    def saveGeometry(self) -> QtCore.QByteArray: ...
    def setShortcutAutoRepeat(self, id: int, enabled: bool = ...) -> None: ...
    def styleSheet(self) -> str: ...
    def setStyleSheet(self, styleSheet: typing.Optional[str]) -> None: ...
    def setAutoFillBackground(self, enabled: bool) -> None: ...
    def autoFillBackground(self) -> bool: ...
    def setWindowModality(self, windowModality: QtCore.Qt.WindowModality) -> None: ...
    def windowModality(self) -> QtCore.Qt.WindowModality: ...
    def testAttribute(self, attribute: QtCore.Qt.WidgetAttribute) -> bool: ...
    def parentWidget(self) -> typing.Optional['QWidget']: ...
    def height(self) -> int: ...
    def width(self) -> int: ...
    def size(self) -> QtCore.QSize: ...
    def geometry(self) -> QtCore.QRect: ...
    def rect(self) -> QtCore.QRect: ...
    def isHidden(self) -> bool: ...
    def isVisible(self) -> bool: ...
    def updatesEnabled(self) -> bool: ...
    def underMouse(self) -> bool: ...
    def hasMouseTracking(self) -> bool: ...
    def setMouseTracking(self, enable: bool) -> None: ...
    def fontInfo(self) -> QtGui.QFontInfo: ...
    def fontMetrics(self) -> QtGui.QFontMetrics: ...
def font(self) -> QtGui.QFont: ...
def maximumHeight(self) -> int: ...
def maximumWidth(self) -> int: ...
def minimumHeight(self) -> int: ...
def minimumWidth(self) -> int: ...
def isModal(self) -> bool: ...
def isEnabled(self) -> bool: ...
def isWindow(self) -> bool: ...
def winId(self) -> PyQt5.sip.voidptr: ...
def windowFlags(self) -> QtCore.Qt.WindowFlags: ...
def windowType(self) -> QtCore.Qt.WindowType: ...
def focusPreviousChild(self) -> bool: ...
def focusNextChild(self) -> bool: ...
def focusNextPrevChild(self, next: bool) -> bool: ...
def destroy(self, destroyWindow: bool = ..., destroySubWindows: bool = ...) -> None: ...
def create(self, window: PyQt5.sip.voidptr = ..., initializeWindow: bool = ..., destroyOldWindow: bool = ...) -> None: ...
def updateMicroFocus(self) -> None: ...
def inputMethodQuery(self, a0: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
def inputMethodEvent(self, a0: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
def metric(self, a0: QtGui.QPaintDevice.PaintDeviceMetric) -> int: ...
def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
def hideEvent(self, a0: typing.Optional[QtGui.QHideEvent]) -> None: ...
def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
def dropEvent(self, a0: typing.Optional[QtGui.QDropEvent]) -> None: ...
def dragLeaveEvent(self, a0: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
def dragMoveEvent(self, a0: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
def dragEnterEvent(self, a0: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
def actionEvent(self, a0: typing.Optional[QtGui.QActionEvent]) -> None: ...
def tabletEvent(self, a0: typing.Optional[QtGui.QTabletEvent]) -> None: ...
def contextMenuEvent(self, a0: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
def closeEvent(self, a0: typing.Optional[QtGui.QCloseEvent]) -> None: ...
def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
def moveEvent(self, a0: typing.Optional[QtGui.QMoveEvent]) -> None: ...
def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
def leaveEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
def enterEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
def focusOutEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
def focusInEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
def keyReleaseEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
def wheelEvent(self, a0: typing.Optional[QtGui.QWheelEvent]) -> None: ...
def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def mouseDoubleClickEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
customContextMenuRequested: typing.ClassVar[QtCore.pyqtSignal]
def isAncestorOf(self, child: typing.Optional['QWidget']) -> bool: ...
def ensurePolished(self) -> None: ...
def paintEngine(self) -> typing.Optional[QtGui.QPaintEngine]: ...
def setAttribute(self, attribute: QtCore.Qt.WidgetAttribute, on: bool = ...) -> None: ...
@typing.overload
def childAt(self, p: QtCore.QPoint) -> typing.Optional['QWidget']: ...
@typing.overload
def childAt(self, ax: int, ay: int) -> typing.Optional['QWidget']: ...
@staticmethod
def find(a0: PyQt5.sip.voidptr) -> typing.Optional['QWidget']: ...
def overrideWindowFlags(self, type: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType]) -> None: ...
def setWindowFlags(self, type: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType]) -> None: ...
def actions(self) -> typing.List['QAction']: ...
def removeAction(self, action: typing.Optional['QAction']) -> None: ...
def insertActions(self, before: typing.Optional['QAction'], actions: typing.Iterable['QAction']) -> None: ...
def insertAction(self, before: typing.Optional['QAction'], action: typing.Optional['QAction']) -> None: ...
def addActions(self, actions: typing.Iterable['QAction']) -> None: ...
def addAction(self, action: typing.Optional['QAction']) -> None: ...
def setAcceptDrops(self, on: bool) -> None: ...
def acceptDrops(self) -> bool: ...
def nextInFocusChain(self) -> typing.Optional['QWidget']: ...
def focusWidget(self) -> typing.Optional['QWidget']: ...
@typing.overload
```

```
    def scroll(self, dx: int, dy: int) -> None: ...
    @typing.overload
    def scroll(self, dx: int, dy: int, a2: QtCore.QRect) -> None: ...
    @typing.overload
    def setParent(self, parent: typing.Optional['QWidget']) -> None: ...
    @typing.overload
    def setParent(self, parent: typing.Optional['QWidget'], f: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType]) -
> None: ...
    def updateGeometry(self) -> None: ...
    def setLayout(self, a0: typing.Optional['QLayout']) -> None: ...
    def layout(self) -> typing.Optional['QLayout']: ...
    def contentsRect(self) -> QtCore.QRect: ...
    def getContentsMargins(self) -> typing.Tuple[typing.Optional[int], typing.Optional[int], typing.Optional[int],
typing.Optional[int]]: ...
    @typing.overload
    def setContentsMargins(self, left: int, top: int, right: int, bottom: int) -> None: ...
    @typing.overload
    def setContentsMargins(self, margins: QtCore.QMargins) -> None: ...
    def visibleRegion(self) -> QtGui.QRegion: ...
    def heightForWidth(self, a0: int) -> int: ...
    @typing.overload
    def setSizePolicy(self, a0: 'QSizePolicy') -> None: ...
    @typing.overload
    def setSizePolicy(self, hor: 'QSizePolicy.Policy', ver: 'QSizePolicy.Policy') -> None: ...
    def sizePolicy(self) -> 'QSizePolicy': ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def overrideWindowState(self, state: typing.Union[QtCore.Qt.WindowStates, QtCore.Qt.WindowState]) -> None: ...
    def setWindowState(self, state: typing.Union[QtCore.Qt.WindowStates, QtCore.Qt.WindowState]) -> None: ...
    def windowState(self) -> QtCore.Qt.WindowStates: ...
    def isFullScreen(self) -> bool: ...
    def isMaximized(self) -> bool: ...
    def isMinimized(self) -> bool: ...
    def isVisibleTo(self, a0: typing.Optional['QWidget']) -> bool: ...
    def adjustSize(self) -> None: ...
    @typing.overload
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    @typing.overload
    def setGeometry(self, ax: int, ay: int, aw: int, ah: int) -> None: ...
    @typing.overload
    def resize(self, a0: QtCore.QSize) -> None: ...
    @typing.overload
    def resize(self, w: int, h: int) -> None: ...
    @typing.overload
    def move(self, a0: QtCore.QPoint) -> None: ...
    @typing.overload
    def move(self, ax: int, ay: int) -> None: ...
    def stackUnder(self, a0: typing.Optional['QWidget']) -> None: ...
    def lower(self) -> None: ...
    def raise_(self) -> None: ...
    def close(self) -> bool: ...
    def showNormal(self) -> None: ...
    def showFullScreen(self) -> None: ...
    def showMaximized(self) -> None: ...
    def showMinimized(self) -> None: ...
    def hide(self) -> None: ...
    def show(self) -> None: ...
    def setHidden(self, hidden: bool) -> None: ...
    def setVisible(self, visible: bool) -> None: ...
    @typing.overload
    def repaint(self) -> None: ...
    @typing.overload
    def repaint(self, x: int, y: int, w: int, h: int) -> None: ...
    @typing.overload
    def repaint(self, a0: QtCore.QRect) -> None: ...
    @typing.overload
    def repaint(self, a0: QtGui.QRegion) -> None: ...
    @typing.overload
    def update(self) -> None: ...
    @typing.overload
    def update(self, a0: QtCore.QRect) -> None: ...
    @typing.overload
    def update(self, a0: QtGui.QRegion) -> None: ...
    @typing.overload
    def update(self, ax: int, ay: int, aw: int, ah: int) -> None: ...
    def setUpdatesEnabled(self, enable: bool) -> None: ...
    @staticmethod
    def keyboardGrabber() -> typing.Optional['QWidget']: ...
    @staticmethod
    def mouseGrabber() -> typing.Optional['QWidget']: ...
    def setShortcutEnabled(self, id: int, enabled: bool = ...) -> None: ...
    def releaseShortcut(self, id: int) -> None: ...
    def grabShortcut(self, key: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, typing.Optional[str],
int], context: QtCore.Qt.ShortcutContext = ...) -> int: ...
    def releaseKeyboard(self) -> None: ...
    def grabKeyboard(self) -> None: ...
    def releaseMouse(self) -> None: ...
    @typing.overload
    def grabMouse(self) -> None: ...
    @typing.overload
    def grabMouse(self, a0: typing.Union[QtGui.QCursor, QtCore.Qt.CursorShape]) -> None: ...
    def setContextMenuPolicy(self, policy: QtCore.Qt.ContextMenuPolicy) -> None: ...
    def contextMenuPolicy(self) -> QtCore.Qt.ContextMenuPolicy: ...
    def focusProxy(self) -> typing.Optional['QWidget']: ...
    def setFocusProxy(self, a0: typing.Optional['QWidget']) -> None: ...
    @staticmethod
    def setTabOrder(a0: typing.Optional['QWidget'], a1: typing.Optional['QWidget']) -> None: ...
    def hasFocus(self) -> bool: ...
    def setFocusPolicy(self, policy: QtCore.Qt.FocusPolicy) -> None: ...
    def focusPolicy(self) -> QtCore.Qt.FocusPolicy: ...
    def clearFocus(self) -> None: ...
    def activateWindow(self) -> None: ...
    def isActiveWindow(self) -> bool: ...
    @typing.overload
    def setFocus(self) -> None: ...
    @typing.overload
    def setFocus(self, reason: QtCore.Qt.FocusReason) -> None: ...
    def isLeftToRight(self) -> bool: ...
    def isRightToLeft(self) -> bool: ...
    def unsetLayoutDirection(self) -> None: ...
    def layoutDirection(self) -> QtCore.Qt.LayoutDirection: ...
    def setLayoutDirection(self, direction: QtCore.Qt.LayoutDirection) -> None: ...
    def setAccessibleDescription(self, description: typing.Optional[str]) -> None: ...
    def accessibleDescription(self) -> str: ...
    def setAccessibleName(self, name: typing.Optional[str]) -> None: ...
    def accessibleName(self) -> str: ...
    def whatsThis(self) -> str: ...
    def setWhatsThis(self, a0: typing.Optional[str]) -> None: ...
    def statusTip(self) -> str: ...
    def setStatusTip(self, a0: typing.Optional[str]) -> None: ...
    def toolTip(self) -> str: ...
    def setToolTip(self, a0: typing.Optional[str]) -> None: ...
    def isWindowModified(self) -> bool: ...
    def windowOpacity(self) -> float: ...
    def setWindowOpacity(self, level: float) -> None: ...
    def windowRole(self) -> str: ...
    def setWindowRole(self, a0: typing.Optional[str]) -> None: ...
    def windowIconText(self) -> str: ...
    def setWindowIconText(self, a0: typing.Optional[str]) -> None: ...
    def windowIcon(self) -> QtGui.QIcon: ...
    def setWindowIcon(self, icon: QtGui.QIcon) -> None: ...
    def windowTitle(self) -> str: ...
    def setWindowTitle(self, a0: typing.Optional[str]) -> None: ...
    def clearMask(self) -> None: ...
    def mask(self) -> QtGui.QRegion: ...
    @typing.overload
    def setMask(self, a0: QtGui.QBitmap) -> None: ...
    @typing.overload
    def setMask(self, a0: QtGui.QRegion) -> None: ...
    def unsetCursor(self) -> None: ...
    def setCursor(self, a0: typing.Union[QtGui.QCursor, QtCore.Qt.CursorShape]) -> None: ...
    def cursor(self) -> QtGui.QCursor: ...
    def setFont(self, a0: QtGui.QFont) -> None: ...
    def foregroundRole(self) -> QtGui.QPalette.ColorRole: ...
    def setForegroundRole(self, a0: QtGui.QPalette.ColorRole) -> None: ...
    def backgroundRole(self) -> QtGui.QPalette.ColorRole: ...
    def setBackgroundRole(self, a0: QtGui.QPalette.ColorRole) -> None: ...
    def setPalette(self, a0: QtGui.QPalette) -> None: ...
    def palette(self) -> QtGui.QPalette: ...
    def window(self) -> typing.Optional['QWidget']: ...
    def mapFrom(self, a0: typing.Optional['QWidget'], a1: QtCore.QPoint) -> QtCore.QPoint: ...
    def mapTo(self, a0: typing.Optional['QWidget'], a1: QtCore.QPoint) -> QtCore.QPoint: ...
    def mapFromParent(self, a0: QtCore.QPoint) -> QtCore.QPoint: ...
    def mapToParent(self, a0: QtCore.QPoint) -> QtCore.QPoint: ...
    def mapFromGlobal(self, a0: QtCore.QPoint) -> QtCore.QPoint: ...
    def mapToGlobal(self, a0: QtCore.QPoint) -> QtCore.QPoint: ...
    def setFixedHeight(self, h: int) -> None: ...
    def setFixedWidth(self, w: int) -> None: ...
    @typing.overload
    def setFixedSize(self, a0: QtCore.QSize) -> None: ...
    @typing.overload
    def setFixedSize(self, w: int, h: int) -> None: ...
    @typing.overload
    def setBaseSize(self, basew: int, baseh: int) -> None: ...
    @typing.overload
    def setBaseSize(self, s: QtCore.QSize) -> None: ...
    def baseSize(self) -> QtCore.QSize: ...
    @typing.overload
    def setSizeIncrement(self, w: int, h: int) -> None: ...
    @typing.overload
    def setSizeIncrement(self, s: QtCore.QSize) -> None: ...
    def sizeIncrement(self) -> QtCore.QSize: ...
    def setMaximumHeight(self, maxh: int) -> None: ...
    def setMaximumWidth(self, maxw: int) -> None: ...
    def setMinimumHeight(self, minh: int) -> None: ...
    def setMinimumWidth(self, minw: int) -> None: ...
    @typing.overload
    def setMaximumSize(self, maxw: int, maxh: int) -> None: ...
    @typing.overload
    def setMaximumSize(self, s: QtCore.QSize) -> None: ...
    @typing.overload
    def setMinimumSize(self, minw: int, minh: int) -> None: ...
    @typing.overload
    def setMinimumSize(self, s: QtCore.QSize) -> None: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def childrenRegion(self) -> QtGui.QRegion: ...
    def childrenRect(self) -> QtCore.QRect: ...
    def frameSize(self) -> QtCore.QSize: ...
    def pos(self) -> QtCore.QPoint: ...
    def y(self) -> int: ...
    def x(self) -> int: ...
    def normalGeometry(self) -> QtCore.QRect: ...
    def frameGeometry(self) -> QtCore.QRect: ...
    def setWindowModified(self, a0: bool) -> None: ...
    def setDisabled(self, a0: bool) -> None: ...
    def setEnabled(self, a0: bool) -> None: ...
    def isEnabledTo(self, a0: typing.Optional['QWidget']) -> bool: ...
    def setStyle(self, a0: typing.Optional['QStyle']) -> None: ...
    def style(self) -> typing.Optional['QStyle']: ...
    def devType(self) -> int: ...


class QAbstractButton(QWidget):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def timerEvent(self, e: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def focusOutEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def keyReleaseEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def nextCheckState(self) -> None: ...
    def checkStateSet(self) -> None: ...
    def hitButton(self, pos: QtCore.QPoint) -> bool: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    toggled: typing.ClassVar[QtCore.pyqtSignal]
    clicked: typing.ClassVar[QtCore.pyqtSignal]
    released: typing.ClassVar[QtCore.pyqtSignal]
    pressed: typing.ClassVar[QtCore.pyqtSignal]
    def setChecked(self, a0: bool) -> None: ...
    def toggle(self) -> None: ...
    def click(self) -> None: ...
    def animateClick(self, msecs: int = ...) -> None: ...
    def setIconSize(self, size: QtCore.QSize) -> None: ...
    def group(self) -> typing.Optional['QButtonGroup']: ...
    def autoExclusive(self) -> bool: ...
    def setAutoExclusive(self, a0: bool) -> None: ...
    def autoRepeat(self) -> bool: ...
    def setAutoRepeat(self, a0: bool) -> None: ...
    def isDown(self) -> bool: ...
    def setDown(self, a0: bool) -> None: ...
    def isChecked(self) -> bool: ...
    def isCheckable(self) -> bool: ...
    def setCheckable(self, a0: bool) -> None: ...
    def shortcut(self) -> QtGui.QKeySequence: ...
    def setShortcut(self, key: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, typing.Optional[str], int])
-> None: ...
    def iconSize(self) -> QtCore.QSize: ...
    def icon(self) -> QtGui.QIcon: ...
    def setIcon(self, icon: QtGui.QIcon) -> None: ...
    def text(self) -> str: ...
    def setText(self, text: typing.Optional[str]) -> None: ...
    def autoRepeatInterval(self) -> int: ...
    def setAutoRepeatInterval(self, a0: int) -> None: ...
    def autoRepeatDelay(self) -> int: ...
    def setAutoRepeatDelay(self, a0: int) -> None: ...


class QAbstractItemDelegate(QtCore.QObject):

    class EndEditHint(int):
        NoHint = ... # type: QAbstractItemDelegate.EndEditHint
        EditNextItem = ... # type: QAbstractItemDelegate.EndEditHint
        EditPreviousItem = ... # type: QAbstractItemDelegate.EndEditHint
        SubmitModelCache = ... # type: QAbstractItemDelegate.EndEditHint
        RevertModelCache = ... # type: QAbstractItemDelegate.EndEditHint

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    sizeHintChanged: typing.ClassVar[QtCore.pyqtSignal]
    closeEditor: typing.ClassVar[QtCore.pyqtSignal]
    commitData: typing.ClassVar[QtCore.pyqtSignal]
    def helpEvent(self, event: typing.Optional[QtGui.QHelpEvent], view: typing.Optional['QAbstractItemView'], option:
'QStyleOptionViewItem', index: QtCore.QModelIndex) -> bool: ...
    def editorEvent(self, event: typing.Optional[QtCore.QEvent], model: typing.Optional[QtCore.QAbstractItemModel], option:
'QStyleOptionViewItem', index: QtCore.QModelIndex) -> bool: ...
    def destroyEditor(self, editor: typing.Optional[QWidget], index: QtCore.QModelIndex) -> None: ...
    def updateEditorGeometry(self, editor: typing.Optional[QWidget], option: 'QStyleOptionViewItem', index:
QtCore.QModelIndex) -> None: ...
    def setModelData(self, editor: typing.Optional[QWidget], model: typing.Optional[QtCore.QAbstractItemModel], index:
QtCore.QModelIndex) -> None: ...
    def setEditorData(self, editor: typing.Optional[QWidget], index: QtCore.QModelIndex) -> None: ...
    def createEditor(self, parent: typing.Optional[QWidget], option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) ->
typing.Optional[QWidget]: ...
```

```
    def sizeHint(self, option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> QtCore.QSize: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) ->
None: ...


class QFrame(QWidget):

    class StyleMask(int):
        Shadow_Mask = ... # type: QFrame.StyleMask
        Shape_Mask = ... # type: QFrame.StyleMask

    class Shape(int):
        NoFrame = ... # type: QFrame.Shape
        Box = ... # type: QFrame.Shape
        Panel = ... # type: QFrame.Shape
        WinPanel = ... # type: QFrame.Shape
        HLine = ... # type: QFrame.Shape
        VLine = ... # type: QFrame.Shape
        StyledPanel = ... # type: QFrame.Shape

    class Shadow(int):
        Plain = ... # type: QFrame.Shadow
        Raised = ... # type: QFrame.Shadow
        Sunken = ... # type: QFrame.Shadow

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def initStyleOption(self, option: typing.Optional['QStyleOptionFrame']) -> None: ...
    def drawFrame(self, a0: typing.Optional[QtGui.QPainter]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setFrameRect(self, a0: QtCore.QRect) -> None: ...
    def frameRect(self) -> QtCore.QRect: ...
    def setMidLineWidth(self, a0: int) -> None: ...
    def midLineWidth(self) -> int: ...
    def setLineWidth(self, a0: int) -> None: ...
    def lineWidth(self) -> int: ...
    def setFrameShadow(self, a0: 'QFrame.Shadow') -> None: ...
    def frameShadow(self) -> 'QFrame.Shadow': ...
    def setFrameShape(self, a0: 'QFrame.Shape') -> None: ...
    def frameShape(self) -> 'QFrame.Shape': ...
    def sizeHint(self) -> QtCore.QSize: ...
    def frameWidth(self) -> int: ...
    def setFrameStyle(self, a0: int) -> None: ...
    def frameStyle(self) -> int: ...


class QAbstractScrollArea(QFrame):

    class SizeAdjustPolicy(int):
        AdjustIgnored = ... # type: QAbstractScrollArea.SizeAdjustPolicy
        AdjustToContentsOnFirstShow = ... # type: QAbstractScrollArea.SizeAdjustPolicy
        AdjustToContents = ... # type: QAbstractScrollArea.SizeAdjustPolicy

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def setSizeAdjustPolicy(self, policy: 'QAbstractScrollArea.SizeAdjustPolicy') -> None: ...
    def sizeAdjustPolicy(self) -> 'QAbstractScrollArea.SizeAdjustPolicy': ...
    def setupViewport(self, viewport: typing.Optional[QWidget]) -> None: ...
    def setViewport(self, widget: typing.Optional[QWidget]) -> None: ...
    def scrollBarWidgets(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) ->
typing.List[QWidget]: ...
    def addScrollBarWidget(self, widget: typing.Optional[QWidget], alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag]) -> None: ...
    def setCornerWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def cornerWidget(self) -> typing.Optional[QWidget]: ...
    def setHorizontalScrollBar(self, scrollbar: typing.Optional['QScrollBar']) -> None: ...
    def setVerticalScrollBar(self, scrollbar: typing.Optional['QScrollBar']) -> None: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def dropEvent(self, a0: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragLeaveEvent(self, a0: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragMoveEvent(self, a0: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def dragEnterEvent(self, a0: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
    def contextMenuEvent(self, a0: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def wheelEvent(self, a0: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseDoubleClickEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def viewportEvent(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def viewportSizeHint(self) -> QtCore.QSize: ...
    def viewportMargins(self) -> QtCore.QMargins: ...
    @typing.overload
    def setViewportMargins(self, left: int, top: int, right: int, bottom: int) -> None: ...
    @typing.overload
    def setViewportMargins(self, margins: QtCore.QMargins) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def maximumViewportSize(self) -> QtCore.QSize: ...
    def viewport(self) -> typing.Optional[QWidget]: ...
    def horizontalScrollBar(self) -> typing.Optional['QScrollBar']: ...
    def setHorizontalScrollBarPolicy(self, a0: QtCore.Qt.ScrollBarPolicy) -> None: ...
    def horizontalScrollBarPolicy(self) -> QtCore.Qt.ScrollBarPolicy: ...
    def verticalScrollBar(self) -> typing.Optional['QScrollBar']: ...
    def setVerticalScrollBarPolicy(self, a0: QtCore.Qt.ScrollBarPolicy) -> None: ...
    def verticalScrollBarPolicy(self) -> QtCore.Qt.ScrollBarPolicy: ...


class QAbstractItemView(QAbstractScrollArea):

    class DropIndicatorPosition(int):
        OnItem = ... # type: QAbstractItemView.DropIndicatorPosition
        AboveItem = ... # type: QAbstractItemView.DropIndicatorPosition
        BelowItem = ... # type: QAbstractItemView.DropIndicatorPosition
        OnViewport = ... # type: QAbstractItemView.DropIndicatorPosition

    class State(int):
        NoState = ... # type: QAbstractItemView.State
        DraggingState = ... # type: QAbstractItemView.State
        DragSelectingState = ... # type: QAbstractItemView.State
        EditingState = ... # type: QAbstractItemView.State
        ExpandingState = ... # type: QAbstractItemView.State
        CollapsingState = ... # type: QAbstractItemView.State
        AnimatingState = ... # type: QAbstractItemView.State

    class CursorAction(int):
        MoveUp = ... # type: QAbstractItemView.CursorAction
        MoveDown = ... # type: QAbstractItemView.CursorAction
        MoveLeft = ... # type: QAbstractItemView.CursorAction
        MoveRight = ... # type: QAbstractItemView.CursorAction
        MoveHome = ... # type: QAbstractItemView.CursorAction
        MoveEnd = ... # type: QAbstractItemView.CursorAction
        MovePageUp = ... # type: QAbstractItemView.CursorAction
        MovePageDown = ... # type: QAbstractItemView.CursorAction
        MoveNext = ... # type: QAbstractItemView.CursorAction
        MovePrevious = ... # type: QAbstractItemView.CursorAction

    class SelectionMode(int):
        NoSelection = ... # type: QAbstractItemView.SelectionMode
        SingleSelection = ... # type: QAbstractItemView.SelectionMode
        MultiSelection = ... # type: QAbstractItemView.SelectionMode
        ExtendedSelection = ... # type: QAbstractItemView.SelectionMode
        ContiguousSelection = ... # type: QAbstractItemView.SelectionMode
    class SelectionBehavior(int):
        SelectItems = ... # type: QAbstractItemView.SelectionBehavior
        SelectRows = ... # type: QAbstractItemView.SelectionBehavior
        SelectColumns = ... # type: QAbstractItemView.SelectionBehavior

    class ScrollMode(int):
        ScrollPerItem = ... # type: QAbstractItemView.ScrollMode
        ScrollPerPixel = ... # type: QAbstractItemView.ScrollMode

    class ScrollHint(int):
        EnsureVisible = ... # type: QAbstractItemView.ScrollHint
        PositionAtTop = ... # type: QAbstractItemView.ScrollHint
        PositionAtBottom = ... # type: QAbstractItemView.ScrollHint
        PositionAtCenter = ... # type: QAbstractItemView.ScrollHint

    class EditTrigger(int):
        NoEditTriggers = ... # type: QAbstractItemView.EditTrigger
        CurrentChanged = ... # type: QAbstractItemView.EditTrigger
        DoubleClicked = ... # type: QAbstractItemView.EditTrigger
        SelectedClicked = ... # type: QAbstractItemView.EditTrigger
        EditKeyPressed = ... # type: QAbstractItemView.EditTrigger
        AnyKeyPressed = ... # type: QAbstractItemView.EditTrigger
        AllEditTriggers = ... # type: QAbstractItemView.EditTrigger

    class DragDropMode(int):
        NoDragDrop = ... # type: QAbstractItemView.DragDropMode
        DragOnly = ... # type: QAbstractItemView.DragDropMode
        DropOnly = ... # type: QAbstractItemView.DragDropMode
        DragDrop = ... # type: QAbstractItemView.DragDropMode
        InternalMove = ... # type: QAbstractItemView.DragDropMode

    class EditTriggers(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
'QAbstractItemView.EditTriggers': ...
        def __xor__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
'QAbstractItemView.EditTriggers': ...
        def __ior__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
'QAbstractItemView.EditTriggers': ...
        def __or__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
'QAbstractItemView.EditTriggers': ...
        def __iand__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
'QAbstractItemView.EditTriggers': ...
        def __and__(self, f: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
'QAbstractItemView.EditTriggers': ...
        def __invert__(self) -> 'QAbstractItemView.EditTriggers': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isPersistentEditorOpen(self, index: QtCore.QModelIndex) -> bool: ...
    def resetHorizontalScrollMode(self) -> None: ...
    def resetVerticalScrollMode(self) -> None: ...
    def defaultDropAction(self) -> QtCore.Qt.DropAction: ...
    def setDefaultDropAction(self, dropAction: QtCore.Qt.DropAction) -> None: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def viewportSizeHint(self) -> QtCore.QSize: ...
    def inputMethodEvent(self, event: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def autoScrollMargin(self) -> int: ...
    def setAutoScrollMargin(self, margin: int) -> None: ...
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    def itemDelegateForColumn(self, column: int) -> typing.Optional[QAbstractItemDelegate]: ...
    def setItemDelegateForColumn(self, column: int, delegate: typing.Optional[QAbstractItemDelegate]) -> None: ...
    def itemDelegateForRow(self, row: int) -> typing.Optional[QAbstractItemDelegate]: ...
    def setItemDelegateForRow(self, row: int, delegate: typing.Optional[QAbstractItemDelegate]) -> None: ...
    def dragDropMode(self) -> 'QAbstractItemView.DragDropMode': ...
    def setDragDropMode(self, behavior: 'QAbstractItemView.DragDropMode') -> None: ...
    def dragDropOverwriteMode(self) -> bool: ...
    def setDragDropOverwriteMode(self, overwrite: bool) -> None: ...
    def horizontalScrollMode(self) -> 'QAbstractItemView.ScrollMode': ...
    def setHorizontalScrollMode(self, mode: 'QAbstractItemView.ScrollMode') -> None: ...
    def verticalScrollMode(self) -> 'QAbstractItemView.ScrollMode': ...
    def setVerticalScrollMode(self, mode: 'QAbstractItemView.ScrollMode') -> None: ...
    def dropIndicatorPosition(self) -> 'QAbstractItemView.DropIndicatorPosition': ...
    def timerEvent(self, e: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def focusOutEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def dropEvent(self, e: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragLeaveEvent(self, e: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragMoveEvent(self, e: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def dragEnterEvent(self, e: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
    def mouseDoubleClickEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def viewportEvent(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def dirtyRegionOffset(self) -> QtCore.QPoint: ...
    def setDirtyRegion(self, region: QtGui.QRegion) -> None: ...
    def scrollDirtyRegion(self, dx: int, dy: int) -> None: ...
    def executeDelayedItemsLayout(self) -> None: ...
    def scheduleDelayedItemsLayout(self) -> None: ...
    def setState(self, state: 'QAbstractItemView.State') -> None: ...
    def state(self) -> 'QAbstractItemView.State': ...
    def viewOptions(self) -> 'QStyleOptionViewItem': ...
    def startDrag(self, supportedActions: typing.Union[QtCore.Qt.DropActions, QtCore.Qt.DropAction]) -> None: ...
    def selectionCommand(self, index: QtCore.QModelIndex, event: typing.Optional[QtCore.QEvent] = ...) ->
QtCore.QItemSelectionModel.SelectionFlags: ...
    def selectedIndexes(self) -> typing.List[QtCore.QModelIndex]: ...
    def visualRegionForSelection(self, selection: QtCore.QItemSelection) -> QtGui.QRegion: ...
    def setSelection(self, rect: QtCore.QRect, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def isIndexHidden(self, index: QtCore.QModelIndex) -> bool: ...
    def verticalOffset(self) -> int: ...
    def horizontalOffset(self) -> int: ...
    def moveCursor(self, cursorAction: 'QAbstractItemView.CursorAction', modifiers:
typing.Union[QtCore.Qt.KeyboardModifiers, QtCore.Qt.KeyboardModifier]) -> QtCore.QModelIndex: ...
    iconSizeChanged: typing.ClassVar[QtCore.pyqtSignal]
    viewportEntered: typing.ClassVar[QtCore.pyqtSignal]
    entered: typing.ClassVar[QtCore.pyqtSignal]
    activated: typing.ClassVar[QtCore.pyqtSignal]
    doubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    clicked: typing.ClassVar[QtCore.pyqtSignal]
    pressed: typing.ClassVar[QtCore.pyqtSignal]
    def editorDestroyed(self, editor: typing.Optional[QtCore.QObject]) -> None: ...
    def commitData(self, editor: typing.Optional[QWidget]) -> None: ...
    def closeEditor(self, editor: typing.Optional[QWidget], hint: QAbstractItemDelegate.EndEditHint) -> None: ...
    def horizontalScrollbarValueChanged(self, value: int) -> None: ...
    def verticalScrollbarValueChanged(self, value: int) -> None: ...
    def horizontalScrollbarAction(self, action: int) -> None: ...
    def verticalScrollbarAction(self, action: int) -> None: ...
    def updateGeometries(self) -> None: ...
    def updateEditorGeometries(self) -> None: ...
    def updateEditorData(self) -> None: ...
    def currentChanged(self, current: QtCore.QModelIndex, previous: QtCore.QModelIndex) -> None: ...
    def selectionChanged(self, selected: QtCore.QItemSelection, deselected: QtCore.QItemSelection) -> None: ...
    def rowsAboutToBeRemoved(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def rowsInserted(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def dataChanged(self, topLeft: QtCore.QModelIndex, bottomRight: QtCore.QModelIndex, roles: typing.Iterable[int] = ...) -
> None: ...
    @typing.overload
    def update(self) -> None: ...
    @typing.overload
    def update(self, index: QtCore.QModelIndex) -> None: ...
    def scrollToBottom(self) -> None: ...
    def scrollToTop(self) -> None: ...
    def setCurrentIndex(self, index: QtCore.QModelIndex) -> None: ...
    def clearSelection(self) -> None: ...
    @typing.overload
    def edit(self, index: QtCore.QModelIndex) -> None: ...
    @typing.overload
    def edit(self, index: QtCore.QModelIndex, trigger: 'QAbstractItemView.EditTrigger', event: typing.Optional[QtCore.QEvent])
-> bool: ...
    def selectAll(self) -> None: ...
    def setRootIndex(self, index: QtCore.QModelIndex) -> None: ...
    def reset(self) -> None: ...
    def indexWidget(self, index: QtCore.QModelIndex) -> typing.Optional[QWidget]: ...
    def setIndexWidget(self, index: QtCore.QModelIndex, widget: typing.Optional[QWidget]) -> None: ...
    def closePersistentEditor(self, index: QtCore.QModelIndex) -> None: ...
    def openPersistentEditor(self, index: QtCore.QModelIndex) -> None: ...
    def sizeHintForColumn(self, column: int) -> int: ...
    def sizeHintForRow(self, row: int) -> int: ...
    def sizeHintForIndex(self, index: QtCore.QModelIndex) -> QtCore.QSize: ...
    def indexAt(self, p: QtCore.QPoint) -> QtCore.QModelIndex: ...
    def scrollTo(self, index: QtCore.QModelIndex, hint: 'QAbstractItemView.ScrollHint' = ...) -> None: ...
    def visualRect(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def keyboardSearch(self, search: typing.Optional[str]) -> None: ...
    def textElideMode(self) -> QtCore.Qt.TextElideMode: ...
    def setTextElideMode(self, mode: QtCore.Qt.TextElideMode) -> None: ...
    def iconSize(self) -> QtCore.QSize: ...
    def setIconSize(self, size: QtCore.QSize) -> None: ...
    def alternatingRowColors(self) -> bool: ...
    def setAlternatingRowColors(self, enable: bool) -> None: ...
    def dragEnabled(self) -> bool: ...
    def setDragEnabled(self, enable: bool) -> None: ...
    def showDropIndicator(self) -> bool: ...
    def setDropIndicatorShown(self, enable: bool) -> None: ...
    def tabKeyNavigation(self) -> bool: ...
    def setTabKeyNavigation(self, enable: bool) -> None: ...
    def hasAutoScroll(self) -> bool: ...
    def setAutoScroll(self, enable: bool) -> None: ...
    def editTriggers(self) -> 'QAbstractItemView.EditTriggers': ...
    def setEditTriggers(self, triggers: typing.Union['QAbstractItemView.EditTriggers', 'QAbstractItemView.EditTrigger']) ->
None: ...
    def rootIndex(self) -> QtCore.QModelIndex: ...
    def currentIndex(self) -> QtCore.QModelIndex: ...
    def selectionBehavior(self) -> 'QAbstractItemView.SelectionBehavior': ...
    def setSelectionBehavior(self, behavior: 'QAbstractItemView.SelectionBehavior') -> None: ...
    def selectionMode(self) -> 'QAbstractItemView.SelectionMode': ...
    def setSelectionMode(self, mode: 'QAbstractItemView.SelectionMode') -> None: ...
    @typing.overload
    def itemDelegate(self) -> typing.Optional[QAbstractItemDelegate]: ...
    @typing.overload
    def itemDelegate(self, index: QtCore.QModelIndex) -> typing.Optional[QAbstractItemDelegate]: ...
    def setItemDelegate(self, delegate: typing.Optional[QAbstractItemDelegate]) -> None: ...
    def selectionModel(self) -> typing.Optional[QtCore.QItemSelectionModel]: ...
    def setSelectionModel(self, selectionModel: typing.Optional[QtCore.QItemSelectionModel]) -> None: ...
    def model(self) -> typing.Optional[QtCore.QAbstractItemModel]: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...


class QAbstractSlider(QWidget):

    class SliderChange(int):
        SliderRangeChange = ... # type: QAbstractSlider.SliderChange
        SliderOrientationChange = ... # type: QAbstractSlider.SliderChange
        SliderStepsChange = ... # type: QAbstractSlider.SliderChange
        SliderValueChange = ... # type: QAbstractSlider.SliderChange

    class SliderAction(int):
        SliderNoAction = ... # type: QAbstractSlider.SliderAction
        SliderSingleStepAdd = ... # type: QAbstractSlider.SliderAction
        SliderSingleStepSub = ... # type: QAbstractSlider.SliderAction
        SliderPageStepAdd = ... # type: QAbstractSlider.SliderAction
        SliderPageStepSub = ... # type: QAbstractSlider.SliderAction
        SliderToMinimum = ... # type: QAbstractSlider.SliderAction
        SliderToMaximum = ... # type: QAbstractSlider.SliderAction
        SliderMove = ... # type: QAbstractSlider.SliderAction

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def timerEvent(self, a0: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def keyPressEvent(self, ev: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def sliderChange(self, change: 'QAbstractSlider.SliderChange') -> None: ...
    def repeatAction(self) -> 'QAbstractSlider.SliderAction': ...
    def setRepeatAction(self, action: 'QAbstractSlider.SliderAction', thresholdTime: int = ..., repeatTime: int = ...) -> None: ...
    actionTriggered: typing.ClassVar[QtCore.pyqtSignal]
    rangeChanged: typing.ClassVar[QtCore.pyqtSignal]
    sliderReleased: typing.ClassVar[QtCore.pyqtSignal]
    sliderMoved: typing.ClassVar[QtCore.pyqtSignal]
    sliderPressed: typing.ClassVar[QtCore.pyqtSignal]
    valueChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setOrientation(self, a0: QtCore.Qt.Orientation) -> None: ...
    def setValue(self, a0: int) -> None: ...
    def triggerAction(self, action: 'QAbstractSlider.SliderAction') -> None: ...
    def value(self) -> int: ...
    def invertedControls(self) -> bool: ...
    def setInvertedControls(self, a0: bool) -> None: ...
    def invertedAppearance(self) -> bool: ...
    def setInvertedAppearance(self, a0: bool) -> None: ...
    def sliderPosition(self) -> int: ...
    def setSliderPosition(self, a0: int) -> None: ...
    def isSliderDown(self) -> bool: ...
    def setSliderDown(self, a0: bool) -> None: ...
    def hasTracking(self) -> bool: ...
    def setTracking(self, enable: bool) -> None: ...
    def pageStep(self) -> int: ...
    def setPageStep(self, a0: int) -> None: ...
    def singleStep(self) -> int: ...
    def setSingleStep(self, a0: int) -> None: ...
    def setRange(self, min: int, max: int) -> None: ...
    def maximum(self) -> int: ...
    def setMaximum(self, a0: int) -> None: ...
    def minimum(self) -> int: ...
    def setMinimum(self, a0: int) -> None: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...


class QAbstractSpinBox(QWidget):

    class StepType(int):
        DefaultStepType = ... # type: QAbstractSpinBox.StepType
        AdaptiveDecimalStepType = ... # type: QAbstractSpinBox.StepType

    class CorrectionMode(int):
        CorrectToPreviousValue = ... # type: QAbstractSpinBox.CorrectionMode
        CorrectToNearestValue = ... # type: QAbstractSpinBox.CorrectionMode

    class ButtonSymbols(int):
        UpDownArrows = ... # type: QAbstractSpinBox.ButtonSymbols
        PlusMinus = ... # type: QAbstractSpinBox.ButtonSymbols
        NoButtons = ... # type: QAbstractSpinBox.ButtonSymbols
    class StepEnabledFlag(int):
        StepNone = ... # type: QAbstractSpinBox.StepEnabledFlag
        StepUpEnabled = ... # type: QAbstractSpinBox.StepEnabledFlag
        StepDownEnabled = ... # type: QAbstractSpinBox.StepEnabledFlag

    class StepEnabled(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) ->
'QAbstractSpinBox.StepEnabled': ...
        def __xor__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) ->
'QAbstractSpinBox.StepEnabled': ...
        def __ior__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) ->
'QAbstractSpinBox.StepEnabled': ...
        def __or__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) ->
'QAbstractSpinBox.StepEnabled': ...
        def __iand__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) ->
'QAbstractSpinBox.StepEnabled': ...
        def __and__(self, f: typing.Union['QAbstractSpinBox.StepEnabled', 'QAbstractSpinBox.StepEnabledFlag']) ->
'QAbstractSpinBox.StepEnabled': ...
        def __invert__(self) -> 'QAbstractSpinBox.StepEnabled': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isGroupSeparatorShown(self) -> bool: ...
    def setGroupSeparatorShown(self, shown: bool) -> None: ...
    def inputMethodQuery(self, a0: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    def keyboardTracking(self) -> bool: ...
    def setKeyboardTracking(self, kt: bool) -> None: ...
    def isAccelerated(self) -> bool: ...
    def setAccelerated(self, on: bool) -> None: ...
    def hasAcceptableInput(self) -> bool: ...
    def correctionMode(self) -> 'QAbstractSpinBox.CorrectionMode': ...
    def setCorrectionMode(self, cm: 'QAbstractSpinBox.CorrectionMode') -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionSpinBox']) -> None: ...
    def stepEnabled(self) -> 'QAbstractSpinBox.StepEnabled': ...
    def setLineEdit(self, e: typing.Optional['QLineEdit']) -> None: ...
    def lineEdit(self) -> typing.Optional['QLineEdit']: ...
    def showEvent(self, e: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def timerEvent(self, e: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def hideEvent(self, e: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def closeEvent(self, e: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def contextMenuEvent(self, e: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def focusOutEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def keyReleaseEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    editingFinished: typing.ClassVar[QtCore.pyqtSignal]
    def clear(self) -> None: ...
    def selectAll(self) -> None: ...
    def stepDown(self) -> None: ...
    def stepUp(self) -> None: ...
    def stepBy(self, steps: int) -> None: ...
    def fixup(self, input: typing.Optional[str]) -> str: ...
    def validate(self, input: typing.Optional[str], pos: int) -> typing.Tuple[QtGui.QValidator.State, str, int]: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def interpretText(self) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def hasFrame(self) -> bool: ...
    def setFrame(self, a0: bool) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def setAlignment(self, flag: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def isReadOnly(self) -> bool: ...
    def setReadOnly(self, r: bool) -> None: ...
    def setWrapping(self, w: bool) -> None: ...
    def wrapping(self) -> bool: ...
    def setSpecialValueText(self, s: typing.Optional[str]) -> None: ...
    def specialValueText(self) -> str: ...
    def text(self) -> str: ...
    def setButtonSymbols(self, bs: 'QAbstractSpinBox.ButtonSymbols') -> None: ...
    def buttonSymbols(self) -> 'QAbstractSpinBox.ButtonSymbols': ...


class QAction(QtCore.QObject):

    class Priority(int):
        LowPriority = ... # type: QAction.Priority
        NormalPriority = ... # type: QAction.Priority
        HighPriority = ... # type: QAction.Priority

    class MenuRole(int):
        NoRole = ... # type: QAction.MenuRole
        TextHeuristicRole = ... # type: QAction.MenuRole
        ApplicationSpecificRole = ... # type: QAction.MenuRole
        AboutQtRole = ... # type: QAction.MenuRole
        AboutRole = ... # type: QAction.MenuRole
        PreferencesRole = ... # type: QAction.MenuRole
        QuitRole = ... # type: QAction.MenuRole

    class ActionEvent(int):
        Trigger = ... # type: QAction.ActionEvent
        Hover = ... # type: QAction.ActionEvent

    @typing.overload
    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QtCore.QObject] = ...) -> None: ...
    @typing.overload
    def __init__(self, icon: QtGui.QIcon, text: typing.Optional[str], parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def isShortcutVisibleInContextMenu(self) -> bool: ...
    def setShortcutVisibleInContextMenu(self, show: bool) -> None: ...
    def priority(self) -> 'QAction.Priority': ...
    def setPriority(self, priority: 'QAction.Priority') -> None: ...
    def isIconVisibleInMenu(self) -> bool: ...
    def setIconVisibleInMenu(self, visible: bool) -> None: ...
    def associatedGraphicsWidgets(self) -> typing.List['QGraphicsWidget']: ...
    def associatedWidgets(self) -> typing.List[QWidget]: ...
    def menuRole(self) -> 'QAction.MenuRole': ...
    def setMenuRole(self, menuRole: 'QAction.MenuRole') -> None: ...
    def autoRepeat(self) -> bool: ...
    def setAutoRepeat(self, a0: bool) -> None: ...
    def shortcuts(self) -> typing.List[QtGui.QKeySequence]: ...
    @typing.overload
    def setShortcuts(self, shortcuts: typing.Iterable[typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey,
typing.Optional[str], int]]) -> None: ...
    @typing.overload
    def setShortcuts(self, a0: QtGui.QKeySequence.StandardKey) -> None: ...
    toggled: typing.ClassVar[QtCore.pyqtSignal]
    hovered: typing.ClassVar[QtCore.pyqtSignal]
    triggered: typing.ClassVar[QtCore.pyqtSignal]
    changed: typing.ClassVar[QtCore.pyqtSignal]
    def setVisible(self, a0: bool) -> None: ...
    def setDisabled(self, b: bool) -> None: ...
    def setEnabled(self, a0: bool) -> None: ...
    def toggle(self) -> None: ...
    def setChecked(self, a0: bool) -> None: ...
    def hover(self) -> None: ...
    def trigger(self) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def parentWidget(self) -> typing.Optional[QWidget]: ...
    def showStatusText(self, widget: typing.Optional[QWidget] = ...) -> bool: ...
    def activate(self, event: 'QAction.ActionEvent') -> None: ...
    def isVisible(self) -> bool: ...
    def isEnabled(self) -> bool: ...
    def isChecked(self) -> bool: ...
    def setData(self, var: typing.Any) -> None: ...
    def data(self) -> typing.Any: ...
    def isCheckable(self) -> bool: ...
    def setCheckable(self, a0: bool) -> None: ...
    def font(self) -> QtGui.QFont: ...
    def setFont(self, font: QtGui.QFont) -> None: ...
    def shortcutContext(self) -> QtCore.Qt.ShortcutContext: ...
    def setShortcutContext(self, context: QtCore.Qt.ShortcutContext) -> None: ...
    def shortcut(self) -> QtGui.QKeySequence: ...
    def setShortcut(self, shortcut: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, typing.Optional[str],
int]) -> None: ...
    def isSeparator(self) -> bool: ...
    def setSeparator(self, b: bool) -> None: ...
    def setMenu(self, menu: typing.Optional['QMenu']) -> None: ...
    def menu(self) -> typing.Optional['QMenu']: ...
    def whatsThis(self) -> str: ...
    def setWhatsThis(self, what: typing.Optional[str]) -> None: ...
    def statusTip(self) -> str: ...
    def setStatusTip(self, statusTip: typing.Optional[str]) -> None: ...
    def toolTip(self) -> str: ...
    def setToolTip(self, tip: typing.Optional[str]) -> None: ...
    def iconText(self) -> str: ...
    def setIconText(self, text: typing.Optional[str]) -> None: ...
    def text(self) -> str: ...
    def setText(self, text: typing.Optional[str]) -> None: ...
    def icon(self) -> QtGui.QIcon: ...
    def setIcon(self, icon: QtGui.QIcon) -> None: ...
    def actionGroup(self) -> typing.Optional['QActionGroup']: ...
    def setActionGroup(self, group: typing.Optional['QActionGroup']) -> None: ...


class QActionGroup(QtCore.QObject):

    class ExclusionPolicy(int):
        None_ = ... # type: QActionGroup.ExclusionPolicy
        Exclusive = ... # type: QActionGroup.ExclusionPolicy
        ExclusiveOptional = ... # type: QActionGroup.ExclusionPolicy

    def __init__(self, parent: typing.Optional[QtCore.QObject]) -> None: ...

    def setExclusionPolicy(self, policy: 'QActionGroup.ExclusionPolicy') -> None: ...
    def exclusionPolicy(self) -> 'QActionGroup.ExclusionPolicy': ...
    hovered: typing.ClassVar[QtCore.pyqtSignal]
    triggered: typing.ClassVar[QtCore.pyqtSignal]
    def setExclusive(self, a0: bool) -> None: ...
    def setVisible(self, a0: bool) -> None: ...
    def setDisabled(self, b: bool) -> None: ...
    def setEnabled(self, a0: bool) -> None: ...
    def isVisible(self) -> bool: ...
    def isEnabled(self) -> bool: ...
    def isExclusive(self) -> bool: ...
    def checkedAction(self) -> typing.Optional[QAction]: ...
    def actions(self) -> typing.List[QAction]: ...
    def removeAction(self, a: typing.Optional[QAction]) -> None: ...
    @typing.overload
    def addAction(self, a: typing.Optional[QAction]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, icon: QtGui.QIcon, text: typing.Optional[str]) -> typing.Optional[QAction]: ...


class QApplication(QtGui.QGuiApplication):

    class ColorSpec(int):
        NormalColor = ... # type: QApplication.ColorSpec
        CustomColor = ... # type: QApplication.ColorSpec
        ManyColor = ... # type: QApplication.ColorSpec

    def __init__(self, argv: typing.List[str]) -> None: ...

    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setStyleSheet(self, sheet: typing.Optional[str]) -> None: ...
    def setAutoSipEnabled(self, enabled: bool) -> None: ...
    @staticmethod
    def closeAllWindows() -> None: ...
    @staticmethod
    def aboutQt() -> None: ...
    focusChanged: typing.ClassVar[QtCore.pyqtSignal]
    def styleSheet(self) -> str: ...
    def autoSipEnabled(self) -> bool: ...
    def notify(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    @staticmethod
    def exec() -> int: ...
    @staticmethod
    def exec_() -> int: ...
    @staticmethod
    def setEffectEnabled(a0: QtCore.Qt.UIEffect, enabled: bool = ...) -> None: ...
    @staticmethod
    def isEffectEnabled(a0: QtCore.Qt.UIEffect) -> bool: ...
    @staticmethod
    def startDragDistance() -> int: ...
    @staticmethod
    def setStartDragDistance(l: int) -> None: ...
    @staticmethod
    def startDragTime() -> int: ...
    @staticmethod
    def setStartDragTime(ms: int) -> None: ...
    @staticmethod
    def globalStrut() -> QtCore.QSize: ...
    @staticmethod
    def setGlobalStrut(a0: QtCore.QSize) -> None: ...
    @staticmethod
    def wheelScrollLines() -> int: ...
    @staticmethod
    def setWheelScrollLines(a0: int) -> None: ...
    @staticmethod
    def keyboardInputInterval() -> int: ...
    @staticmethod
    def setKeyboardInputInterval(a0: int) -> None: ...
    @staticmethod
    def doubleClickInterval() -> int: ...
    @staticmethod
    def setDoubleClickInterval(a0: int) -> None: ...
    @staticmethod
    def cursorFlashTime() -> int: ...
    @staticmethod
    def setCursorFlashTime(a0: int) -> None: ...
    @staticmethod
    def alert(widget: typing.Optional[QWidget], msecs: int = ...) -> None: ...
    @staticmethod
    def beep() -> None: ...
    @typing.overload
    @staticmethod
    def topLevelAt(p: QtCore.QPoint) -> typing.Optional[QWidget]: ...
    @typing.overload
    @staticmethod
    def topLevelAt(x: int, y: int) -> typing.Optional[QWidget]: ...
    @typing.overload
    @staticmethod
    def widgetAt(p: QtCore.QPoint) -> typing.Optional[QWidget]: ...
    @typing.overload
    @staticmethod
    def widgetAt(x: int, y: int) -> typing.Optional[QWidget]: ...
    @staticmethod
    def setActiveWindow(act: typing.Optional[QWidget]) -> None: ...
    @staticmethod
    def activeWindow() -> typing.Optional[QWidget]: ...
    @staticmethod
    def focusWidget() -> typing.Optional[QWidget]: ...
    @staticmethod
    def activeModalWidget() -> typing.Optional[QWidget]: ...
    @staticmethod
    def activePopupWidget() -> typing.Optional[QWidget]: ...
    @staticmethod
    def desktop() -> typing.Optional['QDesktopWidget']: ...
    @staticmethod
    def topLevelWidgets() -> typing.List[QWidget]: ...
    @staticmethod
    def allWidgets() -> typing.List[QWidget]: ...
    @staticmethod
    def windowIcon() -> QtGui.QIcon: ...
    @staticmethod
    def setWindowIcon(icon: QtGui.QIcon) -> None: ...
    @staticmethod
    def fontMetrics() -> QtGui.QFontMetrics: ...
    @staticmethod
    def setFont(a0: QtGui.QFont, className: typing.Optional[str] = ...) -> None: ...
    @typing.overload
    @staticmethod
    def font() -> QtGui.QFont: ...
    @typing.overload
    @staticmethod
    def font(a0: typing.Optional[QWidget]) -> QtGui.QFont: ...
    @typing.overload
    @staticmethod
    def font(className: typing.Optional[str]) -> QtGui.QFont: ...
    @staticmethod
    def setPalette(a0: QtGui.QPalette, className: typing.Optional[str] = ...) -> None: ...
    @typing.overload
    @staticmethod
    def palette() -> QtGui.QPalette: ...
    @typing.overload
    @staticmethod
    def palette(a0: typing.Optional[QWidget]) -> QtGui.QPalette: ...
    @typing.overload
    @staticmethod
    def palette(className: typing.Optional[str]) -> QtGui.QPalette: ...
    @staticmethod
    def setColorSpec(a0: int) -> None: ...
    @staticmethod
    def colorSpec() -> int: ...
    @typing.overload
    @staticmethod
    def setStyle(a0: typing.Optional['QStyle']) -> None: ...
    @typing.overload
    @staticmethod
    def setStyle(a0: typing.Optional[str]) -> typing.Optional['QStyle']: ...
    @staticmethod
    def style() -> typing.Optional['QStyle']: ...


class QLayoutItem(PyQt5.sip.wrapper):

    @typing.overload
    def __init__(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QLayoutItem') -> None: ...

    def controlTypes(self) -> 'QSizePolicy.ControlTypes': ...
    def setAlignment(self, a: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def spacerItem(self) -> typing.Optional['QSpacerItem']: ...
    def layout(self) -> typing.Optional['QLayout']: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    def invalidate(self) -> None: ...
    def minimumHeightForWidth(self, a0: int) -> int: ...
    def heightForWidth(self, a0: int) -> int: ...
    def hasHeightForWidth(self) -> bool: ...
    def isEmpty(self) -> bool: ...
    def geometry(self) -> QtCore.QRect: ...
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QLayout(QtCore.QObject, QLayoutItem):

    class SizeConstraint(int):
        SetDefaultConstraint = ... # type: QLayout.SizeConstraint
        SetNoConstraint = ... # type: QLayout.SizeConstraint
        SetMinimumSize = ... # type: QLayout.SizeConstraint
        SetFixedSize = ... # type: QLayout.SizeConstraint
        SetMaximumSize = ... # type: QLayout.SizeConstraint
        SetMinAndMaxSize = ... # type: QLayout.SizeConstraint

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def __init__(self) -> None: ...

    def replaceWidget(self, from_: typing.Optional[QWidget], to: typing.Optional[QWidget], options:
typing.Union[QtCore.Qt.FindChildOptions, QtCore.Qt.FindChildOption] = ...) -> typing.Optional[QLayoutItem]: ...
    def controlTypes(self) -> 'QSizePolicy.ControlTypes': ...
    def contentsMargins(self) -> QtCore.QMargins: ...
    def contentsRect(self) -> QtCore.QRect: ...
    def getContentsMargins(self) -> typing.Tuple[typing.Optional[int], typing.Optional[int], typing.Optional[int],
typing.Optional[int]]: ...
    @typing.overload
    def setContentsMargins(self, left: int, top: int, right: int, bottom: int) -> None: ...
    @typing.overload
    def setContentsMargins(self, margins: QtCore.QMargins) -> None: ...
    def alignmentRect(self, a0: QtCore.QRect) -> QtCore.QRect: ...
    def addChildWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def addChildLayout(self, l: typing.Optional['QLayout']) -> None: ...
    def childEvent(self, e: typing.Optional[QtCore.QChildEvent]) -> None: ...
    def widgetEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    @staticmethod
    def closestAcceptableSize(w: typing.Optional[QWidget], s: QtCore.QSize) -> QtCore.QSize: ...
    def isEnabled(self) -> bool: ...
    def setEnabled(self, a0: bool) -> None: ...
    def layout(self) -> typing.Optional['QLayout']: ...
    def totalSizeHint(self) -> QtCore.QSize: ...
    def totalMaximumSize(self) -> QtCore.QSize: ...
    def totalMinimumSize(self) -> QtCore.QSize: ...
    def totalHeightForWidth(self, w: int) -> int: ...
    def isEmpty(self) -> bool: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    @typing.overload
    def indexOf(self, a0: typing.Optional[QWidget]) -> int: ...
    @typing.overload
    def indexOf(self, a0: typing.Optional[QLayoutItem]) -> int: ...
    def takeAt(self, index: int) -> typing.Optional[QLayoutItem]: ...
    def itemAt(self, index: int) -> typing.Optional[QLayoutItem]: ...
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def removeItem(self, a0: typing.Optional[QLayoutItem]) -> None: ...
    def removeWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def addItem(self, a0: typing.Optional[QLayoutItem]) -> None: ...
    def addWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def update(self) -> None: ...
    def activate(self) -> bool: ...
    def geometry(self) -> QtCore.QRect: ...
    def invalidate(self) -> None: ...
    def parentWidget(self) -> typing.Optional[QWidget]: ...
    def menuBar(self) -> typing.Optional[QWidget]: ...
    def setMenuBar(self, w: typing.Optional[QWidget]) -> None: ...
    def sizeConstraint(self) -> 'QLayout.SizeConstraint': ...
    def setSizeConstraint(self, a0: 'QLayout.SizeConstraint') -> None: ...
    @typing.overload
    def setAlignment(self, w: typing.Optional[QWidget], alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag]) -> bool: ...
    @typing.overload
    def setAlignment(self, l: typing.Optional['QLayout'], alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag]) -> bool: ...
    @typing.overload
    def setAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def setSpacing(self, a0: int) -> None: ...
    def spacing(self) -> int: ...


class QBoxLayout(QLayout):

    class Direction(int):
        LeftToRight = ... # type: QBoxLayout.Direction
        RightToLeft = ... # type: QBoxLayout.Direction
        TopToBottom = ... # type: QBoxLayout.Direction
        BottomToTop = ... # type: QBoxLayout.Direction
        Down = ... # type: QBoxLayout.Direction
        Up = ... # type: QBoxLayout.Direction

    def __init__(self, direction: 'QBoxLayout.Direction', parent: typing.Optional[QWidget] = ...) -> None: ...

    def insertItem(self, index: int, a1: typing.Optional[QLayoutItem]) -> None: ...
    def stretch(self, index: int) -> int: ...
    def setStretch(self, index: int, stretch: int) -> None: ...
    def insertSpacerItem(self, index: int, spacerItem: typing.Optional['QSpacerItem']) -> None: ...
    def addSpacerItem(self, spacerItem: typing.Optional['QSpacerItem']) -> None: ...
    def setSpacing(self, spacing: int) -> None: ...
    def spacing(self) -> int: ...
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    def count(self) -> int: ...
    def takeAt(self, a0: int) -> typing.Optional[QLayoutItem]: ...
    def itemAt(self, a0: int) -> typing.Optional[QLayoutItem]: ...
    def invalidate(self) -> None: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def minimumHeightForWidth(self, a0: int) -> int: ...
    def heightForWidth(self, a0: int) -> int: ...
    def hasHeightForWidth(self) -> bool: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    @typing.overload
    def setStretchFactor(self, w: typing.Optional[QWidget], stretch: int) -> bool: ...
    @typing.overload
    def setStretchFactor(self, l: typing.Optional[QLayout], stretch: int) -> bool: ...
    def insertLayout(self, index: int, layout: typing.Optional[QLayout], stretch: int = ...) -> None: ...
    def insertWidget(self, index: int, widget: typing.Optional[QWidget], stretch: int = ..., alignment:
typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    def insertStretch(self, index: int, stretch: int = ...) -> None: ...
    def insertSpacing(self, index: int, size: int) -> None: ...
    def addItem(self, a0: typing.Optional[QLayoutItem]) -> None: ...
    def addStrut(self, a0: int) -> None: ...
    def addLayout(self, layout: typing.Optional[QLayout], stretch: int = ...) -> None: ...
    def addWidget(self, a0: typing.Optional[QWidget], stretch: int = ..., alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    def addStretch(self, stretch: int = ...) -> None: ...
    def addSpacing(self, size: int) -> None: ...
    def setDirection(self, a0: 'QBoxLayout.Direction') -> None: ...
    def direction(self) -> 'QBoxLayout.Direction': ...


class QHBoxLayout(QBoxLayout):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...


class QVBoxLayout(QBoxLayout):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...


class QButtonGroup(QtCore.QObject):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    idToggled: typing.ClassVar[QtCore.pyqtSignal]
    idReleased: typing.ClassVar[QtCore.pyqtSignal]
    idPressed: typing.ClassVar[QtCore.pyqtSignal]
    idClicked: typing.ClassVar[QtCore.pyqtSignal]
    buttonToggled: typing.ClassVar[QtCore.pyqtSignal]
    buttonReleased: typing.ClassVar[QtCore.pyqtSignal]
    buttonPressed: typing.ClassVar[QtCore.pyqtSignal]
    buttonClicked: typing.ClassVar[QtCore.pyqtSignal]
    def checkedId(self) -> int: ...
    def id(self, button: typing.Optional[QAbstractButton]) -> int: ...
    def setId(self, button: typing.Optional[QAbstractButton], id: int) -> None: ...
    def checkedButton(self) -> typing.Optional[QAbstractButton]: ...
    def button(self, id: int) -> typing.Optional[QAbstractButton]: ...
    def buttons(self) -> typing.List[QAbstractButton]: ...
    def removeButton(self, a0: typing.Optional[QAbstractButton]) -> None: ...
    def addButton(self, a0: typing.Optional[QAbstractButton], id: int = ...) -> None: ...
    def exclusive(self) -> bool: ...
    def setExclusive(self, a0: bool) -> None: ...


class QCalendarWidget(QWidget):

    class SelectionMode(int):
        NoSelection = ... # type: QCalendarWidget.SelectionMode
        SingleSelection = ... # type: QCalendarWidget.SelectionMode

    class VerticalHeaderFormat(int):
        NoVerticalHeader = ... # type: QCalendarWidget.VerticalHeaderFormat
        ISOWeekNumbers = ... # type: QCalendarWidget.VerticalHeaderFormat

    class HorizontalHeaderFormat(int):
        NoHorizontalHeader = ... # type: QCalendarWidget.HorizontalHeaderFormat
        SingleLetterDayNames = ... # type: QCalendarWidget.HorizontalHeaderFormat
        ShortDayNames = ... # type: QCalendarWidget.HorizontalHeaderFormat
        LongDayNames = ... # type: QCalendarWidget.HorizontalHeaderFormat

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def setCalendar(self, calendar: QtCore.QCalendar) -> None: ...
    def calendar(self) -> QtCore.QCalendar: ...
    def setNavigationBarVisible(self, visible: bool) -> None: ...
    def setDateEditAcceptDelay(self, delay: int) -> None: ...
    def dateEditAcceptDelay(self) -> int: ...
    def setDateEditEnabled(self, enable: bool) -> None: ...
    def isDateEditEnabled(self) -> bool: ...
    def isNavigationBarVisible(self) -> bool: ...
    selectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentPageChanged: typing.ClassVar[QtCore.pyqtSignal]
    clicked: typing.ClassVar[QtCore.pyqtSignal]
    activated: typing.ClassVar[QtCore.pyqtSignal]
    def showToday(self) -> None: ...
    def showSelectedDate(self) -> None: ...
    def showPreviousYear(self) -> None: ...
    def showPreviousMonth(self) -> None: ...
    def showNextYear(self) -> None: ...
    def showNextMonth(self) -> None: ...
    def setSelectedDate(self, date: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def setDateRange(self, min: typing.Union[QtCore.QDate, datetime.date], max: typing.Union[QtCore.QDate, 
datetime.date]) -> None: ...
    def setCurrentPage(self, year: int, month: int) -> None: ...
    def paintCell(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRect, date: typing.Union[QtCore.QDate, 
datetime.date]) -> None: ...
    def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def resizeEvent(self, event: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def mousePressEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def eventFilter(self, watched: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def updateCells(self) -> None: ...
    def updateCell(self, date: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def setDateTextFormat(self, date: typing.Union[QtCore.QDate, datetime.date], color: QtGui.QTextCharFormat) -> None: ...
    @typing.overload
    def dateTextFormat(self) -> typing.Dict[QtCore.QDate, QtGui.QTextCharFormat]: ...
    @typing.overload
    def dateTextFormat(self, date: typing.Union[QtCore.QDate, datetime.date]) -> QtGui.QTextCharFormat: ...
    def setWeekdayTextFormat(self, dayOfWeek: QtCore.Qt.DayOfWeek, format: QtGui.QTextCharFormat) -> None: ...
    def weekdayTextFormat(self, dayOfWeek: QtCore.Qt.DayOfWeek) -> QtGui.QTextCharFormat: ...
    def setHeaderTextFormat(self, format: QtGui.QTextCharFormat) -> None: ...
    def headerTextFormat(self) -> QtGui.QTextCharFormat: ...
    def setVerticalHeaderFormat(self, format: 'QCalendarWidget.VerticalHeaderFormat') -> None: ...
    def verticalHeaderFormat(self) -> 'QCalendarWidget.VerticalHeaderFormat': ...
    def setHorizontalHeaderFormat(self, format: 'QCalendarWidget.HorizontalHeaderFormat') -> None: ...
    def horizontalHeaderFormat(self) -> 'QCalendarWidget.HorizontalHeaderFormat': ...
    def setSelectionMode(self, mode: 'QCalendarWidget.SelectionMode') -> None: ...
    def selectionMode(self) -> 'QCalendarWidget.SelectionMode': ...
    def setGridVisible(self, show: bool) -> None: ...
    def isGridVisible(self) -> bool: ...
    def setFirstDayOfWeek(self, dayOfWeek: QtCore.Qt.DayOfWeek) -> None: ...
    def firstDayOfWeek(self) -> QtCore.Qt.DayOfWeek: ...
    def setMaximumDate(self, date: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def maximumDate(self) -> QtCore.QDate: ...
    def setMinimumDate(self, date: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def minimumDate(self) -> QtCore.QDate: ...
    def monthShown(self) -> int: ...
    def yearShown(self) -> int: ...
    def selectedDate(self) -> QtCore.QDate: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QCheckBox(QAbstractButton):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    def initStyleOption(self, option: typing.Optional['QStyleOptionButton']) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def nextCheckState(self) -> None: ...
    def checkStateSet(self) -> None: ...
    def hitButton(self, pos: QtCore.QPoint) -> bool: ...
    stateChanged: typing.ClassVar[QtCore.pyqtSignal]
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def setCheckState(self, state: QtCore.Qt.CheckState) -> None: ...
    def checkState(self) -> QtCore.Qt.CheckState: ...
    def isTristate(self) -> bool: ...
    def setTristate(self, on: bool = ...) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QDialog(QWidget):

    class DialogCode(int):
        Rejected = ... # type: QDialog.DialogCode
        Accepted = ... # type: QDialog.DialogCode

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def contextMenuEvent(self, a0: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def closeEvent(self, a0: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    rejected: typing.ClassVar[QtCore.pyqtSignal]
    finished: typing.ClassVar[QtCore.pyqtSignal]
    accepted: typing.ClassVar[QtCore.pyqtSignal]
    def open(self) -> None: ...
    def reject(self) -> None: ...
    def accept(self) -> None: ...
    def done(self, a0: int) -> None: ...
    def exec(self) -> int: ...
    def exec_(self) -> int: ...
    def setResult(self, r: int) -> None: ...
    def setModal(self, modal: bool) -> None: ...
    def isSizeGripEnabled(self) -> bool: ...
    def setSizeGripEnabled(self, a0: bool) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setVisible(self, visible: bool) -> None: ...
    def result(self) -> int: ...


class QColorDialog(QDialog):

    class ColorDialogOption(int):
        ShowAlphaChannel = ... # type: QColorDialog.ColorDialogOption
        NoButtons = ... # type: QColorDialog.ColorDialogOption
        DontUseNativeDialog = ... # type: QColorDialog.ColorDialogOption

    class ColorDialogOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) ->
'QColorDialog.ColorDialogOptions': ...
        def __xor__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) ->
'QColorDialog.ColorDialogOptions': ...
        def __ior__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) ->
```

```
'QColorDialog.ColorDialogOptions': ...
        def __or__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) ->
'QColorDialog.ColorDialogOptions': ...
        def __iand__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) ->
'QColorDialog.ColorDialogOptions': ...
        def __and__(self, f: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) ->
'QColorDialog.ColorDialogOptions': ...
        def __invert__(self) -> 'QColorDialog.ColorDialogOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, initial: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], parent: typing.Optional[QWidget] = ...) ->
None: ...

    def setVisible(self, visible: bool) -> None: ...
    @typing.overload
    def open(self) -> None: ...
    @typing.overload
    def open(self, slot: PYQT_SLOT) -> None: ...
    def options(self) -> 'QColorDialog.ColorDialogOptions': ...
    def setOptions(self, options: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption']) -> None:
...
    def testOption(self, option: 'QColorDialog.ColorDialogOption') -> bool: ...
    def setOption(self, option: 'QColorDialog.ColorDialogOption', on: bool = ...) -> None: ...
    def selectedColor(self) -> QtGui.QColor: ...
    def currentColor(self) -> QtGui.QColor: ...
    def setCurrentColor(self, color: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    def done(self, result: int) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    currentColorChanged: typing.ClassVar[QtCore.pyqtSignal]
    colorSelected: typing.ClassVar[QtCore.pyqtSignal]
    @staticmethod
    def setStandardColor(index: int, color: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    @staticmethod
    def standardColor(index: int) -> QtGui.QColor: ...
    @staticmethod
    def setCustomColor(index: int, color: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    @staticmethod
    def customColor(index: int) -> QtGui.QColor: ...
    @staticmethod
    def customCount() -> int: ...
    @staticmethod
    def getColor(initial: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor] = ..., parent: typing.Optional[QWidget] = ..., title:
typing.Optional[str] = ..., options: typing.Union['QColorDialog.ColorDialogOptions', 'QColorDialog.ColorDialogOption'] = ...) ->
QtGui.QColor: ...


class QColumnView(QAbstractItemView):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def currentChanged(self, current: QtCore.QModelIndex, previous: QtCore.QModelIndex) -> None: ...
    def rowsInserted(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def verticalOffset(self) -> int: ...
    def horizontalOffset(self) -> int: ...
    def visualRegionForSelection(self, selection: QtCore.QItemSelection) -> QtGui.QRegion: ...
    def setSelection(self, rect: QtCore.QRect, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def resizeEvent(self, event: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def moveCursor(self, cursorAction: QAbstractItemView.CursorAction, modifiers: typing.Union[QtCore.Qt.KeyboardModifiers,
QtCore.Qt.KeyboardModifier]) -> QtCore.QModelIndex: ...
    def isIndexHidden(self, index: QtCore.QModelIndex) -> bool: ...
    def initializeColumn(self, column: typing.Optional[QAbstractItemView]) -> None: ...
    def createColumn(self, rootIndex: QtCore.QModelIndex) -> typing.Optional[QAbstractItemView]: ...
    updatePreviewWidget: typing.ClassVar[QtCore.pyqtSignal]
    def selectAll(self) -> None: ...
    def setRootIndex(self, index: QtCore.QModelIndex) -> None: ...
    def setSelectionModel(self, selectionModel: typing.Optional[QtCore.QItemSelectionModel]) -> None: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...
    def visualRect(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def scrollTo(self, index: QtCore.QModelIndex, hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def indexAt(self, point: QtCore.QPoint) -> QtCore.QModelIndex: ...
    def setResizeGripsVisible(self, visible: bool) -> None: ...
    def setPreviewWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def setColumnWidths(self, list: typing.Iterable[int]) -> None: ...
    def resizeGripsVisible(self) -> bool: ...
    def previewWidget(self) -> typing.Optional[QWidget]: ...
    def columnWidths(self) -> typing.List[int]: ...


class QComboBox(QWidget):

    class SizeAdjustPolicy(int):
        AdjustToContents = ... # type: QComboBox.SizeAdjustPolicy
        AdjustToContentsOnFirstShow = ... # type: QComboBox.SizeAdjustPolicy
        AdjustToMinimumContentsLength = ... # type: QComboBox.SizeAdjustPolicy
        AdjustToMinimumContentsLengthWithIcon = ... # type: QComboBox.SizeAdjustPolicy

    class InsertPolicy(int):
        NoInsert = ... # type: QComboBox.InsertPolicy
        InsertAtTop = ... # type: QComboBox.InsertPolicy
        InsertAtCurrent = ... # type: QComboBox.InsertPolicy
        InsertAtBottom = ... # type: QComboBox.InsertPolicy
        InsertAfterCurrent = ... # type: QComboBox.InsertPolicy
        InsertBeforeCurrent = ... # type: QComboBox.InsertPolicy
        InsertAlphabetically = ... # type: QComboBox.InsertPolicy

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def placeholderText(self) -> str: ...
    def setPlaceholderText(self, placeholderText: typing.Optional[str]) -> None: ...
    textHighlighted: typing.ClassVar[QtCore.pyqtSignal]
    textActivated: typing.ClassVar[QtCore.pyqtSignal]
    def currentData(self, role: int = ...) -> typing.Any: ...
    @typing.overload
    def inputMethodQuery(self, a0: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    @typing.overload
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery, argument: typing.Any) -> typing.Any: ...
    def inputMethodEvent(self, a0: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def contextMenuEvent(self, e: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def keyReleaseEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def hideEvent(self, e: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def showEvent(self, e: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def focusOutEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionComboBox']) -> None: ...
    highlighted: typing.ClassVar[QtCore.pyqtSignal]
    currentTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentIndexChanged: typing.ClassVar[QtCore.pyqtSignal]
    activated: typing.ClassVar[QtCore.pyqtSignal]
    editTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setCurrentText(self, text: typing.Optional[str]) -> None: ...
    def setEditText(self, text: typing.Optional[str]) -> None: ...
    def clearEditText(self) -> None: ...
    def clear(self) -> None: ...
    def insertSeparator(self, index: int) -> None: ...
    def completer(self) -> typing.Optional['QCompleter']: ...
    def setCompleter(self, c: typing.Optional['QCompleter']) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def hidePopup(self) -> None: ...
    def showPopup(self) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setView(self, itemView: typing.Optional[QAbstractItemView]) -> None: ...
    def view(self) -> typing.Optional[QAbstractItemView]: ...
    def setItemData(self, index: int, value: typing.Any, role: int = ...) -> None: ...
    def setItemIcon(self, index: int, icon: QtGui.QIcon) -> None: ...
    def setItemText(self, index: int, text: typing.Optional[str]) -> None: ...
    def removeItem(self, index: int) -> None: ...
    def insertItems(self, index: int, texts: typing.Iterable[typing.Optional[str]]) -> None: ...
    @typing.overload
    def insertItem(self, index: int, text: typing.Optional[str], userData: typing.Any = ...) -> None: ...
    @typing.overload
    def insertItem(self, index: int, icon: QtGui.QIcon, text: typing.Optional[str], userData: typing.Any = ...) -> None: ...
    @typing.overload
    def addItem(self, text: typing.Optional[str], userData: typing.Any = ...) -> None: ...
    @typing.overload
    def addItem(self, icon: QtGui.QIcon, text: typing.Optional[str], userData: typing.Any = ...) -> None: ...
    def addItems(self, texts: typing.Iterable[typing.Optional[str]]) -> None: ...
    def itemData(self, index: int, role: int = ...) -> typing.Any: ...
    def itemIcon(self, index: int) -> QtGui.QIcon: ...
    def itemText(self, index: int) -> str: ...
    def currentText(self) -> str: ...
    def setCurrentIndex(self, index: int) -> None: ...
    def currentIndex(self) -> int: ...
    def setModelColumn(self, visibleColumn: int) -> None: ...
    def modelColumn(self) -> int: ...
    def setRootModelIndex(self, index: QtCore.QModelIndex) -> None: ...
    def rootModelIndex(self) -> QtCore.QModelIndex: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...
    def model(self) -> typing.Optional[QtCore.QAbstractItemModel]: ...
    def setItemDelegate(self, delegate: typing.Optional[QAbstractItemDelegate]) -> None: ...
    def itemDelegate(self) -> typing.Optional[QAbstractItemDelegate]: ...
    def validator(self) -> typing.Optional[QtGui.QValidator]: ...
    def setValidator(self, v: typing.Optional[QtGui.QValidator]) -> None: ...
    def lineEdit(self) -> typing.Optional['QLineEdit']: ...
    def setLineEdit(self, edit: typing.Optional['QLineEdit']) -> None: ...
    def setEditable(self, editable: bool) -> None: ...
    def isEditable(self) -> bool: ...
    def setIconSize(self, size: QtCore.QSize) -> None: ...
    def iconSize(self) -> QtCore.QSize: ...
    def setMinimumContentsLength(self, characters: int) -> None: ...
    def minimumContentsLength(self) -> int: ...
    def setSizeAdjustPolicy(self, policy: 'QComboBox.SizeAdjustPolicy') -> None: ...
    def sizeAdjustPolicy(self) -> 'QComboBox.SizeAdjustPolicy': ...
    def setInsertPolicy(self, policy: 'QComboBox.InsertPolicy') -> None: ...
    def insertPolicy(self) -> 'QComboBox.InsertPolicy': ...
    def findData(self, data: typing.Any, role: int = ..., flags: typing.Union[QtCore.Qt.MatchFlags, QtCore.Qt.MatchFlag] = ...) -
> int: ...
    def findText(self, text: typing.Optional[str], flags: typing.Union[QtCore.Qt.MatchFlags, QtCore.Qt.MatchFlag] = ...) -> int:
...
    def hasFrame(self) -> bool: ...
    def setFrame(self, a0: bool) -> None: ...
    def setDuplicatesEnabled(self, enable: bool) -> None: ...
    def duplicatesEnabled(self) -> bool: ...
    def maxCount(self) -> int: ...
    def setMaxCount(self, max: int) -> None: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def setMaxVisibleItems(self, maxItems: int) -> None: ...
    def maxVisibleItems(self) -> int: ...


class QPushButton(QAbstractButton):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, icon: QtGui.QIcon, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    def hitButton(self, pos: QtCore.QPoint) -> bool: ...
    def focusOutEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionButton']) -> None: ...
    def showMenu(self) -> None: ...
    def isFlat(self) -> bool: ...
    def setFlat(self, a0: bool) -> None: ...
    def menu(self) -> typing.Optional['QMenu']: ...
    def setMenu(self, menu: typing.Optional['QMenu']) -> None: ...
    def setDefault(self, a0: bool) -> None: ...
    def isDefault(self) -> bool: ...
    def setAutoDefault(self, a0: bool) -> None: ...
    def autoDefault(self) -> bool: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QCommandLinkButton(QPushButton):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], description: typing.Optional[str], parent: typing.Optional[QWidget] = ...) ->
None: ...

    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def heightForWidth(self, a0: int) -> int: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setDescription(self, description: typing.Optional[str]) -> None: ...
    def description(self) -> str: ...


class QStyle(QtCore.QObject):

    class RequestSoftwareInputPanel(int):
        RSIP_OnMouseClickAndAlreadyFocused = ... # type: QStyle.RequestSoftwareInputPanel
        RSIP_OnMouseClick = ... # type: QStyle.RequestSoftwareInputPanel

    class StandardPixmap(int):
        SP_TitleBarMenuButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarMinButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarMaxButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarCloseButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarNormalButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarShadeButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarUnshadeButton = ... # type: QStyle.StandardPixmap
        SP_TitleBarContextHelpButton = ... # type: QStyle.StandardPixmap
        SP_DockWidgetCloseButton = ... # type: QStyle.StandardPixmap
        SP_MessageBoxInformation = ... # type: QStyle.StandardPixmap
        SP_MessageBoxWarning = ... # type: QStyle.StandardPixmap
        SP_MessageBoxCritical = ... # type: QStyle.StandardPixmap
        SP_MessageBoxQuestion = ... # type: QStyle.StandardPixmap
        SP_DesktopIcon = ... # type: QStyle.StandardPixmap
        SP_TrashIcon = ... # type: QStyle.StandardPixmap
        SP_ComputerIcon = ... # type: QStyle.StandardPixmap
        SP_DriveFDIcon = ... # type: QStyle.StandardPixmap
        SP_DriveHDIcon = ... # type: QStyle.StandardPixmap
        SP_DriveCDIcon = ... # type: QStyle.StandardPixmap
        SP_DriveDVDIcon = ... # type: QStyle.StandardPixmap
    SP_DriveNetIcon = ... # type: QStyle.StandardPixmap
    SP_DirOpenIcon = ... # type: QStyle.StandardPixmap
    SP_DirClosedIcon = ... # type: QStyle.StandardPixmap
    SP_DirLinkIcon = ... # type: QStyle.StandardPixmap
    SP_FileIcon = ... # type: QStyle.StandardPixmap
    SP_FileLinkIcon = ... # type: QStyle.StandardPixmap
    SP_ToolBarHorizontalExtensionButton = ... # type: QStyle.StandardPixmap
    SP_ToolBarVerticalExtensionButton = ... # type: QStyle.StandardPixmap
    SP_FileDialogStart = ... # type: QStyle.StandardPixmap
    SP_FileDialogEnd = ... # type: QStyle.StandardPixmap
    SP_FileDialogToParent = ... # type: QStyle.StandardPixmap
    SP_FileDialogNewFolder = ... # type: QStyle.StandardPixmap
    SP_FileDialogDetailedView = ... # type: QStyle.StandardPixmap
    SP_FileDialogInfoView = ... # type: QStyle.StandardPixmap
    SP_FileDialogContentsView = ... # type: QStyle.StandardPixmap
    SP_FileDialogListView = ... # type: QStyle.StandardPixmap
    SP_FileDialogBack = ... # type: QStyle.StandardPixmap
    SP_DirIcon = ... # type: QStyle.StandardPixmap
    SP_DialogOkButton = ... # type: QStyle.StandardPixmap
    SP_DialogCancelButton = ... # type: QStyle.StandardPixmap
    SP_DialogHelpButton = ... # type: QStyle.StandardPixmap
    SP_DialogOpenButton = ... # type: QStyle.StandardPixmap
    SP_DialogSaveButton = ... # type: QStyle.StandardPixmap
    SP_DialogCloseButton = ... # type: QStyle.StandardPixmap
    SP_DialogApplyButton = ... # type: QStyle.StandardPixmap
    SP_DialogResetButton = ... # type: QStyle.StandardPixmap
    SP_DialogDiscardButton = ... # type: QStyle.StandardPixmap
    SP_DialogYesButton = ... # type: QStyle.StandardPixmap
    SP_DialogNoButton = ... # type: QStyle.StandardPixmap
    SP_ArrowUp = ... # type: QStyle.StandardPixmap
    SP_ArrowDown = ... # type: QStyle.StandardPixmap
    SP_ArrowLeft = ... # type: QStyle.StandardPixmap
    SP_ArrowRight = ... # type: QStyle.StandardPixmap
    SP_ArrowBack = ... # type: QStyle.StandardPixmap
    SP_ArrowForward = ... # type: QStyle.StandardPixmap
    SP_DirHomeIcon = ... # type: QStyle.StandardPixmap
    SP_CommandLink = ... # type: QStyle.StandardPixmap
    SP_VistaShield = ... # type: QStyle.StandardPixmap
    SP_BrowserReload = ... # type: QStyle.StandardPixmap
    SP_BrowserStop = ... # type: QStyle.StandardPixmap
    SP_MediaPlay = ... # type: QStyle.StandardPixmap
    SP_MediaStop = ... # type: QStyle.StandardPixmap
    SP_MediaPause = ... # type: QStyle.StandardPixmap
    SP_MediaSkipForward = ... # type: QStyle.StandardPixmap
    SP_MediaSkipBackward = ... # type: QStyle.StandardPixmap
    SP_MediaSeekForward = ... # type: QStyle.StandardPixmap
    SP_MediaSeekBackward = ... # type: QStyle.StandardPixmap
    SP_MediaVolume = ... # type: QStyle.StandardPixmap
    SP_MediaVolumeMuted = ... # type: QStyle.StandardPixmap
    SP_DirLinkOpenIcon = ... # type: QStyle.StandardPixmap
    SP_LineEditClearButton = ... # type: QStyle.StandardPixmap
    SP_DialogYesToAllButton = ... # type: QStyle.StandardPixmap
    SP_DialogNoToAllButton = ... # type: QStyle.StandardPixmap
    SP_DialogSaveAllButton = ... # type: QStyle.StandardPixmap
    SP_DialogAbortButton = ... # type: QStyle.StandardPixmap
    SP_DialogRetryButton = ... # type: QStyle.StandardPixmap
    SP_DialogIgnoreButton = ... # type: QStyle.StandardPixmap
    SP_RestoreDefaultsButton = ... # type: QStyle.StandardPixmap
    SP_CustomBase = ... # type: QStyle.StandardPixmap

class StyleHint(int):
    SH_EtchDisabledText = ... # type: QStyle.StyleHint
    SH_DitherDisabledText = ... # type: QStyle.StyleHint
    SH_ScrollBar_MiddleClickAbsolutePosition = ... # type: QStyle.StyleHint
    SH_ScrollBar_ScrollWhenPointerLeavesControl = ... # type: QStyle.StyleHint
    SH_TabBar_SelectMouseType = ... # type: QStyle.StyleHint
    SH_TabBar_Alignment = ... # type: QStyle.StyleHint
    SH_Header_ArrowAlignment = ... # type: QStyle.StyleHint
    SH_Slider_SnapToValue = ... # type: QStyle.StyleHint
    SH_Slider_SloppyKeyEvents = ... # type: QStyle.StyleHint
```

```
SH_ProgressDialog_CenterCancelButton = ... # type: QStyle.StyleHint
SH_ProgressDialog_TextLabelAlignment = ... # type: QStyle.StyleHint
SH_PrintDialog_RightAlignButtons = ... # type: QStyle.StyleHint
SH_MainWindow_SpaceBelowMenuBar = ... # type: QStyle.StyleHint
SH_FontDialog_SelectAssociatedText = ... # type: QStyle.StyleHint
SH_Menu_AllowActiveAndDisabled = ... # type: QStyle.StyleHint
SH_Menu_SpaceActivatesItem = ... # type: QStyle.StyleHint
SH_Menu_SubMenuPopupDelay = ... # type: QStyle.StyleHint
SH_ScrollView_FrameOnlyAroundContents = ... # type: QStyle.StyleHint
SH_MenuBar_AltKeyNavigation = ... # type: QStyle.StyleHint
SH_ComboBox_ListMouseTracking = ... # type: QStyle.StyleHint
SH_Menu_MouseTracking = ... # type: QStyle.StyleHint
SH_MenuBar_MouseTracking = ... # type: QStyle.StyleHint
SH_ItemView_ChangeHighlightOnFocus = ... # type: QStyle.StyleHint
SH_Widget_ShareActivation = ... # type: QStyle.StyleHint
SH_Workspace_FillSpaceOnMaximize = ... # type: QStyle.StyleHint
SH_ComboBox_Popup = ... # type: QStyle.StyleHint
SH_TitleBar_NoBorder = ... # type: QStyle.StyleHint
SH_ScrollBar_StopMouseOverSlider = ... # type: QStyle.StyleHint
SH_BlinkCursorWhenTextSelected = ... # type: QStyle.StyleHint
SH_RichText_FullWidthSelection = ... # type: QStyle.StyleHint
SH_Menu_Scrollable = ... # type: QStyle.StyleHint
SH_GroupBox_TextLabelVerticalAlignment = ... # type: QStyle.StyleHint
SH_GroupBox_TextLabelColor = ... # type: QStyle.StyleHint
SH_Menu_SloppySubMenus = ... # type: QStyle.StyleHint
SH_Table_GridLineColor = ... # type: QStyle.StyleHint
SH_LineEdit_PasswordCharacter = ... # type: QStyle.StyleHint
SH_DialogButtons_DefaultButton = ... # type: QStyle.StyleHint
SH_ToolBox_SelectedPageTitleBold = ... # type: QStyle.StyleHint
SH_TabBar_PreferNoArrows = ... # type: QStyle.StyleHint
SH_ScrollBar_LeftClickAbsolutePosition = ... # type: QStyle.StyleHint
SH_UnderlineShortcut = ... # type: QStyle.StyleHint
SH_SpinBox_AnimateButton = ... # type: QStyle.StyleHint
SH_SpinBox_KeyPressAutoRepeatRate = ... # type: QStyle.StyleHint
SH_SpinBox_ClickAutoRepeatRate = ... # type: QStyle.StyleHint
SH_Menu_FillScreenWithScroll = ... # type: QStyle.StyleHint
SH_ToolTipLabel_Opacity = ... # type: QStyle.StyleHint
SH_DrawMenuBarSeparator = ... # type: QStyle.StyleHint
SH_TitleBar_ModifyNotification = ... # type: QStyle.StyleHint
SH_Button_FocusPolicy = ... # type: QStyle.StyleHint
SH_MessageBox_UseBorderForButtonSpacing = ... # type: QStyle.StyleHint
SH_TitleBar_AutoRaise = ... # type: QStyle.StyleHint
SH_ToolButton_PopupDelay = ... # type: QStyle.StyleHint
SH_FocusFrame_Mask = ... # type: QStyle.StyleHint
SH_RubberBand_Mask = ... # type: QStyle.StyleHint
SH_WindowFrame_Mask = ... # type: QStyle.StyleHint
SH_SpinControls_DisableOnBounds = ... # type: QStyle.StyleHint
SH_Dial_BackgroundRole = ... # type: QStyle.StyleHint
SH_ComboBox_LayoutDirection = ... # type: QStyle.StyleHint
SH_ItemView_EllipsisLocation = ... # type: QStyle.StyleHint
SH_ItemView_ShowDecorationSelected = ... # type: QStyle.StyleHint
SH_ItemView_ActivateItemOnSingleClick = ... # type: QStyle.StyleHint
SH_ScrollBar_ContextMenu = ... # type: QStyle.StyleHint
SH_ScrollBar_RollBetweenButtons = ... # type: QStyle.StyleHint
SH_Slider_StopMouseOverSlider = ... # type: QStyle.StyleHint
SH_Slider_AbsoluteSetButtons = ... # type: QStyle.StyleHint
SH_Slider_PageSetButtons = ... # type: QStyle.StyleHint
SH_Menu_KeyboardSearch = ... # type: QStyle.StyleHint
SH_TabBar_ElideMode = ... # type: QStyle.StyleHint
SH_DialogButtonLayout = ... # type: QStyle.StyleHint
SH_ComboBox_PopupFrameStyle = ... # type: QStyle.StyleHint
SH_MessageBox_TextInteractionFlags = ... # type: QStyle.StyleHint
SH_DialogButtonBox_ButtonsHaveIcons = ... # type: QStyle.StyleHint
SH_SpellCheckUnderlineStyle = ... # type: QStyle.StyleHint
SH_MessageBox_CenterButtons = ... # type: QStyle.StyleHint
SH_Menu_SelectionWrap = ... # type: QStyle.StyleHint
SH_ItemView_MovementWithoutUpdatingSelection = ... # type: QStyle.StyleHint
SH_ToolTip_Mask = ... # type: QStyle.StyleHint
SH_FocusFrame_AboveWidget = ... # type: QStyle.StyleHint
SH_TextControl_FocusIndicatorTextCharFormat = ... # type: QStyle.StyleHint
```

```
    SH_WizardStyle = ... # type: QStyle.StyleHint
    SH_ItemView_ArrowKeysNavigateIntoChildren = ... # type: QStyle.StyleHint
    SH_Menu_Mask = ... # type: QStyle.StyleHint
    SH_Menu_FlashTriggeredItem = ... # type: QStyle.StyleHint
    SH_Menu_FadeOutOnHide = ... # type: QStyle.StyleHint
    SH_SpinBox_ClickAutoRepeatThreshold = ... # type: QStyle.StyleHint
    SH_ItemView_PaintAlternatingRowColorsForEmptyArea = ... # type: QStyle.StyleHint
    SH_FormLayoutWrapPolicy = ... # type: QStyle.StyleHint
    SH_TabWidget_DefaultTabPosition = ... # type: QStyle.StyleHint
    SH_ToolBar_Movable = ... # type: QStyle.StyleHint
    SH_FormLayoutFieldGrowthPolicy = ... # type: QStyle.StyleHint
    SH_FormLayoutFormAlignment = ... # type: QStyle.StyleHint
    SH_FormLayoutLabelAlignment = ... # type: QStyle.StyleHint
    SH_ItemView_DrawDelegateFrame = ... # type: QStyle.StyleHint
    SH_TabBar_CloseButtonPosition = ... # type: QStyle.StyleHint
    SH_DockWidget_ButtonsHaveFrame = ... # type: QStyle.StyleHint
    SH_ToolButtonStyle = ... # type: QStyle.StyleHint
    SH_RequestSoftwareInputPanel = ... # type: QStyle.StyleHint
    SH_ListViewExpand_SelectMouseType = ... # type: QStyle.StyleHint
    SH_ScrollBar_Transient = ... # type: QStyle.StyleHint
    SH_Menu_SupportsSections = ... # type: QStyle.StyleHint
    SH_ToolTip_WakeUpDelay = ... # type: QStyle.StyleHint
    SH_ToolTip_FallAsleepDelay = ... # type: QStyle.StyleHint
    SH_Widget_Animate = ... # type: QStyle.StyleHint
    SH_Splitter_OpaqueResize = ... # type: QStyle.StyleHint
    SH_LineEdit_PasswordMaskDelay = ... # type: QStyle.StyleHint
    SH_TabBar_ChangeCurrentDelay = ... # type: QStyle.StyleHint
    SH_Menu_SubMenuUniDirection = ... # type: QStyle.StyleHint
    SH_Menu_SubMenuUniDirectionFailCount = ... # type: QStyle.StyleHint
    SH_Menu_SubMenuSloppySelectOtherActions = ... # type: QStyle.StyleHint
    SH_Menu_SubMenuSloppyCloseTimeout = ... # type: QStyle.StyleHint
    SH_Menu_SubMenuResetWhenReenteringParent = ... # type: QStyle.StyleHint
    SH_Menu_SubMenuDontStartSloppyOnLeave = ... # type: QStyle.StyleHint
    SH_ItemView_ScrollMode = ... # type: QStyle.StyleHint
    SH_TitleBar_ShowToolTipsOnButtons = ... # type: QStyle.StyleHint
    SH_Widget_Animation_Duration = ... # type: QStyle.StyleHint
    SH_ComboBox_AllowWheelScrolling = ... # type: QStyle.StyleHint
    SH_SpinBox_ButtonsInsideFrame = ... # type: QStyle.StyleHint
    SH_SpinBox_StepModifier = ... # type: QStyle.StyleHint
    SH_CustomBase = ... # type: QStyle.StyleHint

class ContentsType(int):
    CT_PushButton = ... # type: QStyle.ContentsType
    CT_CheckBox = ... # type: QStyle.ContentsType
    CT_RadioButton = ... # type: QStyle.ContentsType
    CT_ToolButton = ... # type: QStyle.ContentsType
    CT_ComboBox = ... # type: QStyle.ContentsType
    CT_Splitter = ... # type: QStyle.ContentsType
    CT_ProgressBar = ... # type: QStyle.ContentsType
    CT_MenuItem = ... # type: QStyle.ContentsType
    CT_MenuBarItem = ... # type: QStyle.ContentsType
    CT_MenuBar = ... # type: QStyle.ContentsType
    CT_Menu = ... # type: QStyle.ContentsType
    CT_TabBarTab = ... # type: QStyle.ContentsType
    CT_Slider = ... # type: QStyle.ContentsType
    CT_ScrollBar = ... # type: QStyle.ContentsType
    CT_LineEdit = ... # type: QStyle.ContentsType
    CT_SpinBox = ... # type: QStyle.ContentsType
    CT_SizeGrip = ... # type: QStyle.ContentsType
    CT_TabWidget = ... # type: QStyle.ContentsType
    CT_DialogButtons = ... # type: QStyle.ContentsType
    CT_HeaderSection = ... # type: QStyle.ContentsType
    CT_GroupBox = ... # type: QStyle.ContentsType
    CT_MdiControls = ... # type: QStyle.ContentsType
    CT_ItemViewItem = ... # type: QStyle.ContentsType
    CT_CustomBase = ... # type: QStyle.ContentsType

class PixelMetric(int):
    PM_ButtonMargin = ... # type: QStyle.PixelMetric
    PM_ButtonDefaultIndicator = ... # type: QStyle.PixelMetric
```

```
PM_MenuButtonIndicator = ... # type: QStyle.PixelMetric
PM_ButtonShiftHorizontal = ... # type: QStyle.PixelMetric
PM_ButtonShiftVertical = ... # type: QStyle.PixelMetric
PM_DefaultFrameWidth = ... # type: QStyle.PixelMetric
PM_SpinBoxFrameWidth = ... # type: QStyle.PixelMetric
PM_ComboBoxFrameWidth = ... # type: QStyle.PixelMetric
PM_MaximumDragDistance = ... # type: QStyle.PixelMetric
PM_ScrollBarExtent = ... # type: QStyle.PixelMetric
PM_ScrollBarSliderMin = ... # type: QStyle.PixelMetric
PM_SliderThickness = ... # type: QStyle.PixelMetric
PM_SliderControlThickness = ... # type: QStyle.PixelMetric
PM_SliderLength = ... # type: QStyle.PixelMetric
PM_SliderTickmarkOffset = ... # type: QStyle.PixelMetric
PM_SliderSpaceAvailable = ... # type: QStyle.PixelMetric
PM_DockWidgetSeparatorExtent = ... # type: QStyle.PixelMetric
PM_DockWidgetHandleExtent = ... # type: QStyle.PixelMetric
PM_DockWidgetFrameWidth = ... # type: QStyle.PixelMetric
PM_TabBarTabOverlap = ... # type: QStyle.PixelMetric
PM_TabBarTabHSpace = ... # type: QStyle.PixelMetric
PM_TabBarTabVSpace = ... # type: QStyle.PixelMetric
PM_TabBarBaseHeight = ... # type: QStyle.PixelMetric
PM_TabBarBaseOverlap = ... # type: QStyle.PixelMetric
PM_ProgressBarChunkWidth = ... # type: QStyle.PixelMetric
PM_SplitterWidth = ... # type: QStyle.PixelMetric
PM_TitleBarHeight = ... # type: QStyle.PixelMetric
PM_MenuScrollerHeight = ... # type: QStyle.PixelMetric
PM_MenuHMargin = ... # type: QStyle.PixelMetric
PM_MenuVMargin = ... # type: QStyle.PixelMetric
PM_MenuPanelWidth = ... # type: QStyle.PixelMetric
PM_MenuTearoffHeight = ... # type: QStyle.PixelMetric
PM_MenuDesktopFrameWidth = ... # type: QStyle.PixelMetric
PM_MenuBarPanelWidth = ... # type: QStyle.PixelMetric
PM_MenuBarItemSpacing = ... # type: QStyle.PixelMetric
PM_MenuBarVMargin = ... # type: QStyle.PixelMetric
PM_MenuBarHMargin = ... # type: QStyle.PixelMetric
PM_IndicatorWidth = ... # type: QStyle.PixelMetric
PM_IndicatorHeight = ... # type: QStyle.PixelMetric
PM_ExclusiveIndicatorWidth = ... # type: QStyle.PixelMetric
PM_ExclusiveIndicatorHeight = ... # type: QStyle.PixelMetric
PM_DialogButtonsSeparator = ... # type: QStyle.PixelMetric
PM_DialogButtonsButtonWidth = ... # type: QStyle.PixelMetric
PM_DialogButtonsButtonHeight = ... # type: QStyle.PixelMetric
PM_MdiSubWindowFrameWidth = ... # type: QStyle.PixelMetric
PM_MDIFrameWidth = ... # type: QStyle.PixelMetric
PM_MdiSubWindowMinimizedWidth = ... # type: QStyle.PixelMetric
PM_MDIMinimizedWidth = ... # type: QStyle.PixelMetric
PM_HeaderMargin = ... # type: QStyle.PixelMetric
PM_HeaderMarkSize = ... # type: QStyle.PixelMetric
PM_HeaderGripMargin = ... # type: QStyle.PixelMetric
PM_TabBarTabShiftHorizontal = ... # type: QStyle.PixelMetric
PM_TabBarTabShiftVertical = ... # type: QStyle.PixelMetric
PM_TabBarScrollButtonWidth = ... # type: QStyle.PixelMetric
PM_ToolBarFrameWidth = ... # type: QStyle.PixelMetric
PM_ToolBarHandleExtent = ... # type: QStyle.PixelMetric
PM_ToolBarItemSpacing = ... # type: QStyle.PixelMetric
PM_ToolBarItemMargin = ... # type: QStyle.PixelMetric
PM_ToolBarSeparatorExtent = ... # type: QStyle.PixelMetric
PM_ToolBarExtensionExtent = ... # type: QStyle.PixelMetric
PM_SpinBoxSliderHeight = ... # type: QStyle.PixelMetric
PM_DefaultTopLevelMargin = ... # type: QStyle.PixelMetric
PM_DefaultChildMargin = ... # type: QStyle.PixelMetric
PM_DefaultLayoutSpacing = ... # type: QStyle.PixelMetric
PM_ToolBarIconSize = ... # type: QStyle.PixelMetric
PM_ListViewIconSize = ... # type: QStyle.PixelMetric
PM_IconViewIconSize = ... # type: QStyle.PixelMetric
PM_SmallIconSize = ... # type: QStyle.PixelMetric
PM_LargeIconSize = ... # type: QStyle.PixelMetric
PM_FocusFrameVMargin = ... # type: QStyle.PixelMetric
PM_FocusFrameHMargin = ... # type: QStyle.PixelMetric
PM_ToolTipLabelFrameWidth = ... # type: QStyle.PixelMetric
```

```
        PM_CheckBoxLabelSpacing = ... # type: QStyle.PixelMetric
        PM_TabBarIconSize = ... # type: QStyle.PixelMetric
        PM_SizeGripSize = ... # type: QStyle.PixelMetric
        PM_DockWidgetTitleMargin = ... # type: QStyle.PixelMetric
        PM_MessageBoxIconSize = ... # type: QStyle.PixelMetric
        PM_ButtonIconSize = ... # type: QStyle.PixelMetric
        PM_DockWidgetTitleBarButtonMargin = ... # type: QStyle.PixelMetric
        PM_RadioButtonLabelSpacing = ... # type: QStyle.PixelMetric
        PM_LayoutLeftMargin = ... # type: QStyle.PixelMetric
        PM_LayoutTopMargin = ... # type: QStyle.PixelMetric
        PM_LayoutRightMargin = ... # type: QStyle.PixelMetric
        PM_LayoutBottomMargin = ... # type: QStyle.PixelMetric
        PM_LayoutHorizontalSpacing = ... # type: QStyle.PixelMetric
        PM_LayoutVerticalSpacing = ... # type: QStyle.PixelMetric
        PM_TabBar_ScrollButtonOverlap = ... # type: QStyle.PixelMetric
        PM_TextCursorWidth = ... # type: QStyle.PixelMetric
        PM_TabCloseIndicatorWidth = ... # type: QStyle.PixelMetric
        PM_TabCloseIndicatorHeight = ... # type: QStyle.PixelMetric
        PM_ScrollView_ScrollBarSpacing = ... # type: QStyle.PixelMetric
        PM_SubMenuOverlap = ... # type: QStyle.PixelMetric
        PM_ScrollView_ScrollBarOverlap = ... # type: QStyle.PixelMetric
        PM_TreeViewIndentation = ... # type: QStyle.PixelMetric
        PM_HeaderDefaultSectionSizeHorizontal = ... # type: QStyle.PixelMetric
        PM_HeaderDefaultSectionSizeVertical = ... # type: QStyle.PixelMetric
        PM_TitleBarButtonIconSize = ... # type: QStyle.PixelMetric
        PM_TitleBarButtonSize = ... # type: QStyle.PixelMetric
        PM_CustomBase = ... # type: QStyle.PixelMetric

    class SubControl(int):
        SC_None = ... # type: QStyle.SubControl
        SC_ScrollBarAddLine = ... # type: QStyle.SubControl
        SC_ScrollBarSubLine = ... # type: QStyle.SubControl
        SC_ScrollBarAddPage = ... # type: QStyle.SubControl
        SC_ScrollBarSubPage = ... # type: QStyle.SubControl
        SC_ScrollBarFirst = ... # type: QStyle.SubControl
        SC_ScrollBarLast = ... # type: QStyle.SubControl
        SC_ScrollBarSlider = ... # type: QStyle.SubControl
        SC_ScrollBarGroove = ... # type: QStyle.SubControl
        SC_SpinBoxUp = ... # type: QStyle.SubControl
        SC_SpinBoxDown = ... # type: QStyle.SubControl
        SC_SpinBoxFrame = ... # type: QStyle.SubControl
        SC_SpinBoxEditField = ... # type: QStyle.SubControl
        SC_ComboBoxFrame = ... # type: QStyle.SubControl
        SC_ComboBoxEditField = ... # type: QStyle.SubControl
        SC_ComboBoxArrow = ... # type: QStyle.SubControl
        SC_ComboBoxListBoxPopup = ... # type: QStyle.SubControl
        SC_SliderGroove = ... # type: QStyle.SubControl
        SC_SliderHandle = ... # type: QStyle.SubControl
        SC_SliderTickmarks = ... # type: QStyle.SubControl
        SC_ToolButton = ... # type: QStyle.SubControl
        SC_ToolButtonMenu = ... # type: QStyle.SubControl
        SC_TitleBarSysMenu = ... # type: QStyle.SubControl
        SC_TitleBarMinButton = ... # type: QStyle.SubControl
        SC_TitleBarMaxButton = ... # type: QStyle.SubControl
        SC_TitleBarCloseButton = ... # type: QStyle.SubControl
        SC_TitleBarNormalButton = ... # type: QStyle.SubControl
        SC_TitleBarShadeButton = ... # type: QStyle.SubControl
        SC_TitleBarUnshadeButton = ... # type: QStyle.SubControl
        SC_TitleBarContextHelpButton = ... # type: QStyle.SubControl
        SC_TitleBarLabel = ... # type: QStyle.SubControl
        SC_DialGroove = ... # type: QStyle.SubControl
        SC_DialHandle = ... # type: QStyle.SubControl
        SC_DialTickmarks = ... # type: QStyle.SubControl
        SC_GroupBoxCheckBox = ... # type: QStyle.SubControl
        SC_GroupBoxLabel = ... # type: QStyle.SubControl
        SC_GroupBoxContents = ... # type: QStyle.SubControl
        SC_GroupBoxFrame = ... # type: QStyle.SubControl
        SC_MdiMinButton = ... # type: QStyle.SubControl
        SC_MdiNormalButton = ... # type: QStyle.SubControl
        SC_MdiCloseButton = ... # type: QStyle.SubControl
    SC_CustomBase = ... # type: QStyle.SubControl
    SC_All = ... # type: QStyle.SubControl

class ComplexControl(int):
    CC_SpinBox = ... # type: QStyle.ComplexControl
    CC_ComboBox = ... # type: QStyle.ComplexControl
    CC_ScrollBar = ... # type: QStyle.ComplexControl
    CC_Slider = ... # type: QStyle.ComplexControl
    CC_ToolButton = ... # type: QStyle.ComplexControl
    CC_TitleBar = ... # type: QStyle.ComplexControl
    CC_Dial = ... # type: QStyle.ComplexControl
    CC_GroupBox = ... # type: QStyle.ComplexControl
    CC_MdiControls = ... # type: QStyle.ComplexControl
    CC_CustomBase = ... # type: QStyle.ComplexControl

class SubElement(int):
    SE_PushButtonContents = ... # type: QStyle.SubElement
    SE_PushButtonFocusRect = ... # type: QStyle.SubElement
    SE_CheckBoxIndicator = ... # type: QStyle.SubElement
    SE_CheckBoxContents = ... # type: QStyle.SubElement
    SE_CheckBoxFocusRect = ... # type: QStyle.SubElement
    SE_CheckBoxClickRect = ... # type: QStyle.SubElement
    SE_RadioButtonIndicator = ... # type: QStyle.SubElement
    SE_RadioButtonContents = ... # type: QStyle.SubElement
    SE_RadioButtonFocusRect = ... # type: QStyle.SubElement
    SE_RadioButtonClickRect = ... # type: QStyle.SubElement
    SE_ComboBoxFocusRect = ... # type: QStyle.SubElement
    SE_SliderFocusRect = ... # type: QStyle.SubElement
    SE_ProgressBarGroove = ... # type: QStyle.SubElement
    SE_ProgressBarContents = ... # type: QStyle.SubElement
    SE_ProgressBarLabel = ... # type: QStyle.SubElement
    SE_ToolBoxTabContents = ... # type: QStyle.SubElement
    SE_HeaderLabel = ... # type: QStyle.SubElement
    SE_HeaderArrow = ... # type: QStyle.SubElement
    SE_TabWidgetTabBar = ... # type: QStyle.SubElement
    SE_TabWidgetTabPane = ... # type: QStyle.SubElement
    SE_TabWidgetTabContents = ... # type: QStyle.SubElement
    SE_TabWidgetLeftCorner = ... # type: QStyle.SubElement
    SE_TabWidgetRightCorner = ... # type: QStyle.SubElement
    SE_ViewItemCheckIndicator = ... # type: QStyle.SubElement
    SE_TabBarTearIndicator = ... # type: QStyle.SubElement
    SE_TreeViewDisclosureItem = ... # type: QStyle.SubElement
    SE_LineEditContents = ... # type: QStyle.SubElement
    SE_FrameContents = ... # type: QStyle.SubElement
    SE_DockWidgetCloseButton = ... # type: QStyle.SubElement
    SE_DockWidgetFloatButton = ... # type: QStyle.SubElement
    SE_DockWidgetTitleBarText = ... # type: QStyle.SubElement
    SE_DockWidgetIcon = ... # type: QStyle.SubElement
    SE_CheckBoxLayoutItem = ... # type: QStyle.SubElement
    SE_ComboBoxLayoutItem = ... # type: QStyle.SubElement
    SE_DateTimeEditLayoutItem = ... # type: QStyle.SubElement
    SE_DialogButtonBoxLayoutItem = ... # type: QStyle.SubElement
    SE_LabelLayoutItem = ... # type: QStyle.SubElement
    SE_ProgressBarLayoutItem = ... # type: QStyle.SubElement
    SE_PushButtonLayoutItem = ... # type: QStyle.SubElement
    SE_RadioButtonLayoutItem = ... # type: QStyle.SubElement
    SE_SliderLayoutItem = ... # type: QStyle.SubElement
    SE_SpinBoxLayoutItem = ... # type: QStyle.SubElement
    SE_ToolButtonLayoutItem = ... # type: QStyle.SubElement
    SE_FrameLayoutItem = ... # type: QStyle.SubElement
    SE_GroupBoxLayoutItem = ... # type: QStyle.SubElement
    SE_TabWidgetLayoutItem = ... # type: QStyle.SubElement
    SE_ItemViewItemCheckIndicator = ... # type: QStyle.SubElement
    SE_ItemViewItemDecoration = ... # type: QStyle.SubElement
    SE_ItemViewItemText = ... # type: QStyle.SubElement
    SE_ItemViewItemFocusRect = ... # type: QStyle.SubElement
    SE_TabBarTabLeftButton = ... # type: QStyle.SubElement
    SE_TabBarTabRightButton = ... # type: QStyle.SubElement
    SE_TabBarTabText = ... # type: QStyle.SubElement
    SE_ShapedFrameContents = ... # type: QStyle.SubElement
    SE_ToolBarHandle = ... # type: QStyle.SubElement
    SE_TabBarTearIndicatorLeft = ... # type: QStyle.SubElement
    SE_TabBarScrollLeftButton = ... # type: QStyle.SubElement
    SE_TabBarScrollRightButton = ... # type: QStyle.SubElement
    SE_TabBarTearIndicatorRight = ... # type: QStyle.SubElement
    SE_PushButtonBevel = ... # type: QStyle.SubElement
    SE_CustomBase = ... # type: QStyle.SubElement

class ControlElement(int):
    CE_PushButton = ... # type: QStyle.ControlElement
    CE_PushButtonBevel = ... # type: QStyle.ControlElement
    CE_PushButtonLabel = ... # type: QStyle.ControlElement
    CE_CheckBox = ... # type: QStyle.ControlElement
    CE_CheckBoxLabel = ... # type: QStyle.ControlElement
    CE_RadioButton = ... # type: QStyle.ControlElement
    CE_RadioButtonLabel = ... # type: QStyle.ControlElement
    CE_TabBarTab = ... # type: QStyle.ControlElement
    CE_TabBarTabShape = ... # type: QStyle.ControlElement
    CE_TabBarTabLabel = ... # type: QStyle.ControlElement
    CE_ProgressBar = ... # type: QStyle.ControlElement
    CE_ProgressBarGroove = ... # type: QStyle.ControlElement
    CE_ProgressBarContents = ... # type: QStyle.ControlElement
    CE_ProgressBarLabel = ... # type: QStyle.ControlElement
    CE_MenuItem = ... # type: QStyle.ControlElement
    CE_MenuScroller = ... # type: QStyle.ControlElement
    CE_MenuVMargin = ... # type: QStyle.ControlElement
    CE_MenuHMargin = ... # type: QStyle.ControlElement
    CE_MenuTearoff = ... # type: QStyle.ControlElement
    CE_MenuEmptyArea = ... # type: QStyle.ControlElement
    CE_MenuBarItem = ... # type: QStyle.ControlElement
    CE_MenuBarEmptyArea = ... # type: QStyle.ControlElement
    CE_ToolButtonLabel = ... # type: QStyle.ControlElement
    CE_Header = ... # type: QStyle.ControlElement
    CE_HeaderSection = ... # type: QStyle.ControlElement
    CE_HeaderLabel = ... # type: QStyle.ControlElement
    CE_ToolBoxTab = ... # type: QStyle.ControlElement
    CE_SizeGrip = ... # type: QStyle.ControlElement
    CE_Splitter = ... # type: QStyle.ControlElement
    CE_RubberBand = ... # type: QStyle.ControlElement
    CE_DockWidgetTitle = ... # type: QStyle.ControlElement
    CE_ScrollBarAddLine = ... # type: QStyle.ControlElement
    CE_ScrollBarSubLine = ... # type: QStyle.ControlElement
    CE_ScrollBarAddPage = ... # type: QStyle.ControlElement
    CE_ScrollBarSubPage = ... # type: QStyle.ControlElement
    CE_ScrollBarSlider = ... # type: QStyle.ControlElement
    CE_ScrollBarFirst = ... # type: QStyle.ControlElement
    CE_ScrollBarLast = ... # type: QStyle.ControlElement
    CE_FocusFrame = ... # type: QStyle.ControlElement
    CE_ComboBoxLabel = ... # type: QStyle.ControlElement
    CE_ToolBar = ... # type: QStyle.ControlElement
    CE_ToolBoxTabShape = ... # type: QStyle.ControlElement
    CE_ToolBoxTabLabel = ... # type: QStyle.ControlElement
    CE_HeaderEmptyArea = ... # type: QStyle.ControlElement
    CE_ColumnViewGrip = ... # type: QStyle.ControlElement
    CE_ItemViewItem = ... # type: QStyle.ControlElement
    CE_ShapedFrame = ... # type: QStyle.ControlElement
    CE_CustomBase = ... # type: QStyle.ControlElement

class PrimitiveElement(int):
    PE_Frame = ... # type: QStyle.PrimitiveElement
    PE_FrameDefaultButton = ... # type: QStyle.PrimitiveElement
    PE_FrameDockWidget = ... # type: QStyle.PrimitiveElement
    PE_FrameFocusRect = ... # type: QStyle.PrimitiveElement
    PE_FrameGroupBox = ... # type: QStyle.PrimitiveElement
    PE_FrameLineEdit = ... # type: QStyle.PrimitiveElement
    PE_FrameMenu = ... # type: QStyle.PrimitiveElement
    PE_FrameStatusBar = ... # type: QStyle.PrimitiveElement
    PE_FrameTabWidget = ... # type: QStyle.PrimitiveElement
    PE_FrameWindow = ... # type: QStyle.PrimitiveElement
    PE_FrameButtonBevel = ... # type: QStyle.PrimitiveElement
    PE_FrameButtonTool = ... # type: QStyle.PrimitiveElement
    PE_FrameTabBarBase = ... # type: QStyle.PrimitiveElement
    PE_PanelButtonCommand = ... # type: QStyle.PrimitiveElement
    PE_PanelButtonBevel = ... # type: QStyle.PrimitiveElement
    PE_PanelButtonTool = ... # type: QStyle.PrimitiveElement
    PE_PanelMenuBar = ... # type: QStyle.PrimitiveElement
    PE_PanelToolBar = ... # type: QStyle.PrimitiveElement
    PE_PanelLineEdit = ... # type: QStyle.PrimitiveElement
    PE_IndicatorArrowDown = ... # type: QStyle.PrimitiveElement
    PE_IndicatorArrowLeft = ... # type: QStyle.PrimitiveElement
    PE_IndicatorArrowRight = ... # type: QStyle.PrimitiveElement
    PE_IndicatorArrowUp = ... # type: QStyle.PrimitiveElement
    PE_IndicatorBranch = ... # type: QStyle.PrimitiveElement
    PE_IndicatorButtonDropDown = ... # type: QStyle.PrimitiveElement
    PE_IndicatorViewItemCheck = ... # type: QStyle.PrimitiveElement
    PE_IndicatorCheckBox = ... # type: QStyle.PrimitiveElement
    PE_IndicatorDockWidgetResizeHandle = ... # type: QStyle.PrimitiveElement
    PE_IndicatorHeaderArrow = ... # type: QStyle.PrimitiveElement
    PE_IndicatorMenuCheckMark = ... # type: QStyle.PrimitiveElement
    PE_IndicatorProgressChunk = ... # type: QStyle.PrimitiveElement
    PE_IndicatorRadioButton = ... # type: QStyle.PrimitiveElement
    PE_IndicatorSpinDown = ... # type: QStyle.PrimitiveElement
    PE_IndicatorSpinMinus = ... # type: QStyle.PrimitiveElement
    PE_IndicatorSpinPlus = ... # type: QStyle.PrimitiveElement
    PE_IndicatorSpinUp = ... # type: QStyle.PrimitiveElement
    PE_IndicatorToolBarHandle = ... # type: QStyle.PrimitiveElement
    PE_IndicatorToolBarSeparator = ... # type: QStyle.PrimitiveElement
    PE_PanelTipLabel = ... # type: QStyle.PrimitiveElement
    PE_IndicatorTabTear = ... # type: QStyle.PrimitiveElement
    PE_PanelScrollAreaCorner = ... # type: QStyle.PrimitiveElement
    PE_Widget = ... # type: QStyle.PrimitiveElement
    PE_IndicatorColumnViewArrow = ... # type: QStyle.PrimitiveElement
    PE_FrameStatusBarItem = ... # type: QStyle.PrimitiveElement
    PE_IndicatorItemViewItemCheck = ... # type: QStyle.PrimitiveElement
    PE_IndicatorItemViewItemDrop = ... # type: QStyle.PrimitiveElement
    PE_PanelItemViewItem = ... # type: QStyle.PrimitiveElement
    PE_PanelItemViewRow = ... # type: QStyle.PrimitiveElement
    PE_PanelStatusBar = ... # type: QStyle.PrimitiveElement
    PE_IndicatorTabClose = ... # type: QStyle.PrimitiveElement
    PE_PanelMenu = ... # type: QStyle.PrimitiveElement
    PE_IndicatorTabTearLeft = ... # type: QStyle.PrimitiveElement
    PE_IndicatorTabTearRight = ... # type: QStyle.PrimitiveElement
    PE_CustomBase = ... # type: QStyle.PrimitiveElement

class StateFlag(int):
    State_None = ... # type: QStyle.StateFlag
    State_Enabled = ... # type: QStyle.StateFlag
    State_Raised = ... # type: QStyle.StateFlag
    State_Sunken = ... # type: QStyle.StateFlag
    State_Off = ... # type: QStyle.StateFlag
    State_NoChange = ... # type: QStyle.StateFlag
    State_On = ... # type: QStyle.StateFlag
    State_DownArrow = ... # type: QStyle.StateFlag
    State_Horizontal = ... # type: QStyle.StateFlag
    State_HasFocus = ... # type: QStyle.StateFlag
    State_Top = ... # type: QStyle.StateFlag
    State_Bottom = ... # type: QStyle.StateFlag
    State_FocusAtBorder = ... # type: QStyle.StateFlag
    State_AutoRaise = ... # type: QStyle.StateFlag
    State_MouseOver = ... # type: QStyle.StateFlag
    State_UpArrow = ... # type: QStyle.StateFlag
    State_Selected = ... # type: QStyle.StateFlag
    State_Active = ... # type: QStyle.StateFlag
    State_Open = ... # type: QStyle.StateFlag
    State_Children = ... # type: QStyle.StateFlag
    State_Item = ... # type: QStyle.StateFlag
    State_Sibling = ... # type: QStyle.StateFlag
    State_Editing = ... # type: QStyle.StateFlag
    State_KeyboardFocusChange = ... # type: QStyle.StateFlag
    State_ReadOnly = ... # type: QStyle.StateFlag
```

```
        State_Window = ... # type: QStyle.StateFlag
        State_Small = ... # type: QStyle.StateFlag
        State_Mini = ... # type: QStyle.StateFlag

    class State(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> 'QStyle.State': ...
        def __xor__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> 'QStyle.State': ...
        def __ior__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> 'QStyle.State': ...
        def __or__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> 'QStyle.State': ...
        def __iand__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> 'QStyle.State': ...
        def __and__(self, f: typing.Union['QStyle.State', 'QStyle.StateFlag']) -> 'QStyle.State': ...
        def __invert__(self) -> 'QStyle.State': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    class SubControls(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> 'QStyle.SubControls': ...
        def __xor__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> 'QStyle.SubControls': ...
        def __ior__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> 'QStyle.SubControls': ...
        def __or__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> 'QStyle.SubControls': ...
        def __iand__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> 'QStyle.SubControls': ...
        def __and__(self, f: typing.Union['QStyle.SubControls', 'QStyle.SubControl']) -> 'QStyle.SubControls': ...
        def __invert__(self) -> 'QStyle.SubControls': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self) -> None: ...

    def proxy(self) -> typing.Optional['QStyle']: ...
    def combinedLayoutSpacing(self, controls1: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType'], controls2: 
typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType'], orientation: QtCore.Qt.Orientation, option: 
typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] = ...) -> int: ...
    def layoutSpacing(self, control1: 'QSizePolicy.ControlType', control2: 'QSizePolicy.ControlType', orientation: 
QtCore.Qt.Orientation, option: typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] = ...) -> int: ...
    @staticmethod
    def alignedRect(direction: QtCore.Qt.LayoutDirection, alignment: typing.Union[QtCore.Qt.Alignment, 
QtCore.Qt.AlignmentFlag], size: QtCore.QSize, rectangle: QtCore.QRect) -> QtCore.QRect: ...
    @staticmethod
    def visualAlignment(direction: QtCore.Qt.LayoutDirection, alignment: typing.Union[QtCore.Qt.Alignment, 
QtCore.Qt.AlignmentFlag]) -> QtCore.Qt.Alignment: ...
    @staticmethod
    def sliderValueFromPosition(min: int, max: int, position: int, span: int, upsideDown: bool = ...) -> int: ...
    @staticmethod
    def sliderPositionFromValue(min: int, max: int, logicalValue: int, span: int, upsideDown: bool = ...) -> int: ...
    @staticmethod
    def visualPos(direction: QtCore.Qt.LayoutDirection, boundingRect: QtCore.QRect, logicalPos: QtCore.QPoint) -> 
QtCore.QPoint: ...
    @staticmethod
    def visualRect(direction: QtCore.Qt.LayoutDirection, boundingRect: QtCore.QRect, logicalRect: QtCore.QRect) -> 
QtCore.QRect: ...
```

```
    def generatedIconPixmap(self, iconMode: QtGui.QIcon.Mode, pixmap: QtGui.QPixmap, opt: 
typing.Optional['QStyleOption']) -> QtGui.QPixmap: ...
    def standardIcon(self, standardIcon: 'QStyle.StandardPixmap', option: typing.Optional['QStyleOption'] = ..., widget: 
typing.Optional[QWidget] = ...) -> QtGui.QIcon: ...
    def standardPixmap(self, standardPixmap: 'QStyle.StandardPixmap', option: typing.Optional['QStyleOption'] = ..., widget: 
typing.Optional[QWidget] = ...) -> QtGui.QPixmap: ...
    def styleHint(self, stylehint: 'QStyle.StyleHint', option: typing.Optional['QStyleOption'] = ..., widget: 
typing.Optional[QWidget] = ..., returnData: typing.Optional['QStyleHintReturn'] = ...) -> int: ...
    def sizeFromContents(self, ct: 'QStyle.ContentsType', opt: typing.Optional['QStyleOption'], contentsSize: QtCore.QSize, 
widget: typing.Optional[QWidget] = ...) -> QtCore.QSize: ...
    def pixelMetric(self, metric: 'QStyle.PixelMetric', option: typing.Optional['QStyleOption'] = ..., widget: 
typing.Optional[QWidget] = ...) -> int: ...
    def subControlRect(self, cc: 'QStyle.ComplexControl', opt: typing.Optional['QStyleOptionComplex'], sc: 'QStyle.SubControl', 
widget: typing.Optional[QWidget] = ...) -> QtCore.QRect: ...
    def hitTestComplexControl(self, cc: 'QStyle.ComplexControl', opt: typing.Optional['QStyleOptionComplex'], pt: 
QtCore.QPoint, widget: typing.Optional[QWidget] = ...) -> 'QStyle.SubControl': ...
    def drawComplexControl(self, cc: 'QStyle.ComplexControl', opt: typing.Optional['QStyleOptionComplex'], p: 
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def subElementRect(self, subElement: 'QStyle.SubElement', option: typing.Optional['QStyleOption'], widget: 
typing.Optional[QWidget] = ...) -> QtCore.QRect: ...
    def drawControl(self, element: 'QStyle.ControlElement', opt: typing.Optional['QStyleOption'], p: 
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def drawPrimitive(self, pe: 'QStyle.PrimitiveElement', opt: typing.Optional['QStyleOption'], p: 
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def standardPalette(self) -> QtGui.QPalette: ...
    def drawItemPixmap(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRect, alignment: int, pixmap: 
QtGui.QPixmap) -> None: ...
    def drawItemText(self, painter: typing.Optional[QtGui.QPainter], rectangle: QtCore.QRect, alignment: int, palette: 
QtGui.QPalette, enabled: bool, text: typing.Optional[str], textRole: QtGui.QPalette.ColorRole = ...) -> None: ...
    def itemPixmapRect(self, r: QtCore.QRect, flags: int, pixmap: QtGui.QPixmap) -> QtCore.QRect: ...
    def itemTextRect(self, fm: QtGui.QFontMetrics, r: QtCore.QRect, flags: int, enabled: bool, text: typing.Optional[str]) -> 
QtCore.QRect: ...
    @typing.overload
    def unpolish(self, a0: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def unpolish(self, a0: typing.Optional[QApplication]) -> None: ...
    @typing.overload
    def polish(self, a0: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def polish(self, a0: typing.Optional[QApplication]) -> None: ...
    @typing.overload
    def polish(self, a0: QtGui.QPalette) -> QtGui.QPalette: ...


class QCommonStyle(QStyle):

    def __init__(self) -> None: ...

    def layoutSpacing(self, control1: 'QSizePolicy.ControlType', control2: 'QSizePolicy.ControlType', orientation: 
QtCore.Qt.Orientation, option: typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] = ...) -> int: ...
    def standardIcon(self, standardIcon: QStyle.StandardPixmap, option: typing.Optional['QStyleOption'] = ..., widget: 
typing.Optional[QWidget] = ...) -> QtGui.QIcon: ...
    def generatedIconPixmap(self, iconMode: QtGui.QIcon.Mode, pixmap: QtGui.QPixmap, opt: 
typing.Optional['QStyleOption']) -> QtGui.QPixmap: ...
    def standardPixmap(self, sp: QStyle.StandardPixmap, option: typing.Optional['QStyleOption'] = ..., widget: 
typing.Optional[QWidget] = ...) -> QtGui.QPixmap: ...
    def styleHint(self, sh: QStyle.StyleHint, option: typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] = 
..., returnData: typing.Optional['QStyleHintReturn'] = ...) -> int: ...
    def pixelMetric(self, m: QStyle.PixelMetric, option: typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] 
= ...) -> int: ...
    def sizeFromContents(self, ct: QStyle.ContentsType, opt: typing.Optional['QStyleOption'], contentsSize: QtCore.QSize, 
widget: typing.Optional[QWidget] = ...) -> QtCore.QSize: ...
    def subControlRect(self, cc: QStyle.ComplexControl, opt: typing.Optional['QStyleOptionComplex'], sc: QStyle.SubControl, 
widget: typing.Optional[QWidget] = ...) -> QtCore.QRect: ...
    def hitTestComplexControl(self, cc: QStyle.ComplexControl, opt: typing.Optional['QStyleOptionComplex'], pt: 
QtCore.QPoint, widget: typing.Optional[QWidget] = ...) -> QStyle.SubControl: ...
    def drawComplexControl(self, cc: QStyle.ComplexControl, opt: typing.Optional['QStyleOptionComplex'], p: 
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def subElementRect(self, r: QStyle.SubElement, opt: typing.Optional['QStyleOption'], widget: typing.Optional[QWidget] = 
...) -> QtCore.QRect: ...
```

```
    def drawControl(self, element: QStyle.ControlElement, opt: typing.Optional['QStyleOption'], p: 
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def drawPrimitive(self, pe: QStyle.PrimitiveElement, opt: typing.Optional['QStyleOption'], p: 
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def unpolish(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def unpolish(self, application: typing.Optional[QApplication]) -> None: ...
    @typing.overload
    def polish(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def polish(self, app: typing.Optional[QApplication]) -> None: ...
    @typing.overload
    def polish(self, a0: QtGui.QPalette) -> QtGui.QPalette: ...


class QCompleter(QtCore.QObject):

    class ModelSorting(int):
        UnsortedModel = ... # type: QCompleter.ModelSorting
        CaseSensitivelySortedModel = ... # type: QCompleter.ModelSorting
        CaseInsensitivelySortedModel = ... # type: QCompleter.ModelSorting

    class CompletionMode(int):
        PopupCompletion = ... # type: QCompleter.CompletionMode
        UnfilteredPopupCompletion = ... # type: QCompleter.CompletionMode
        InlineCompletion = ... # type: QCompleter.CompletionMode

    @typing.overload
    def __init__(self, model: typing.Optional[QtCore.QAbstractItemModel], parent: typing.Optional[QtCore.QObject] = ...) -> 
None: ...
    @typing.overload
    def __init__(self, list: typing.Iterable[typing.Optional[str]], parent: typing.Optional[QtCore.QObject] = ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def filterMode(self) -> QtCore.Qt.MatchFlags: ...
    def setFilterMode(self, filterMode: typing.Union[QtCore.Qt.MatchFlags, QtCore.Qt.MatchFlag]) -> None: ...
    def setMaxVisibleItems(self, maxItems: int) -> None: ...
    def maxVisibleItems(self) -> int: ...
    highlighted: typing.ClassVar[QtCore.pyqtSignal]
    activated: typing.ClassVar[QtCore.pyqtSignal]
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def eventFilter(self, o: typing.Optional[QtCore.QObject], e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setWrapAround(self, wrap: bool) -> None: ...
    def setCompletionPrefix(self, prefix: typing.Optional[str]) -> None: ...
    def complete(self, rect: QtCore.QRect = ...) -> None: ...
    def wrapAround(self) -> bool: ...
    def splitPath(self, path: typing.Optional[str]) -> typing.List[str]: ...
    def pathFromIndex(self, index: QtCore.QModelIndex) -> str: ...
    def completionPrefix(self) -> str: ...
    def completionModel(self) -> typing.Optional[QtCore.QAbstractItemModel]: ...
    def currentCompletion(self) -> str: ...
    def currentIndex(self) -> QtCore.QModelIndex: ...
    def currentRow(self) -> int: ...
    def setCurrentRow(self, row: int) -> bool: ...
    def completionCount(self) -> int: ...
    def completionRole(self) -> int: ...
    def setCompletionRole(self, role: int) -> None: ...
    def completionColumn(self) -> int: ...
    def setCompletionColumn(self, column: int) -> None: ...
    def modelSorting(self) -> 'QCompleter.ModelSorting': ...
    def setModelSorting(self, sorting: 'QCompleter.ModelSorting') -> None: ...
    def caseSensitivity(self) -> QtCore.Qt.CaseSensitivity: ...
    def setCaseSensitivity(self, caseSensitivity: QtCore.Qt.CaseSensitivity) -> None: ...
    def setPopup(self, popup: typing.Optional[QAbstractItemView]) -> None: ...
    def popup(self) -> typing.Optional[QAbstractItemView]: ...
    def completionMode(self) -> 'QCompleter.CompletionMode': ...
    def setCompletionMode(self, mode: 'QCompleter.CompletionMode') -> None: ...
    def model(self) -> typing.Optional[QtCore.QAbstractItemModel]: ...
    def setModel(self, c: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    def setWidget(self, widget: typing.Optional[QWidget]) -> None: ...


class QDataWidgetMapper(QtCore.QObject):

    class SubmitPolicy(int):
        AutoSubmit = ... # type: QDataWidgetMapper.SubmitPolicy
        ManualSubmit = ... # type: QDataWidgetMapper.SubmitPolicy

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    currentIndexChanged: typing.ClassVar[QtCore.pyqtSignal]
    def toPrevious(self) -> None: ...
    def toNext(self) -> None: ...
    def toLast(self) -> None: ...
    def toFirst(self) -> None: ...
    def submit(self) -> bool: ...
    def setCurrentModelIndex(self, index: QtCore.QModelIndex) -> None: ...
    def setCurrentIndex(self, index: int) -> None: ...
    def revert(self) -> None: ...
    def currentIndex(self) -> int: ...
    def clearMapping(self) -> None: ...
    def mappedWidgetAt(self, section: int) -> typing.Optional[QWidget]: ...
    def mappedSection(self, widget: typing.Optional[QWidget]) -> int: ...
    def mappedPropertyName(self, widget: typing.Optional[QWidget]) -> QtCore.QByteArray: ...
    def removeMapping(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def addMapping(self, widget: typing.Optional[QWidget], section: int) -> None: ...
    @typing.overload
    def addMapping(self, widget: typing.Optional[QWidget], section: int, propertyName: typing.Union[QtCore.QByteArray,
bytes, bytearray]) -> None: ...
    def submitPolicy(self) -> 'QDataWidgetMapper.SubmitPolicy': ...
    def setSubmitPolicy(self, policy: 'QDataWidgetMapper.SubmitPolicy') -> None: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setOrientation(self, aOrientation: QtCore.Qt.Orientation) -> None: ...
    def rootIndex(self) -> QtCore.QModelIndex: ...
    def setRootIndex(self, index: QtCore.QModelIndex) -> None: ...
    def itemDelegate(self) -> typing.Optional[QAbstractItemDelegate]: ...
    def setItemDelegate(self, delegate: typing.Optional[QAbstractItemDelegate]) -> None: ...
    def model(self) -> typing.Optional[QtCore.QAbstractItemModel]: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...


class QDateTimeEdit(QAbstractSpinBox):

    class Section(int):
        NoSection = ... # type: QDateTimeEdit.Section
        AmPmSection = ... # type: QDateTimeEdit.Section
        MSecSection = ... # type: QDateTimeEdit.Section
        SecondSection = ... # type: QDateTimeEdit.Section
        MinuteSection = ... # type: QDateTimeEdit.Section
        HourSection = ... # type: QDateTimeEdit.Section
        DaySection = ... # type: QDateTimeEdit.Section
        MonthSection = ... # type: QDateTimeEdit.Section
        YearSection = ... # type: QDateTimeEdit.Section
        TimeSections_Mask = ... # type: QDateTimeEdit.Section
        DateSections_Mask = ... # type: QDateTimeEdit.Section

    class Sections(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> 'QDateTimeEdit.Sections': ...
        def __xor__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> 'QDateTimeEdit.Sections': ...
        def __ior__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> 'QDateTimeEdit.Sections': ...
        def __or__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> 'QDateTimeEdit.Sections': ...
        def __iand__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> 'QDateTimeEdit.Sections': ...
        def __and__(self, f: typing.Union['QDateTimeEdit.Sections', 'QDateTimeEdit.Section']) -> 'QDateTimeEdit.Sections': ...
        def __invert__(self) -> 'QDateTimeEdit.Sections': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, datetime: typing.Union[QtCore.QDateTime, datetime.datetime], parent: typing.Optional[QWidget] = ...) -
> None: ...
    @typing.overload
    def __init__(self, date: typing.Union[QtCore.QDate, datetime.date], parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, time: typing.Union[QtCore.QTime, datetime.time], parent: typing.Optional[QWidget] = ...) -> None: ...

    def setCalendar(self, calendar: QtCore.QCalendar) -> None: ...
    def calendar(self) -> QtCore.QCalendar: ...
    def setTimeSpec(self, spec: QtCore.Qt.TimeSpec) -> None: ...
    def timeSpec(self) -> QtCore.Qt.TimeSpec: ...
    def setCalendarWidget(self, calendarWidget: typing.Optional[QCalendarWidget]) -> None: ...
    def calendarWidget(self) -> typing.Optional[QCalendarWidget]: ...
    def setDateTimeRange(self, min: typing.Union[QtCore.QDateTime, datetime.datetime], max:
typing.Union[QtCore.QDateTime, datetime.datetime]) -> None: ...
    def setMaximumDateTime(self, dt: typing.Union[QtCore.QDateTime, datetime.datetime]) -> None: ...
    def clearMaximumDateTime(self) -> None: ...
    def maximumDateTime(self) -> QtCore.QDateTime: ...
    def setMinimumDateTime(self, dt: typing.Union[QtCore.QDateTime, datetime.datetime]) -> None: ...
    def clearMinimumDateTime(self) -> None: ...
    def minimumDateTime(self) -> QtCore.QDateTime: ...
    def stepEnabled(self) -> QAbstractSpinBox.StepEnabled: ...
    def textFromDateTime(self, dt: typing.Union[QtCore.QDateTime, datetime.datetime]) -> str: ...
    def dateTimeFromText(self, text: typing.Optional[str]) -> QtCore.QDateTime: ...
    def fixup(self, input: typing.Optional[str]) -> str: ...
    def validate(self, input: typing.Optional[str], pos: int) -> typing.Tuple[QtGui.QValidator.State, str, int]: ...
    def paintEvent(self, event: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def mousePressEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionSpinBox']) -> None: ...
    def setTime(self, time: typing.Union[QtCore.QTime, datetime.time]) -> None: ...
    def setDate(self, date: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def setDateTime(self, dateTime: typing.Union[QtCore.QDateTime, datetime.datetime]) -> None: ...
    dateChanged: typing.ClassVar[QtCore.pyqtSignal]
    timeChanged: typing.ClassVar[QtCore.pyqtSignal]
    dateTimeChanged: typing.ClassVar[QtCore.pyqtSignal]
    def sectionCount(self) -> int: ...
    def setCurrentSectionIndex(self, index: int) -> None: ...
    def currentSectionIndex(self) -> int: ...
    def sectionAt(self, index: int) -> 'QDateTimeEdit.Section': ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def stepBy(self, steps: int) -> None: ...
    def clear(self) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setSelectedSection(self, section: 'QDateTimeEdit.Section') -> None: ...
    def setCalendarPopup(self, enable: bool) -> None: ...
    def calendarPopup(self) -> bool: ...
    def setDisplayFormat(self, format: typing.Optional[str]) -> None: ...
    def displayFormat(self) -> str: ...
    def sectionText(self, s: 'QDateTimeEdit.Section') -> str: ...
    def setCurrentSection(self, section: 'QDateTimeEdit.Section') -> None: ...
    def currentSection(self) -> 'QDateTimeEdit.Section': ...
    def displayedSections(self) -> 'QDateTimeEdit.Sections': ...
    def setTimeRange(self, min: typing.Union[QtCore.QTime, datetime.time], max: typing.Union[QtCore.QTime,
datetime.time]) -> None: ...
    def clearMaximumTime(self) -> None: ...
    def setMaximumTime(self, max: typing.Union[QtCore.QTime, datetime.time]) -> None: ...
    def maximumTime(self) -> QtCore.QTime: ...
    def clearMinimumTime(self) -> None: ...
    def setMinimumTime(self, min: typing.Union[QtCore.QTime, datetime.time]) -> None: ...
    def minimumTime(self) -> QtCore.QTime: ...
    def setDateRange(self, min: typing.Union[QtCore.QDate, datetime.date], max: typing.Union[QtCore.QDate,
datetime.date]) -> None: ...
    def clearMaximumDate(self) -> None: ...
    def setMaximumDate(self, max: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def maximumDate(self) -> QtCore.QDate: ...
    def clearMinimumDate(self) -> None: ...
    def setMinimumDate(self, min: typing.Union[QtCore.QDate, datetime.date]) -> None: ...
    def minimumDate(self) -> QtCore.QDate: ...
    def time(self) -> QtCore.QTime: ...
    def date(self) -> QtCore.QDate: ...
    def dateTime(self) -> QtCore.QDateTime: ...


class QTimeEdit(QDateTimeEdit):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, time: typing.Union[QtCore.QTime, datetime.time], parent: typing.Optional[QWidget] = ...) -> None: ...


class QDateEdit(QDateTimeEdit):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, date: typing.Union[QtCore.QDate, datetime.date], parent: typing.Optional[QWidget] = ...) -> None: ...


class QDesktopWidget(QWidget):

    def __init__(self) -> None: ...

    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    primaryScreenChanged: typing.ClassVar[QtCore.pyqtSignal]
    screenCountChanged: typing.ClassVar[QtCore.pyqtSignal]
    workAreaResized: typing.ClassVar[QtCore.pyqtSignal]
    resized: typing.ClassVar[QtCore.pyqtSignal]
    @typing.overload
    def availableGeometry(self, screen: int = ...) -> QtCore.QRect: ...
    @typing.overload
    def availableGeometry(self, widget: typing.Optional[QWidget]) -> QtCore.QRect: ...
    @typing.overload
    def availableGeometry(self, point: QtCore.QPoint) -> QtCore.QRect: ...
    @typing.overload
    def screenGeometry(self, screen: int = ...) -> QtCore.QRect: ...
    @typing.overload
    def screenGeometry(self, widget: typing.Optional[QWidget]) -> QtCore.QRect: ...
    @typing.overload
    def screenGeometry(self, point: QtCore.QPoint) -> QtCore.QRect: ...
    def screenCount(self) -> int: ...
    def screen(self, screen: int = ...) -> typing.Optional[QWidget]: ...
    @typing.overload
    def screenNumber(self, widget: typing.Optional[QWidget] = ...) -> int: ...
    @typing.overload
    def screenNumber(self, a0: QtCore.QPoint) -> int: ...
    def primaryScreen(self) -> int: ...
    def isVirtualDesktop(self) -> bool: ...


class QDial(QAbstractSlider):
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def sliderChange(self, change: QAbstractSlider.SliderChange) -> None: ...
    def mouseMoveEvent(self, me: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, me: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, me: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, pe: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, re: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionSlider']) -> None: ...
    def setWrapping(self, on: bool) -> None: ...
    def setNotchesVisible(self, visible: bool) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def notchesVisible(self) -> bool: ...
    def notchTarget(self) -> float: ...
    def setNotchTarget(self, target: float) -> None: ...
    def notchSize(self) -> int: ...
    def wrapping(self) -> bool: ...


class QDialogButtonBox(QWidget):

    class StandardButton(int):
        NoButton = ... # type: QDialogButtonBox.StandardButton
        Ok = ... # type: QDialogButtonBox.StandardButton
        Save = ... # type: QDialogButtonBox.StandardButton
        SaveAll = ... # type: QDialogButtonBox.StandardButton
        Open = ... # type: QDialogButtonBox.StandardButton
        Yes = ... # type: QDialogButtonBox.StandardButton
        YesToAll = ... # type: QDialogButtonBox.StandardButton
        No = ... # type: QDialogButtonBox.StandardButton
        NoToAll = ... # type: QDialogButtonBox.StandardButton
        Abort = ... # type: QDialogButtonBox.StandardButton
        Retry = ... # type: QDialogButtonBox.StandardButton
        Ignore = ... # type: QDialogButtonBox.StandardButton
        Close = ... # type: QDialogButtonBox.StandardButton
        Cancel = ... # type: QDialogButtonBox.StandardButton
        Discard = ... # type: QDialogButtonBox.StandardButton
        Help = ... # type: QDialogButtonBox.StandardButton
        Apply = ... # type: QDialogButtonBox.StandardButton
        Reset = ... # type: QDialogButtonBox.StandardButton
        RestoreDefaults = ... # type: QDialogButtonBox.StandardButton

    class ButtonRole(int):
        InvalidRole = ... # type: QDialogButtonBox.ButtonRole
        AcceptRole = ... # type: QDialogButtonBox.ButtonRole
        RejectRole = ... # type: QDialogButtonBox.ButtonRole
        DestructiveRole = ... # type: QDialogButtonBox.ButtonRole
        ActionRole = ... # type: QDialogButtonBox.ButtonRole
        HelpRole = ... # type: QDialogButtonBox.ButtonRole
        YesRole = ... # type: QDialogButtonBox.ButtonRole
        NoRole = ... # type: QDialogButtonBox.ButtonRole
        ResetRole = ... # type: QDialogButtonBox.ButtonRole
        ApplyRole = ... # type: QDialogButtonBox.ButtonRole

    class ButtonLayout(int):
        WinLayout = ... # type: QDialogButtonBox.ButtonLayout
        MacLayout = ... # type: QDialogButtonBox.ButtonLayout
        KdeLayout = ... # type: QDialogButtonBox.ButtonLayout
        GnomeLayout = ... # type: QDialogButtonBox.ButtonLayout
        AndroidLayout = ... # type: QDialogButtonBox.ButtonLayout

    class StandardButtons(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) -> None: ...
        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) ->
'QDialogButtonBox.StandardButtons': ...
        def __xor__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) ->
'QDialogButtonBox.StandardButtons': ...
        def __ior__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) ->
'QDialogButtonBox.StandardButtons': ...
        def __or__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) ->
'QDialogButtonBox.StandardButtons': ...
        def __iand__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) ->
'QDialogButtonBox.StandardButtons': ...
        def __and__(self, f: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton']) ->
'QDialogButtonBox.StandardButtons': ...
        def __invert__(self) -> 'QDialogButtonBox.StandardButtons': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, orientation: QtCore.Qt.Orientation, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, buttons: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton'], parent:
typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, buttons: typing.Union['QDialogButtonBox.StandardButtons', 'QDialogButtonBox.StandardButton'],
orientation: QtCore.Qt.Orientation, parent: typing.Optional[QWidget] = ...) -> None: ...

    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def changeEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    rejected: typing.ClassVar[QtCore.pyqtSignal]
    helpRequested: typing.ClassVar[QtCore.pyqtSignal]
    clicked: typing.ClassVar[QtCore.pyqtSignal]
    accepted: typing.ClassVar[QtCore.pyqtSignal]
    def centerButtons(self) -> bool: ...
    def setCenterButtons(self, center: bool) -> None: ...
    def button(self, which: 'QDialogButtonBox.StandardButton') -> typing.Optional[QPushButton]: ...
    def standardButton(self, button: typing.Optional[QAbstractButton]) -> 'QDialogButtonBox.StandardButton': ...
    def standardButtons(self) -> 'QDialogButtonBox.StandardButtons': ...
    def setStandardButtons(self, buttons: typing.Union['QDialogButtonBox.StandardButtons',
'QDialogButtonBox.StandardButton']) -> None: ...
    def buttonRole(self, button: typing.Optional[QAbstractButton]) -> 'QDialogButtonBox.ButtonRole': ...
    def buttons(self) -> typing.List[QAbstractButton]: ...
    def clear(self) -> None: ...
    def removeButton(self, button: typing.Optional[QAbstractButton]) -> None: ...
    @typing.overload
    def addButton(self, button: typing.Optional[QAbstractButton], role: 'QDialogButtonBox.ButtonRole') -> None: ...
    @typing.overload
    def addButton(self, text: typing.Optional[str], role: 'QDialogButtonBox.ButtonRole') -> typing.Optional[QPushButton]: ...
    @typing.overload
    def addButton(self, button: 'QDialogButtonBox.StandardButton') -> typing.Optional[QPushButton]: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setOrientation(self, orientation: QtCore.Qt.Orientation) -> None: ...


class QDirModel(QtCore.QAbstractItemModel):

    class Roles(int):
        FileIconRole = ... # type: QDirModel.Roles
        FilePathRole = ... # type: QDirModel.Roles
        FileNameRole = ... # type: QDirModel.Roles

    @typing.overload
    def __init__(self, nameFilters: typing.Iterable[typing.Optional[str]], filters: typing.Union[QtCore.QDir.Filters,
QtCore.QDir.Filter], sort: typing.Union[QtCore.QDir.SortFlags, QtCore.QDir.SortFlag], parent: typing.Optional[QtCore.QObject]
= ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def fileInfo(self, index: QtCore.QModelIndex) -> QtCore.QFileInfo: ...
    def fileIcon(self, index: QtCore.QModelIndex) -> QtGui.QIcon: ...
    def fileName(self, index: QtCore.QModelIndex) -> str: ...
    def filePath(self, index: QtCore.QModelIndex) -> str: ...
    def remove(self, index: QtCore.QModelIndex) -> bool: ...
    def rmdir(self, index: QtCore.QModelIndex) -> bool: ...
    def mkdir(self, parent: QtCore.QModelIndex, name: typing.Optional[str]) -> QtCore.QModelIndex: ...
    def isDir(self, index: QtCore.QModelIndex) -> bool: ...
    def refresh(self, parent: QtCore.QModelIndex = ...) -> None: ...
    def lazyChildCount(self) -> bool: ...
    def setLazyChildCount(self, enable: bool) -> None: ...
    def isReadOnly(self) -> bool: ...
    def setReadOnly(self, enable: bool) -> None: ...
    def resolveSymlinks(self) -> bool: ...
    def setResolveSymlinks(self, enable: bool) -> None: ...
    def sorting(self) -> QtCore.QDir.SortFlags: ...
    def setSorting(self, sort: typing.Union[QtCore.QDir.SortFlags, QtCore.QDir.SortFlag]) -> None: ...
    def filter(self) -> QtCore.QDir.Filters: ...
    def setFilter(self, filters: typing.Union[QtCore.QDir.Filters, QtCore.QDir.Filter]) -> None: ...
    def nameFilters(self) -> typing.List[str]: ...
    def setNameFilters(self, filters: typing.Iterable[typing.Optional[str]]) -> None: ...
    def iconProvider(self) -> typing.Optional['QFileIconProvider']: ...
    def setIconProvider(self, provider: typing.Optional['QFileIconProvider']) -> None: ...
    def supportedDropActions(self) -> QtCore.Qt.DropActions: ...
    def dropMimeData(self, data: typing.Optional[QtCore.QMimeData], action: QtCore.Qt.DropAction, row: int, column: int,
parent: QtCore.QModelIndex) -> bool: ...
    def mimeData(self, indexes: typing.Iterable[QtCore.QModelIndex]) -> typing.Optional[QtCore.QMimeData]: ...
    def mimeTypes(self) -> typing.List[str]: ...
    def sort(self, column: int, order: QtCore.Qt.SortOrder = ...) -> None: ...
    def flags(self, index: QtCore.QModelIndex) -> QtCore.Qt.ItemFlags: ...
    def hasChildren(self, parent: QtCore.QModelIndex = ...) -> bool: ...
    def headerData(self, section: int, orientation: QtCore.Qt.Orientation, role: int = ...) -> typing.Any: ...
    def setData(self, index: QtCore.QModelIndex, value: typing.Any, role: int = ...) -> bool: ...
    def data(self, index: QtCore.QModelIndex, role: int = ...) -> typing.Any: ...
    def columnCount(self, parent: QtCore.QModelIndex = ...) -> int: ...
    def rowCount(self, parent: QtCore.QModelIndex = ...) -> int: ...
    @typing.overload
    def parent(self, child: QtCore.QModelIndex) -> QtCore.QModelIndex: ...
    @typing.overload
    def parent(self) -> typing.Optional[QtCore.QObject]: ...
    @typing.overload
    def index(self, row: int, column: int, parent: QtCore.QModelIndex = ...) -> QtCore.QModelIndex: ...
    @typing.overload
    def index(self, path: typing.Optional[str], column: int = ...) -> QtCore.QModelIndex: ...


class QDockWidget(QWidget):

    class DockWidgetFeature(int):
        DockWidgetClosable = ... # type: QDockWidget.DockWidgetFeature
        DockWidgetMovable = ... # type: QDockWidget.DockWidgetFeature
        DockWidgetFloatable = ... # type: QDockWidget.DockWidgetFeature
        DockWidgetVerticalTitleBar = ... # type: QDockWidget.DockWidgetFeature
        AllDockWidgetFeatures = ... # type: QDockWidget.DockWidgetFeature
        NoDockWidgetFeatures = ... # type: QDockWidget.DockWidgetFeature

    class DockWidgetFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) ->
```

```
'QDockWidget.DockWidgetFeatures': ...
        def __xor__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> 
'QDockWidget.DockWidgetFeatures': ...
        def __ior__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> 
'QDockWidget.DockWidgetFeatures': ...
        def __or__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> 
'QDockWidget.DockWidgetFeatures': ...
        def __iand__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> 
'QDockWidget.DockWidgetFeatures': ...
        def __and__(self, f: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> 
'QDockWidget.DockWidgetFeatures': ...
        def __invert__(self) -> 'QDockWidget.DockWidgetFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, title: typing.Optional[str], parent: typing.Optional[QWidget] = ..., flags: 
typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags, 
QtCore.Qt.WindowType] = ...) -> None: ...

    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def paintEvent(self, event: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def closeEvent(self, event: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def changeEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionDockWidget']) -> None: ...
    visibilityChanged: typing.ClassVar[QtCore.pyqtSignal]
    dockLocationChanged: typing.ClassVar[QtCore.pyqtSignal]
    allowedAreasChanged: typing.ClassVar[QtCore.pyqtSignal]
    topLevelChanged: typing.ClassVar[QtCore.pyqtSignal]
    featuresChanged: typing.ClassVar[QtCore.pyqtSignal]
    def titleBarWidget(self) -> typing.Optional[QWidget]: ...
    def setTitleBarWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def toggleViewAction(self) -> typing.Optional[QAction]: ...
    def isAreaAllowed(self, area: QtCore.Qt.DockWidgetArea) -> bool: ...
    def allowedAreas(self) -> QtCore.Qt.DockWidgetAreas: ...
    def setAllowedAreas(self, areas: typing.Union[QtCore.Qt.DockWidgetAreas, QtCore.Qt.DockWidgetArea]) -> None: ...
    def isFloating(self) -> bool: ...
    def setFloating(self, floating: bool) -> None: ...
    def features(self) -> 'QDockWidget.DockWidgetFeatures': ...
    def setFeatures(self, features: typing.Union['QDockWidget.DockWidgetFeatures', 'QDockWidget.DockWidgetFeature']) -> 
None: ...
    def setWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def widget(self) -> typing.Optional[QWidget]: ...


class QErrorMessage(QDialog):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def done(self, a0: int) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    @typing.overload
    def showMessage(self, message: typing.Optional[str]) -> None: ...
    @typing.overload
    def showMessage(self, message: typing.Optional[str], type: typing.Optional[str]) -> None: ...
    @staticmethod
    def qtHandler() -> typing.Optional['QErrorMessage']: ...


class QFileDialog(QDialog):

    class Option(int):
        ShowDirsOnly = ... # type: QFileDialog.Option
        DontResolveSymlinks = ... # type: QFileDialog.Option
        DontConfirmOverwrite = ... # type: QFileDialog.Option
        DontUseSheet = ... # type: QFileDialog.Option
        DontUseNativeDialog = ... # type: QFileDialog.Option
        ReadOnly = ... # type: QFileDialog.Option
        HideNameFilterDetails = ... # type: QFileDialog.Option
        DontUseCustomDirectoryIcons = ... # type: QFileDialog.Option

    class DialogLabel(int):
        LookIn = ... # type: QFileDialog.DialogLabel
        FileName = ... # type: QFileDialog.DialogLabel
        FileType = ... # type: QFileDialog.DialogLabel
        Accept = ... # type: QFileDialog.DialogLabel
        Reject = ... # type: QFileDialog.DialogLabel

    class AcceptMode(int):
        AcceptOpen = ... # type: QFileDialog.AcceptMode
        AcceptSave = ... # type: QFileDialog.AcceptMode

    class FileMode(int):
        AnyFile = ... # type: QFileDialog.FileMode
        ExistingFile = ... # type: QFileDialog.FileMode
        Directory = ... # type: QFileDialog.FileMode
        ExistingFiles = ... # type: QFileDialog.FileMode
        DirectoryOnly = ... # type: QFileDialog.FileMode

    class ViewMode(int):
        Detail = ... # type: QFileDialog.ViewMode
        List = ... # type: QFileDialog.ViewMode

    class Options(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> 'QFileDialog.Options': ...
        def __xor__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> 'QFileDialog.Options': ...
        def __ior__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> 'QFileDialog.Options': ...
        def __or__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> 'QFileDialog.Options': ...
        def __iand__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> 'QFileDialog.Options': ...
        def __and__(self, f: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> 'QFileDialog.Options': ...
        def __invert__(self) -> 'QFileDialog.Options': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget], f: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType]) ->
None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory: typing.Optional[str]
= ..., filter: typing.Optional[str] = ...) -> None: ...

    @staticmethod
    def saveFileContent(fileContent: typing.Union[QtCore.QByteArray, bytes, bytearray], fileNameHint: typing.Optional[str] =
...) -> None: ...
    def selectedMimeTypeFilter(self) -> str: ...
    def supportedSchemes(self) -> typing.List[str]: ...
    def setSupportedSchemes(self, schemes: typing.Iterable[typing.Optional[str]]) -> None: ...
    @staticmethod
    def getSaveFileUrl(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory: QtCore.QUrl = ...,
filter: typing.Optional[str] = ..., initialFilter: typing.Optional[str] = ..., options: typing.Union['QFileDialog.Options',
'QFileDialog.Option'] = ..., supportedSchemes: typing.Iterable[typing.Optional[str]] = ...) -> typing.Tuple[QtCore.QUrl, str]:
...
    @staticmethod
    def getOpenFileUrls(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory: QtCore.QUrl = ...,
filter: typing.Optional[str] = ..., initialFilter: typing.Optional[str] = ..., options: typing.Union['QFileDialog.Options',
'QFileDialog.Option'] = ..., supportedSchemes: typing.Iterable[typing.Optional[str]] = ...) ->
typing.Tuple[typing.List[QtCore.QUrl], str]: ...
    @staticmethod
    def getOpenFileUrl(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory: QtCore.QUrl = ...,
filter: typing.Optional[str] = ..., initialFilter: typing.Optional[str] = ..., options: typing.Union['QFileDialog.Options',
'QFileDialog.Option'] = ..., supportedSchemes: typing.Iterable[typing.Optional[str]] = ...) -> typing.Tuple[QtCore.QUrl, str]:
...
    directoryUrlEntered: typing.ClassVar[QtCore.pyqtSignal]
    currentUrlChanged: typing.ClassVar[QtCore.pyqtSignal]
    urlsSelected: typing.ClassVar[QtCore.pyqtSignal]
    urlSelected: typing.ClassVar[QtCore.pyqtSignal]
    def selectMimeTypeFilter(self, filter: typing.Optional[str]) -> None: ...
    def mimeTypeFilters(self) -> typing.List[str]: ...
    def setMimeTypeFilters(self, filters: typing.Iterable[typing.Optional[str]]) -> None: ...
    def selectedUrls(self) -> typing.List[QtCore.QUrl]: ...
    def selectUrl(self, url: QtCore.QUrl) -> None: ...
    def directoryUrl(self) -> QtCore.QUrl: ...
    def setDirectoryUrl(self, directory: QtCore.QUrl) -> None: ...
    def setVisible(self, visible: bool) -> None: ...
    @typing.overload
    def open(self) -> None: ...
    @typing.overload
    def open(self, slot: PYQT_SLOT) -> None: ...
    def options(self) -> 'QFileDialog.Options': ...
    def setOptions(self, options: typing.Union['QFileDialog.Options', 'QFileDialog.Option']) -> None: ...
    def testOption(self, option: 'QFileDialog.Option') -> bool: ...
    def setOption(self, option: 'QFileDialog.Option', on: bool = ...) -> None: ...
    def setFilter(self, filters: typing.Union[QtCore.QDir.Filters, QtCore.QDir.Filter]) -> None: ...
    def filter(self) -> QtCore.QDir.Filters: ...
    def selectedNameFilter(self) -> str: ...
    def selectNameFilter(self, filter: typing.Optional[str]) -> None: ...
    def nameFilters(self) -> typing.List[str]: ...
    def setNameFilters(self, filters: typing.Iterable[typing.Optional[str]]) -> None: ...
    def setNameFilter(self, filter: typing.Optional[str]) -> None: ...
    def proxyModel(self) -> typing.Optional[QtCore.QAbstractProxyModel]: ...
    def setProxyModel(self, model: typing.Optional[QtCore.QAbstractProxyModel]) -> None: ...
    def restoreState(self, state: typing.Union[QtCore.QByteArray, bytes, bytearray]) -> bool: ...
    def saveState(self) -> QtCore.QByteArray: ...
    def sidebarUrls(self) -> typing.List[QtCore.QUrl]: ...
    def setSidebarUrls(self, urls: typing.Iterable[QtCore.QUrl]) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def accept(self) -> None: ...
    def done(self, result: int) -> None: ...
    @staticmethod
    def getSaveFileName(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory:
typing.Optional[str] = ..., filter: typing.Optional[str] = ..., initialFilter: typing.Optional[str] = ..., options:
typing.Union['QFileDialog.Options', 'QFileDialog.Option'] = ...) -> typing.Tuple[str, str]: ...
    @staticmethod
    def getOpenFileNames(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory:
typing.Optional[str] = ..., filter: typing.Optional[str] = ..., initialFilter: typing.Optional[str] = ..., options:
typing.Union['QFileDialog.Options', 'QFileDialog.Option'] = ...) -> typing.Tuple[typing.List[str], str]: ...
    @staticmethod
    def getOpenFileName(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory:
typing.Optional[str] = ..., filter: typing.Optional[str] = ..., initialFilter: typing.Optional[str] = ..., options:
typing.Union['QFileDialog.Options', 'QFileDialog.Option'] = ...) -> typing.Tuple[str, str]: ...
    @staticmethod
    def getExistingDirectoryUrl(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory:
QtCore.QUrl = ..., options: typing.Union['QFileDialog.Options', 'QFileDialog.Option'] = ..., supportedSchemes:
typing.Iterable[typing.Optional[str]] = ...) -> QtCore.QUrl: ...
    @staticmethod
    def getExistingDirectory(parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., directory:
typing.Optional[str] = ..., options: typing.Union['QFileDialog.Options', 'QFileDialog.Option'] = ...) -> str: ...
    fileSelected: typing.ClassVar[QtCore.pyqtSignal]
    filterSelected: typing.ClassVar[QtCore.pyqtSignal]
    filesSelected: typing.ClassVar[QtCore.pyqtSignal]
    directoryEntered: typing.ClassVar[QtCore.pyqtSignal]
    currentChanged: typing.ClassVar[QtCore.pyqtSignal]
    def labelText(self, label: 'QFileDialog.DialogLabel') -> str: ...
    def setLabelText(self, label: 'QFileDialog.DialogLabel', text: typing.Optional[str]) -> None: ...
    def iconProvider(self) -> typing.Optional['QFileIconProvider']: ...
    def setIconProvider(self, provider: typing.Optional['QFileIconProvider']) -> None: ...
    def itemDelegate(self) -> typing.Optional[QAbstractItemDelegate]: ...
    def setItemDelegate(self, delegate: typing.Optional[QAbstractItemDelegate]) -> None: ...
    def history(self) -> typing.List[str]: ...
    def setHistory(self, paths: typing.Iterable[typing.Optional[str]]) -> None: ...
    def defaultSuffix(self) -> str: ...
    def setDefaultSuffix(self, suffix: typing.Optional[str]) -> None: ...
    def acceptMode(self) -> 'QFileDialog.AcceptMode': ...
    def setAcceptMode(self, mode: 'QFileDialog.AcceptMode') -> None: ...
    def fileMode(self) -> 'QFileDialog.FileMode': ...
    def setFileMode(self, mode: 'QFileDialog.FileMode') -> None: ...
    def viewMode(self) -> 'QFileDialog.ViewMode': ...
    def setViewMode(self, mode: 'QFileDialog.ViewMode') -> None: ...
    def selectedFiles(self) -> typing.List[str]: ...
    def selectFile(self, filename: typing.Optional[str]) -> None: ...
    def directory(self) -> QtCore.QDir: ...
    @typing.overload
    def setDirectory(self, directory: typing.Optional[str]) -> None: ...
    @typing.overload
    def setDirectory(self, adirectory: QtCore.QDir) -> None: ...


class QFileIconProvider(PyQt5.sipsimplewrapper):

    class Option(int):
        DontUseCustomDirectoryIcons = ... # type: QFileIconProvider.Option

    class IconType(int):
        Computer = ... # type: QFileIconProvider.IconType
        Desktop = ... # type: QFileIconProvider.IconType
        Trashcan = ... # type: QFileIconProvider.IconType
        Network = ... # type: QFileIconProvider.IconType
        Drive = ... # type: QFileIconProvider.IconType
        Folder = ... # type: QFileIconProvider.IconType
        File = ... # type: QFileIconProvider.IconType

    class Options(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) ->
'QFileIconProvider.Options': ...
        def __xor__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) ->
'QFileIconProvider.Options': ...
        def __ior__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) -> 'QFileIconProvider.Options':
...
        def __or__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) -> 'QFileIconProvider.Options':
...
        def __iand__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) ->
'QFileIconProvider.Options': ...
        def __and__(self, f: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) ->
'QFileIconProvider.Options': ...
        def __invert__(self) -> 'QFileIconProvider.Options': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self) -> None: ...

    def options(self) -> 'QFileIconProvider.Options': ...
    def setOptions(self, options: typing.Union['QFileIconProvider.Options', 'QFileIconProvider.Option']) -> None: ...
    def type(self, info: QtCore.QFileInfo) -> str: ...
    @typing.overload
    def icon(self, type: 'QFileIconProvider.IconType') -> QtGui.QIcon: ...
    @typing.overload
    def icon(self, info: QtCore.QFileInfo) -> QtGui.QIcon: ...
```

```
class QFileSystemModel(QtCore.QAbstractItemModel):

    class Option(int):
        DontWatchForChanges = ... # type: QFileSystemModel.Option
        DontResolveSymlinks = ... # type: QFileSystemModel.Option
        DontUseCustomDirectoryIcons = ... # type: QFileSystemModel.Option

    class Roles(int):
        FileIconRole = ... # type: QFileSystemModel.Roles
        FilePathRole = ... # type: QFileSystemModel.Roles
        FileNameRole = ... # type: QFileSystemModel.Roles
        FilePermissions = ... # type: QFileSystemModel.Roles

    class Options(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) ->
'QFileSystemModel.Options': ...
        def __xor__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) ->
'QFileSystemModel.Options': ...
        def __ior__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) ->
'QFileSystemModel.Options': ...
        def __or__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) ->
'QFileSystemModel.Options': ...
        def __iand__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) ->
'QFileSystemModel.Options': ...
        def __and__(self, f: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) ->
'QFileSystemModel.Options': ...
        def __invert__(self) -> 'QFileSystemModel.Options': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def options(self) -> 'QFileSystemModel.Options': ...
    def setOptions(self, options: typing.Union['QFileSystemModel.Options', 'QFileSystemModel.Option']) -> None: ...
    def testOption(self, option: 'QFileSystemModel.Option') -> bool: ...
    def setOption(self, option: 'QFileSystemModel.Option', on: bool = ...) -> None: ...
    def sibling(self, row: int, column: int, idx: QtCore.QModelIndex) -> QtCore.QModelIndex: ...
    def timerEvent(self, event: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    directoryLoaded: typing.ClassVar[QtCore.pyqtSignal]
    rootPathChanged: typing.ClassVar[QtCore.pyqtSignal]
    fileRenamed: typing.ClassVar[QtCore.pyqtSignal]
    def remove(self, index: QtCore.QModelIndex) -> bool: ...
    def fileInfo(self, aindex: QtCore.QModelIndex) -> QtCore.QFileInfo: ...
    def fileIcon(self, aindex: QtCore.QModelIndex) -> QtGui.QIcon: ...
    def fileName(self, aindex: QtCore.QModelIndex) -> str: ...
    def rmdir(self, index: QtCore.QModelIndex) -> bool: ...
    def permissions(self, index: QtCore.QModelIndex) -> QtCore.QFileDevice.Permissions: ...
    def mkdir(self, parent: QtCore.QModelIndex, name: typing.Optional[str]) -> QtCore.QModelIndex: ...
    def lastModified(self, index: QtCore.QModelIndex) -> QtCore.QDateTime: ...
    def type(self, index: QtCore.QModelIndex) -> str: ...
    def size(self, index: QtCore.QModelIndex) -> int: ...
    def isDir(self, index: QtCore.QModelIndex) -> bool: ...
    def filePath(self, index: QtCore.QModelIndex) -> str: ...
    def nameFilters(self) -> typing.List[str]: ...
    def setNameFilters(self, filters: typing.Iterable[typing.Optional[str]]) -> None: ...
    def nameFilterDisables(self) -> bool: ...
    def setNameFilterDisables(self, enable: bool) -> None: ...
    def isReadOnly(self) -> bool: ...
    def setReadOnly(self, enable: bool) -> None: ...
    def resolveSymlinks(self) -> bool: ...
    def setResolveSymlinks(self, enable: bool) -> None: ...
    def filter(self) -> QtCore.QDir.Filters: ...
    def setFilter(self, filters: typing.Union[QtCore.QDir.Filters, QtCore.QDir.Filter]) -> None: ...
    def iconProvider(self) -> typing.Optional[QFileIconProvider]: ...
    def setIconProvider(self, provider: typing.Optional[QFileIconProvider]) -> None: ...
    def rootDirectory(self) -> QtCore.QDir: ...
    def rootPath(self) -> str: ...
    def setRootPath(self, path: typing.Optional[str]) -> QtCore.QModelIndex: ...
    def supportedDropActions(self) -> QtCore.Qt.DropActions: ...
    def dropMimeData(self, data: typing.Optional[QtCore.QMimeData], action: QtCore.Qt.DropAction, row: int, column: int,
parent: QtCore.QModelIndex) -> bool: ...
    def mimeData(self, indexes: typing.Iterable[QtCore.QModelIndex]) -> typing.Optional[QtCore.QMimeData]: ...
    def mimeTypes(self) -> typing.List[str]: ...
    def sort(self, column: int, order: QtCore.Qt.SortOrder = ...) -> None: ...
    def flags(self, index: QtCore.QModelIndex) -> QtCore.Qt.ItemFlags: ...
    def headerData(self, section: int, orientation: QtCore.Qt.Orientation, role: int = ...) -> typing.Any: ...
    def setData(self, index: QtCore.QModelIndex, value: typing.Any, role: int = ...) -> bool: ...
    def data(self, index: QtCore.QModelIndex, role: int = ...) -> typing.Any: ...
    def myComputer(self, role: int = ...) -> typing.Any: ...
    def columnCount(self, parent: QtCore.QModelIndex = ...) -> int: ...
    def rowCount(self, parent: QtCore.QModelIndex = ...) -> int: ...
    def fetchMore(self, parent: QtCore.QModelIndex) -> None: ...
    def canFetchMore(self, parent: QtCore.QModelIndex) -> bool: ...
    def hasChildren(self, parent: QtCore.QModelIndex = ...) -> bool: ...
    def parent(self, child: QtCore.QModelIndex) -> QtCore.QModelIndex: ...
    @typing.overload
    def index(self, row: int, column: int, parent: QtCore.QModelIndex = ...) -> QtCore.QModelIndex: ...
    @typing.overload
    def index(self, path: typing.Optional[str], column: int = ...) -> QtCore.QModelIndex: ...


class QFocusFrame(QWidget):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOption']) -> None: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    def setWidget(self, widget: typing.Optional[QWidget]) -> None: ...


class QFontComboBox(QComboBox):

    class FontFilter(int):
        AllFonts = ... # type: QFontComboBox.FontFilter
        ScalableFonts = ... # type: QFontComboBox.FontFilter
        NonScalableFonts = ... # type: QFontComboBox.FontFilter
        MonospacedFonts = ... # type: QFontComboBox.FontFilter
        ProportionalFonts = ... # type: QFontComboBox.FontFilter

    class FontFilters(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) ->
'QFontComboBox.FontFilters': ...
        def __xor__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) ->
'QFontComboBox.FontFilters': ...
        def __ior__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) ->
```

```
'QFontComboBox.FontFilters': ...
        def __or__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) ->
'QFontComboBox.FontFilters': ...
        def __iand__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) ->
'QFontComboBox.FontFilters': ...
        def __and__(self, f: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) ->
'QFontComboBox.FontFilters': ...
        def __invert__(self) -> 'QFontComboBox.FontFilters': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    currentFontChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setCurrentFont(self, f: QtGui.QFont) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def currentFont(self) -> QtGui.QFont: ...
    def setFontFilters(self, filters: typing.Union['QFontComboBox.FontFilters', 'QFontComboBox.FontFilter']) -> None: ...
    def writingSystem(self) -> QtGui.QFontDatabase.WritingSystem: ...
    def setWritingSystem(self, a0: QtGui.QFontDatabase.WritingSystem) -> None: ...
    def fontFilters(self) -> 'QFontComboBox.FontFilters': ...


class QFontDialog(QDialog):

    class FontDialogOption(int):
        NoButtons = ... # type: QFontDialog.FontDialogOption
        DontUseNativeDialog = ... # type: QFontDialog.FontDialogOption
        ScalableFonts = ... # type: QFontDialog.FontDialogOption
        NonScalableFonts = ... # type: QFontDialog.FontDialogOption
        MonospacedFonts = ... # type: QFontDialog.FontDialogOption
        ProportionalFonts = ... # type: QFontDialog.FontDialogOption

    class FontDialogOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) ->
'QFontDialog.FontDialogOptions': ...
        def __xor__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) ->
'QFontDialog.FontDialogOptions': ...
        def __ior__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) ->
'QFontDialog.FontDialogOptions': ...
        def __or__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) ->
'QFontDialog.FontDialogOptions': ...
        def __iand__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) ->
'QFontDialog.FontDialogOptions': ...
        def __and__(self, f: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) ->
'QFontDialog.FontDialogOptions': ...
        def __invert__(self) -> 'QFontDialog.FontDialogOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, initial: QtGui.QFont, parent: typing.Optional[QWidget] = ...) -> None: ...

    fontSelected: typing.ClassVar[QtCore.pyqtSignal]
    currentFontChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setVisible(self, visible: bool) -> None: ...
    @typing.overload
    def open(self) -> None: ...
    @typing.overload
    def open(self, slot: PYQT_SLOT) -> None: ...
    def options(self) -> 'QFontDialog.FontDialogOptions': ...
    def setOptions(self, options: typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption']) -> None: ...
    def testOption(self, option: 'QFontDialog.FontDialogOption') -> bool: ...
    def setOption(self, option: 'QFontDialog.FontDialogOption', on: bool = ...) -> None: ...
    def selectedFont(self) -> QtGui.QFont: ...
    def currentFont(self) -> QtGui.QFont: ...
    def setCurrentFont(self, font: QtGui.QFont) -> None: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def done(self, result: int) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    @typing.overload
    @staticmethod
    def getFont(initial: QtGui.QFont, parent: typing.Optional[QWidget] = ..., caption: typing.Optional[str] = ..., options:
typing.Union['QFontDialog.FontDialogOptions', 'QFontDialog.FontDialogOption'] = ...) -> typing.Tuple[QtGui.QFont,
typing.Optional[bool]]: ...
    @typing.overload
    @staticmethod
    def getFont(parent: typing.Optional[QWidget] = ...) -> typing.Tuple[QtGui.QFont, typing.Optional[bool]]: ...


class QFormLayout(QLayout):

    class ItemRole(int):
        LabelRole = ... # type: QFormLayout.ItemRole
        FieldRole = ... # type: QFormLayout.ItemRole
        SpanningRole = ... # type: QFormLayout.ItemRole

    class RowWrapPolicy(int):
        DontWrapRows = ... # type: QFormLayout.RowWrapPolicy
        WrapLongRows = ... # type: QFormLayout.RowWrapPolicy
        WrapAllRows = ... # type: QFormLayout.RowWrapPolicy

    class FieldGrowthPolicy(int):
        FieldsStayAtSizeHint = ... # type: QFormLayout.FieldGrowthPolicy
        ExpandingFieldsGrow = ... # type: QFormLayout.FieldGrowthPolicy
        AllNonFixedFieldsGrow = ... # type: QFormLayout.FieldGrowthPolicy

    class TakeRowResult(PyQt5.sipsimplewrapper):

        fieldItem = ... # type: QLayoutItem
        labelItem = ... # type: QLayoutItem

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, a0: 'QFormLayout.TakeRowResult') -> None: ...

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    @typing.overload
    def takeRow(self, row: int) -> 'QFormLayout.TakeRowResult': ...
    @typing.overload
    def takeRow(self, widget: typing.Optional[QWidget]) -> 'QFormLayout.TakeRowResult': ...
    @typing.overload
    def takeRow(self, layout: typing.Optional[QLayout]) -> 'QFormLayout.TakeRowResult': ...
    @typing.overload
    def removeRow(self, row: int) -> None: ...
    @typing.overload
    def removeRow(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def removeRow(self, layout: typing.Optional[QLayout]) -> None: ...
    def rowCount(self) -> int: ...
    def count(self) -> int: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def heightForWidth(self, width: int) -> int: ...
    def hasHeightForWidth(self) -> bool: ...
    def invalidate(self) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def setGeometry(self, rect: QtCore.QRect) -> None: ...
    def takeAt(self, index: int) -> typing.Optional[QLayoutItem]: ...
    def addItem(self, item: typing.Optional[QLayoutItem]) -> None: ...
    @typing.overload
    def labelForField(self, field: typing.Optional[QWidget]) -> typing.Optional[QWidget]: ...
    @typing.overload
    def labelForField(self, field: typing.Optional[QLayout]) -> typing.Optional[QWidget]: ...
    def getLayoutPosition(self, layout: typing.Optional[QLayout]) -> typing.Tuple[typing.Optional[int],
typing.Optional['QFormLayout.ItemRole']]: ...
    def getWidgetPosition(self, widget: typing.Optional[QWidget]) -> typing.Tuple[typing.Optional[int],
typing.Optional['QFormLayout.ItemRole']]: ...
    def getItemPosition(self, index: int) -> typing.Tuple[typing.Optional[int], typing.Optional['QFormLayout.ItemRole']]: ...
    @typing.overload
    def itemAt(self, row: int, role: 'QFormLayout.ItemRole') -> typing.Optional[QLayoutItem]: ...
    @typing.overload
    def itemAt(self, index: int) -> typing.Optional[QLayoutItem]: ...
    def setLayout(self, row: int, role: 'QFormLayout.ItemRole', layout: typing.Optional[QLayout]) -> None: ...
    def setWidget(self, row: int, role: 'QFormLayout.ItemRole', widget: typing.Optional[QWidget]) -> None: ...
    def setItem(self, row: int, role: 'QFormLayout.ItemRole', item: typing.Optional[QLayoutItem]) -> None: ...
    @typing.overload
    def insertRow(self, row: int, label: typing.Optional[QWidget], field: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def insertRow(self, row: int, label: typing.Optional[QWidget], field: typing.Optional[QLayout]) -> None: ...
    @typing.overload
    def insertRow(self, row: int, labelText: typing.Optional[str], field: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def insertRow(self, row: int, labelText: typing.Optional[str], field: typing.Optional[QLayout]) -> None: ...
    @typing.overload
    def insertRow(self, row: int, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def insertRow(self, row: int, layout: typing.Optional[QLayout]) -> None: ...
    @typing.overload
    def addRow(self, label: typing.Optional[QWidget], field: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def addRow(self, label: typing.Optional[QWidget], field: typing.Optional[QLayout]) -> None: ...
    @typing.overload
    def addRow(self, labelText: typing.Optional[str], field: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def addRow(self, labelText: typing.Optional[str], field: typing.Optional[QLayout]) -> None: ...
    @typing.overload
    def addRow(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def addRow(self, layout: typing.Optional[QLayout]) -> None: ...
    def setSpacing(self, a0: int) -> None: ...
    def spacing(self) -> int: ...
    def verticalSpacing(self) -> int: ...
    def setVerticalSpacing(self, spacing: int) -> None: ...
    def horizontalSpacing(self) -> int: ...
    def setHorizontalSpacing(self, spacing: int) -> None: ...
    def formAlignment(self) -> QtCore.Qt.Alignment: ...
    def setFormAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def labelAlignment(self) -> QtCore.Qt.Alignment: ...
    def setLabelAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def rowWrapPolicy(self) -> 'QFormLayout.RowWrapPolicy': ...
    def setRowWrapPolicy(self, policy: 'QFormLayout.RowWrapPolicy') -> None: ...
    def fieldGrowthPolicy(self) -> 'QFormLayout.FieldGrowthPolicy': ...
    def setFieldGrowthPolicy(self, policy: 'QFormLayout.FieldGrowthPolicy') -> None: ...


class QGesture(QtCore.QObject):

    class GestureCancelPolicy(int):
        CancelNone = ... # type: QGesture.GestureCancelPolicy
        CancelAllInContext = ... # type: QGesture.GestureCancelPolicy

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def gestureCancelPolicy(self) -> 'QGesture.GestureCancelPolicy': ...
    def setGestureCancelPolicy(self, policy: 'QGesture.GestureCancelPolicy') -> None: ...
    def unsetHotSpot(self) -> None: ...
    def hasHotSpot(self) -> bool: ...
    def setHotSpot(self, value: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def hotSpot(self) -> QtCore.QPointF: ...
    def state(self) -> QtCore.Qt.GestureState: ...
    def gestureType(self) -> QtCore.Qt.GestureType: ...


class QPanGesture(QGesture):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def setAcceleration(self, value: float) -> None: ...
    def setOffset(self, value: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def setLastOffset(self, value: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def acceleration(self) -> float: ...
    def delta(self) -> QtCore.QPointF: ...
    def offset(self) -> QtCore.QPointF: ...
    def lastOffset(self) -> QtCore.QPointF: ...


class QPinchGesture(QGesture):

    class ChangeFlag(int):
        ScaleFactorChanged = ... # type: QPinchGesture.ChangeFlag
        RotationAngleChanged = ... # type: QPinchGesture.ChangeFlag
        CenterPointChanged = ... # type: QPinchGesture.ChangeFlag

    class ChangeFlags(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> 
'QPinchGesture.ChangeFlags': ...
        def __xor__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> 
'QPinchGesture.ChangeFlags': ...
        def __ior__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> 
'QPinchGesture.ChangeFlags': ...
        def __or__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> 
'QPinchGesture.ChangeFlags': ...
        def __iand__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> 
'QPinchGesture.ChangeFlags': ...
        def __and__(self, f: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> 
'QPinchGesture.ChangeFlags': ...
        def __invert__(self) -> 'QPinchGesture.ChangeFlags': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def setRotationAngle(self, value: float) -> None: ...
    def setLastRotationAngle(self, value: float) -> None: ...
    def setTotalRotationAngle(self, value: float) -> None: ...
    def rotationAngle(self) -> float: ...
    def lastRotationAngle(self) -> float: ...
    def totalRotationAngle(self) -> float: ...
    def setScaleFactor(self, value: float) -> None: ...
    def setLastScaleFactor(self, value: float) -> None: ...
    def setTotalScaleFactor(self, value: float) -> None: ...
    def scaleFactor(self) -> float: ...
    def lastScaleFactor(self) -> float: ...
    def totalScaleFactor(self) -> float: ...
    def setCenterPoint(self, value: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def setLastCenterPoint(self, value: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def setStartCenterPoint(self, value: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def centerPoint(self) -> QtCore.QPointF: ...
    def lastCenterPoint(self) -> QtCore.QPointF: ...
    def startCenterPoint(self) -> QtCore.QPointF: ...
    def setChangeFlags(self, value: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> None: ...
    def changeFlags(self) -> 'QPinchGesture.ChangeFlags': ...
    def setTotalChangeFlags(self, value: typing.Union['QPinchGesture.ChangeFlags', 'QPinchGesture.ChangeFlag']) -> None: ...
    def totalChangeFlags(self) -> 'QPinchGesture.ChangeFlags': ...


class QSwipeGesture(QGesture):

    class SwipeDirection(int):
        NoDirection = ... # type: QSwipeGesture.SwipeDirection
        Left = ... # type: QSwipeGesture.SwipeDirection
        Right = ... # type: QSwipeGesture.SwipeDirection
        Up = ... # type: QSwipeGesture.SwipeDirection
        Down = ... # type: QSwipeGesture.SwipeDirection

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def setSwipeAngle(self, value: float) -> None: ...
    def swipeAngle(self) -> float: ...
    def verticalDirection(self) -> 'QSwipeGesture.SwipeDirection': ...
    def horizontalDirection(self) -> 'QSwipeGesture.SwipeDirection': ...


class QTapGesture(QGesture):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def setPosition(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def position(self) -> QtCore.QPointF: ...


class QTapAndHoldGesture(QGesture):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    @staticmethod
    def timeout() -> int: ...
    @staticmethod
    def setTimeout(msecs: int) -> None: ...
    def setPosition(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    def position(self) -> QtCore.QPointF: ...


class QGestureEvent(QtCore.QEvent):

    @typing.overload
    def __init__(self, gestures: typing.Iterable[QGesture]) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QGestureEvent') -> None: ...

    def mapToGraphicsScene(self, gesturePoint: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.QPointF: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    @typing.overload
    def ignore(self) -> None: ...
    @typing.overload
    def ignore(self, a0: typing.Optional[QGesture]) -> None: ...
    @typing.overload
    def ignore(self, a0: QtCore.Qt.GestureType) -> None: ...
    @typing.overload
    def accept(self) -> None: ...
    @typing.overload
    def accept(self, a0: typing.Optional[QGesture]) -> None: ...
    @typing.overload
    def accept(self, a0: QtCore.Qt.GestureType) -> None: ...
    @typing.overload
    def isAccepted(self) -> bool: ...
    @typing.overload
    def isAccepted(self, a0: typing.Optional[QGesture]) -> bool: ...
    @typing.overload
    def isAccepted(self, a0: QtCore.Qt.GestureType) -> bool: ...
    @typing.overload
    def setAccepted(self, accepted: bool) -> None: ...
    @typing.overload
    def setAccepted(self, a0: typing.Optional[QGesture], a1: bool) -> None: ...
    @typing.overload
    def setAccepted(self, a0: QtCore.Qt.GestureType, a1: bool) -> None: ...
    def canceledGestures(self) -> typing.List[QGesture]: ...
    def activeGestures(self) -> typing.List[QGesture]: ...
    def gesture(self, type: QtCore.Qt.GestureType) -> typing.Optional[QGesture]: ...
    def gestures(self) -> typing.List[QGesture]: ...


class QGestureRecognizer(PyQt5.sip.wrapper):

    class ResultFlag(int):
        Ignore = ... # type: QGestureRecognizer.ResultFlag
        MayBeGesture = ... # type: QGestureRecognizer.ResultFlag
        TriggerGesture = ... # type: QGestureRecognizer.ResultFlag
        FinishGesture = ... # type: QGestureRecognizer.ResultFlag
        CancelGesture = ... # type: QGestureRecognizer.ResultFlag
        ConsumeEventHint = ... # type: QGestureRecognizer.ResultFlag

    class Result(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) ->
'QGestureRecognizer.Result': ...
        def __xor__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) ->
'QGestureRecognizer.Result': ...
        def __ior__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) ->
'QGestureRecognizer.Result': ...
        def __or__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) ->
'QGestureRecognizer.Result': ...
        def __iand__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) ->
'QGestureRecognizer.Result': ...
        def __and__(self, f: typing.Union['QGestureRecognizer.Result', 'QGestureRecognizer.ResultFlag']) ->
'QGestureRecognizer.Result': ...
        def __invert__(self) -> 'QGestureRecognizer.Result': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QGestureRecognizer') -> None: ...

    @staticmethod
    def unregisterRecognizer(type: QtCore.Qt.GestureType) -> None: ...
    @staticmethod
    def registerRecognizer(recognizer: typing.Optional['QGestureRecognizer']) -> QtCore.Qt.GestureType: ...
    def reset(self, state: typing.Optional[QGesture]) -> None: ...
    def recognize(self, state: typing.Optional[QGesture], watched: typing.Optional[QtCore.QObject], event:
typing.Optional[QtCore.QEvent]) -> 'QGestureRecognizer.Result': ...
    def create(self, target: typing.Optional[QtCore.QObject]) -> typing.Optional[QGesture]: ...
```

```
class QGraphicsAnchor(QtCore.QObject):

    def sizePolicy(self) -> 'QSizePolicy.Policy': ...
    def setSizePolicy(self, policy: 'QSizePolicy.Policy') -> None: ...
    def spacing(self) -> float: ...
    def unsetSpacing(self) -> None: ...
    def setSpacing(self, spacing: float) -> None: ...


class QGraphicsLayoutItem(PyQt5.sip.wrapper):

    def __init__(self, parent: typing.Optional['QGraphicsLayoutItem'] = ..., isLayout: bool = ...) -> None: ...

    def setOwnedByLayout(self, ownedByLayout: bool) -> None: ...
    def setGraphicsItem(self, item: typing.Optional['QGraphicsItem']) -> None: ...
    def sizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def ownedByLayout(self) -> bool: ...
    def graphicsItem(self) -> typing.Optional['QGraphicsItem']: ...
    def maximumHeight(self) -> float: ...
    def maximumWidth(self) -> float: ...
    def preferredHeight(self) -> float: ...
    def preferredWidth(self) -> float: ...
    def minimumHeight(self) -> float: ...
    def minimumWidth(self) -> float: ...
    def isLayout(self) -> bool: ...
    def setParentLayoutItem(self, parent: typing.Optional['QGraphicsLayoutItem']) -> None: ...
    def parentLayoutItem(self) -> typing.Optional['QGraphicsLayoutItem']: ...
    def updateGeometry(self) -> None: ...
    def effectiveSizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def contentsRect(self) -> QtCore.QRectF: ...
    def getContentsMargins(self) -> typing.Tuple[typing.Optional[float], typing.Optional[float], typing.Optional[float],
typing.Optional[float]]: ...
    def geometry(self) -> QtCore.QRectF: ...
    def setGeometry(self, rect: QtCore.QRectF) -> None: ...
    def setMaximumHeight(self, height: float) -> None: ...
    def setMaximumWidth(self, width: float) -> None: ...
    def maximumSize(self) -> QtCore.QSizeF: ...
    @typing.overload
    def setMaximumSize(self, size: QtCore.QSizeF) -> None: ...
    @typing.overload
    def setMaximumSize(self, aw: float, ah: float) -> None: ...
    def setPreferredHeight(self, height: float) -> None: ...
    def setPreferredWidth(self, width: float) -> None: ...
    def preferredSize(self) -> QtCore.QSizeF: ...
    @typing.overload
    def setPreferredSize(self, size: QtCore.QSizeF) -> None: ...
    @typing.overload
    def setPreferredSize(self, aw: float, ah: float) -> None: ...
    def setMinimumHeight(self, height: float) -> None: ...
    def setMinimumWidth(self, width: float) -> None: ...
    def minimumSize(self) -> QtCore.QSizeF: ...
    @typing.overload
    def setMinimumSize(self, size: QtCore.QSizeF) -> None: ...
    @typing.overload
    def setMinimumSize(self, aw: float, ah: float) -> None: ...
    def sizePolicy(self) -> 'QSizePolicy': ...
    @typing.overload
    def setSizePolicy(self, policy: 'QSizePolicy') -> None: ...
    @typing.overload
    def setSizePolicy(self, hPolicy: 'QSizePolicy.Policy', vPolicy: 'QSizePolicy.Policy', controlType: 'QSizePolicy.ControlType' = ...)
-> None: ...


class QGraphicsLayout(QGraphicsLayoutItem):

    def __init__(self, parent: typing.Optional[QGraphicsLayoutItem] = ...) -> None: ...

    def addChildLayoutItem(self, layoutItem: typing.Optional[QGraphicsLayoutItem]) -> None: ...
    def updateGeometry(self) -> None: ...
    def removeAt(self, index: int) -> None: ...
    def itemAt(self, i: int) -> typing.Optional[QGraphicsLayoutItem]: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def widgetEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def invalidate(self) -> None: ...
    def isActivated(self) -> bool: ...
    def activate(self) -> None: ...
    def getContentsMargins(self) -> typing.Tuple[typing.Optional[float], typing.Optional[float], typing.Optional[float],
typing.Optional[float]]: ...
    def setContentsMargins(self, left: float, top: float, right: float, bottom: float) -> None: ...


class QGraphicsAnchorLayout(QGraphicsLayout):

    def __init__(self, parent: typing.Optional[QGraphicsLayoutItem] = ...) -> None: ...

    def sizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def invalidate(self) -> None: ...
    def itemAt(self, index: int) -> typing.Optional[QGraphicsLayoutItem]: ...
    def count(self) -> int: ...
    def setGeometry(self, rect: QtCore.QRectF) -> None: ...
    def removeAt(self, index: int) -> None: ...
    def verticalSpacing(self) -> float: ...
    def horizontalSpacing(self) -> float: ...
    def setSpacing(self, spacing: float) -> None: ...
    def setVerticalSpacing(self, spacing: float) -> None: ...
    def setHorizontalSpacing(self, spacing: float) -> None: ...
    def addAnchors(self, firstItem: typing.Optional[QGraphicsLayoutItem], secondItem:
typing.Optional[QGraphicsLayoutItem], orientations: typing.Union[QtCore.Qt.Orientations, QtCore.Qt.Orientation] = ...) ->
None: ...
    def addCornerAnchors(self, firstItem: typing.Optional[QGraphicsLayoutItem], firstCorner: QtCore.Qt.Corner, secondItem:
typing.Optional[QGraphicsLayoutItem], secondCorner: QtCore.Qt.Corner) -> None: ...
    def anchor(self, firstItem: typing.Optional[QGraphicsLayoutItem], firstEdge: QtCore.Qt.AnchorPoint, secondItem:
typing.Optional[QGraphicsLayoutItem], secondEdge: QtCore.Qt.AnchorPoint) -> typing.Optional[QGraphicsAnchor]: ...
    def addAnchor(self, firstItem: typing.Optional[QGraphicsLayoutItem], firstEdge: QtCore.Qt.AnchorPoint, secondItem:
typing.Optional[QGraphicsLayoutItem], secondEdge: QtCore.Qt.AnchorPoint) -> typing.Optional[QGraphicsAnchor]: ...


class QGraphicsEffect(QtCore.QObject):

    class PixmapPadMode(int):
        NoPad = ... # type: QGraphicsEffect.PixmapPadMode
        PadToTransparentBorder = ... # type: QGraphicsEffect.PixmapPadMode
        PadToEffectiveBoundingRect = ... # type: QGraphicsEffect.PixmapPadMode

    class ChangeFlag(int):
        SourceAttached = ... # type: QGraphicsEffect.ChangeFlag
        SourceDetached = ... # type: QGraphicsEffect.ChangeFlag
        SourceBoundingRectChanged = ... # type: QGraphicsEffect.ChangeFlag
        SourceInvalidated = ... # type: QGraphicsEffect.ChangeFlag

    class ChangeFlags(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) ->
'QGraphicsEffect.ChangeFlags': ...
        def __xor__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) ->
'QGraphicsEffect.ChangeFlags': ...
        def __ior__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) ->
'QGraphicsEffect.ChangeFlags': ...
        def __or__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) ->
'QGraphicsEffect.ChangeFlags': ...
```

```
        def __iand__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) ->
'QGraphicsEffect.ChangeFlags': ...
        def __and__(self, f: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) ->
'QGraphicsEffect.ChangeFlags': ...
        def __invert__(self) -> 'QGraphicsEffect.ChangeFlags': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def sourcePixmap(self, system: QtCore.Qt.CoordinateSystem = ..., mode: 'QGraphicsEffect.PixmapPadMode' = ...) ->
typing.Tuple[QtGui.QPixmap, typing.Optional[QtCore.QPoint]]: ...
    def drawSource(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    def sourceBoundingRect(self, system: QtCore.Qt.CoordinateSystem = ...) -> QtCore.QRectF: ...
    def sourceIsPixmap(self) -> bool: ...
    def updateBoundingRect(self) -> None: ...
    def sourceChanged(self, flags: typing.Union['QGraphicsEffect.ChangeFlags', 'QGraphicsEffect.ChangeFlag']) -> None: ...
    def draw(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    enabledChanged: typing.ClassVar[QtCore.pyqtSignal]
    def update(self) -> None: ...
    def setEnabled(self, enable: bool) -> None: ...
    def isEnabled(self) -> bool: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def boundingRectFor(self, sourceRect: QtCore.QRectF) -> QtCore.QRectF: ...


class QGraphicsColorizeEffect(QGraphicsEffect):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def draw(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    strengthChanged: typing.ClassVar[QtCore.pyqtSignal]
    colorChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setStrength(self, strength: float) -> None: ...
    def setColor(self, c: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    def strength(self) -> float: ...
    def color(self) -> QtGui.QColor: ...


class QGraphicsBlurEffect(QGraphicsEffect):

    class BlurHint(int):
        PerformanceHint = ... # type: QGraphicsBlurEffect.BlurHint
        QualityHint = ... # type: QGraphicsBlurEffect.BlurHint
        AnimationHint = ... # type: QGraphicsBlurEffect.BlurHint

    class BlurHints(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) ->
'QGraphicsBlurEffect.BlurHints': ...
        def __xor__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) ->
'QGraphicsBlurEffect.BlurHints': ...
        def __ior__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) ->
'QGraphicsBlurEffect.BlurHints': ...
        def __or__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) ->
'QGraphicsBlurEffect.BlurHints': ...
        def __iand__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) ->
'QGraphicsBlurEffect.BlurHints': ...
        def __and__(self, f: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) ->
'QGraphicsBlurEffect.BlurHints': ...
        def __invert__(self) -> 'QGraphicsBlurEffect.BlurHints': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def draw(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    blurHintsChanged: typing.ClassVar[QtCore.pyqtSignal]
    blurRadiusChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setBlurHints(self, hints: typing.Union['QGraphicsBlurEffect.BlurHints', 'QGraphicsBlurEffect.BlurHint']) -> None: ...
    def setBlurRadius(self, blurRadius: float) -> None: ...
    def blurHints(self) -> 'QGraphicsBlurEffect.BlurHints': ...
    def blurRadius(self) -> float: ...
    def boundingRectFor(self, rect: QtCore.QRectF) -> QtCore.QRectF: ...


class QGraphicsDropShadowEffect(QGraphicsEffect):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def draw(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    colorChanged: typing.ClassVar[QtCore.pyqtSignal]
    blurRadiusChanged: typing.ClassVar[QtCore.pyqtSignal]
    offsetChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setColor(self, color: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    def setBlurRadius(self, blurRadius: float) -> None: ...
    def setYOffset(self, dy: float) -> None: ...
    def setXOffset(self, dx: float) -> None: ...
    @typing.overload
    def setOffset(self, ofs: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    @typing.overload
    def setOffset(self, dx: float, dy: float) -> None: ...
    @typing.overload
    def setOffset(self, d: float) -> None: ...
    def color(self) -> QtGui.QColor: ...
    def blurRadius(self) -> float: ...
    def yOffset(self) -> float: ...
    def xOffset(self) -> float: ...
    def offset(self) -> QtCore.QPointF: ...
    def boundingRectFor(self, rect: QtCore.QRectF) -> QtCore.QRectF: ...


class QGraphicsOpacityEffect(QGraphicsEffect):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def draw(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    opacityMaskChanged: typing.ClassVar[QtCore.pyqtSignal]
    opacityChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setOpacityMask(self, mask: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def setOpacity(self, opacity: float) -> None: ...
    def opacityMask(self) -> QtGui.QBrush: ...
    def opacity(self) -> float: ...


class QGraphicsGridLayout(QGraphicsLayout):

    def __init__(self, parent: typing.Optional[QGraphicsLayoutItem] = ...) -> None: ...

    def removeItem(self, item: typing.Optional[QGraphicsLayoutItem]) -> None: ...
    def sizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def setGeometry(self, rect: QtCore.QRectF) -> None: ...
    def invalidate(self) -> None: ...
    def removeAt(self, index: int) -> None: ...
    def count(self) -> int: ...
    @typing.overload
    def itemAt(self, row: int, column: int) -> typing.Optional[QGraphicsLayoutItem]: ...
    @typing.overload
    def itemAt(self, index: int) -> typing.Optional[QGraphicsLayoutItem]: ...
    def columnCount(self) -> int: ...
    def rowCount(self) -> int: ...
    def alignment(self, item: typing.Optional[QGraphicsLayoutItem]) -> QtCore.Qt.Alignment: ...
    def setAlignment(self, item: typing.Optional[QGraphicsLayoutItem], alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag]) -> None: ...
    def columnAlignment(self, column: int) -> QtCore.Qt.Alignment: ...
    def setColumnAlignment(self, column: int, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) ->
None: ...
    def rowAlignment(self, row: int) -> QtCore.Qt.Alignment: ...
    def setRowAlignment(self, row: int, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def setColumnFixedWidth(self, column: int, width: float) -> None: ...
    def columnMaximumWidth(self, column: int) -> float: ...
    def setColumnMaximumWidth(self, column: int, width: float) -> None: ...
    def columnPreferredWidth(self, column: int) -> float: ...
    def setColumnPreferredWidth(self, column: int, width: float) -> None: ...
    def columnMinimumWidth(self, column: int) -> float: ...
    def setColumnMinimumWidth(self, column: int, width: float) -> None: ...
    def setRowFixedHeight(self, row: int, height: float) -> None: ...
    def rowMaximumHeight(self, row: int) -> float: ...
    def setRowMaximumHeight(self, row: int, height: float) -> None: ...
    def rowPreferredHeight(self, row: int) -> float: ...
    def setRowPreferredHeight(self, row: int, height: float) -> None: ...
    def rowMinimumHeight(self, row: int) -> float: ...
    def setRowMinimumHeight(self, row: int, height: float) -> None: ...
    def columnStretchFactor(self, column: int) -> int: ...
    def setColumnStretchFactor(self, column: int, stretch: int) -> None: ...
    def rowStretchFactor(self, row: int) -> int: ...
    def setRowStretchFactor(self, row: int, stretch: int) -> None: ...
    def columnSpacing(self, column: int) -> float: ...
    def setColumnSpacing(self, column: int, spacing: float) -> None: ...
    def rowSpacing(self, row: int) -> float: ...
    def setRowSpacing(self, row: int, spacing: float) -> None: ...
    def setSpacing(self, spacing: float) -> None: ...
    def verticalSpacing(self) -> float: ...
    def setVerticalSpacing(self, spacing: float) -> None: ...
    def horizontalSpacing(self) -> float: ...
    def setHorizontalSpacing(self, spacing: float) -> None: ...
    @typing.overload
    def addItem(self, item: typing.Optional[QGraphicsLayoutItem], row: int, column: int, rowSpan: int, columnSpan: int,
alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    @typing.overload
    def addItem(self, item: typing.Optional[QGraphicsLayoutItem], row: int, column: int, alignment:
typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...


class QGraphicsItem(PyQt5.sip.wrapper):

    class PanelModality(int):
        NonModal = ... # type: QGraphicsItem.PanelModality
        PanelModal = ... # type: QGraphicsItem.PanelModality
        SceneModal = ... # type: QGraphicsItem.PanelModality

    class GraphicsItemFlag(int):
        ItemIsMovable = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemIsSelectable = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemIsFocusable = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemClipsToShape = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemClipsChildrenToShape = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemIgnoresTransformations = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemIgnoresParentOpacity = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemDoesntPropagateOpacityToChildren = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemStacksBehindParent = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemUsesExtendedStyleOption = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemHasNoContents = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemSendsGeometryChanges = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemAcceptsInputMethod = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemNegativeZStacksBehindParent = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemIsPanel = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemSendsScenePositionChanges = ... # type: QGraphicsItem.GraphicsItemFlag
        ItemContainsChildrenInShape = ... # type: QGraphicsItem.GraphicsItemFlag
    class GraphicsItemChange(int):
        ItemPositionChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemMatrixChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemVisibleChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemEnabledChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemSelectedChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemParentChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemChildAddedChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemChildRemovedChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemTransformChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemPositionHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemTransformHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemSceneChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemVisibleHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemEnabledHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemSelectedHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemParentHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemSceneHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemCursorChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemCursorHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemToolTipChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemToolTipHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemFlagsChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemFlagsHaveChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemZValueChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemZValueHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemOpacityChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemOpacityHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemScenePositionHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemRotationChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemRotationHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemScaleChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemScaleHasChanged = ... # type: QGraphicsItem.GraphicsItemChange
        ItemTransformOriginPointChange = ... # type: QGraphicsItem.GraphicsItemChange
        ItemTransformOriginPointHasChanged = ... # type: QGraphicsItem.GraphicsItemChange

    class CacheMode(int):
        NoCache = ... # type: QGraphicsItem.CacheMode
        ItemCoordinateCache = ... # type: QGraphicsItem.CacheMode
        DeviceCoordinateCache = ... # type: QGraphicsItem.CacheMode

    class GraphicsItemFlags(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) ->
'QGraphicsItem.GraphicsItemFlags': ...
        def __xor__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) ->
'QGraphicsItem.GraphicsItemFlags': ...
        def __ior__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) ->
'QGraphicsItem.GraphicsItemFlags': ...
        def __or__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) ->
'QGraphicsItem.GraphicsItemFlags': ...
        def __iand__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) ->
'QGraphicsItem.GraphicsItemFlags': ...
        def __and__(self, f: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) ->
'QGraphicsItem.GraphicsItemFlags': ...
        def __invert__(self) -> 'QGraphicsItem.GraphicsItemFlags': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    Type = ... # type: int
    UserType = ... # type: int
    def __init__(self, parent: typing.Optional['QGraphicsItem'] = ...) -> None: ...

    def updateMicroFocus(self) -> None: ...
    def setInputMethodHints(self, hints: typing.Union[QtCore.Qt.InputMethodHints, QtCore.Qt.InputMethodHint]) -> None: ...
    def inputMethodHints(self) -> QtCore.Qt.InputMethodHints: ...
    def stackBefore(self, sibling: typing.Optional['QGraphicsItem']) -> None: ...
    @typing.overload
    def setTransformOriginPoint(self, origin: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    @typing.overload
    def setTransformOriginPoint(self, ax: float, ay: float) -> None: ...
    def transformOriginPoint(self) -> QtCore.QPointF: ...
    def setTransformations(self, transformations: typing.Iterable['QGraphicsTransform']) -> None: ...
    def transformations(self) -> typing.List['QGraphicsTransform']: ...
    def scale(self) -> float: ...
    def setScale(self, scale: float) -> None: ...
    def rotation(self) -> float: ...
    def setRotation(self, angle: float) -> None: ...
    def setY(self, y: float) -> None: ...
    def setX(self, x: float) -> None: ...
    def focusItem(self) -> typing.Optional['QGraphicsItem']: ...
    def setFocusProxy(self, item: typing.Optional['QGraphicsItem']) -> None: ...
    def focusProxy(self) -> typing.Optional['QGraphicsItem']: ...
    def setActive(self, active: bool) -> None: ...
    def isActive(self) -> bool: ...
    def setFiltersChildEvents(self, enabled: bool) -> None: ...
    def filtersChildEvents(self) -> bool: ...
    def setAcceptTouchEvents(self, enabled: bool) -> None: ...
    def acceptTouchEvents(self) -> bool: ...
    def setGraphicsEffect(self, effect: typing.Optional[QGraphicsEffect]) -> None: ...
    def graphicsEffect(self) -> typing.Optional[QGraphicsEffect]: ...
    def isBlockedByModalPanel(self) -> typing.Tuple[bool, typing.Optional['QGraphicsItem']]: ...
    def setPanelModality(self, panelModality: 'QGraphicsItem.PanelModality') -> None: ...
    def panelModality(self) -> 'QGraphicsItem.PanelModality': ...
    def toGraphicsObject(self) -> typing.Optional['QGraphicsObject']: ...
    def isPanel(self) -> bool: ...
    def panel(self) -> typing.Optional['QGraphicsItem']: ...
    def parentObject(self) -> typing.Optional['QGraphicsObject']: ...
    @typing.overload
    def mapRectFromScene(self, rect: QtCore.QRectF) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectFromScene(self, ax: float, ay: float, w: float, h: float) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectFromParent(self, rect: QtCore.QRectF) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectFromParent(self, ax: float, ay: float, w: float, h: float) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectFromItem(self, item: typing.Optional['QGraphicsItem'], rect: QtCore.QRectF) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectFromItem(self, item: typing.Optional['QGraphicsItem'], ax: float, ay: float, w: float, h: float) -> 
QtCore.QRectF: ...
    @typing.overload
    def mapRectToScene(self, rect: QtCore.QRectF) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectToScene(self, ax: float, ay: float, w: float, h: float) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectToParent(self, rect: QtCore.QRectF) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectToParent(self, ax: float, ay: float, w: float, h: float) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectToItem(self, item: typing.Optional['QGraphicsItem'], rect: QtCore.QRectF) -> QtCore.QRectF: ...
    @typing.overload
    def mapRectToItem(self, item: typing.Optional['QGraphicsItem'], ax: float, ay: float, w: float, h: float) -> QtCore.QRectF: 
...
    def clipPath(self) -> QtGui.QPainterPath: ...
    def isClipped(self) -> bool: ...
    def itemTransform(self, other: typing.Optional['QGraphicsItem']) -> typing.Tuple[QtGui.QTransform, 
typing.Optional[bool]]: ...
    def setOpacity(self, opacity: float) -> None: ...
    def effectiveOpacity(self) -> float: ...
def opacity(self) -> float: ...
def isUnderMouse(self) -> bool: ...
def commonAncestorItem(self, other: typing.Optional['QGraphicsItem']) -> typing.Optional['QGraphicsItem']: ...
def scroll(self, dx: float, dy: float, rect: QtCore.QRectF = ...) -> None: ...
def setBoundingRegionGranularity(self, granularity: float) -> None: ...
def boundingRegionGranularity(self) -> float: ...
def boundingRegion(self, itemToDeviceTransform: QtGui.QTransform) -> QtGui.QRegion: ...
def ungrabKeyboard(self) -> None: ...
def grabKeyboard(self) -> None: ...
def ungrabMouse(self) -> None: ...
def grabMouse(self) -> None: ...
def setAcceptHoverEvents(self, enabled: bool) -> None: ...
def acceptHoverEvents(self) -> bool: ...
def isVisibleTo(self, parent: typing.Optional['QGraphicsItem']) -> bool: ...
def setCacheMode(self, mode: 'QGraphicsItem.CacheMode', logicalCacheSize: QtCore.QSize = ...) -> None: ...
def cacheMode(self) -> 'QGraphicsItem.CacheMode': ...
def isWindow(self) -> bool: ...
def isWidget(self) -> bool: ...
def childItems(self) -> typing.List['QGraphicsItem']: ...
def window(self) -> typing.Optional['QGraphicsWidget']: ...
def topLevelWidget(self) -> typing.Optional['QGraphicsWidget']: ...
def parentWidget(self) -> typing.Optional['QGraphicsWidget']: ...
@typing.overload
def isObscured(self, rect: QtCore.QRectF = ...) -> bool: ...
@typing.overload
def isObscured(self, ax: float, ay: float, w: float, h: float) -> bool: ...
def resetTransform(self) -> None: ...
def setTransform(self, matrix: QtGui.QTransform, combine: bool = ...) -> None: ...
def deviceTransform(self, viewportTransform: QtGui.QTransform) -> QtGui.QTransform: ...
def sceneTransform(self) -> QtGui.QTransform: ...
def transform(self) -> QtGui.QTransform: ...
def wheelEvent(self, event: typing.Optional['QGraphicsSceneWheelEvent']) -> None: ...
def sceneEventFilter(self, watched: typing.Optional['QGraphicsItem'], event: typing.Optional[QtCore.QEvent]) -> bool: ...
def sceneEvent(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
def prepareGeometryChange(self) -> None: ...
def mouseReleaseEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
def mousePressEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
def mouseMoveEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
def mouseDoubleClickEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
def keyReleaseEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
def itemChange(self, change: 'QGraphicsItem.GraphicsItemChange', value: typing.Any) -> typing.Any: ...
def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
def inputMethodEvent(self, event: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
def hoverMoveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
def hoverLeaveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
def hoverEnterEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
def focusOutEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
def focusInEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
def dropEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
def dragMoveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
def dragLeaveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
def dragEnterEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
def contextMenuEvent(self, event: typing.Optional['QGraphicsSceneContextMenuEvent']) -> None: ...
def removeSceneEventFilter(self, filterItem: typing.Optional['QGraphicsItem']) -> None: ...
def installSceneEventFilter(self, filterItem: typing.Optional['QGraphicsItem']) -> None: ...
def type(self) -> int: ...
def setData(self, key: int, value: typing.Any) -> None: ...
def data(self, key: int) -> typing.Any: ...
def isAncestorOf(self, child: typing.Optional['QGraphicsItem']) -> bool: ...
@typing.overload
def mapFromScene(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.QPointF: ...
@typing.overload
def mapFromScene(self, rect: QtCore.QRectF) -> QtGui.QPolygonF: ...
@typing.overload
def mapFromScene(self, polygon: QtGui.QPolygonF) -> QtGui.QPolygonF: ...
@typing.overload
def mapFromScene(self, path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
@typing.overload
def mapFromScene(self, ax: float, ay: float) -> QtCore.QPointF: ...
```

```
    @typing.overload
    def mapFromScene(self, ax: float, ay: float, w: float, h: float) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapFromParent(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.QPointF: ...
    @typing.overload
    def mapFromParent(self, rect: QtCore.QRectF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapFromParent(self, polygon: QtGui.QPolygonF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapFromParent(self, path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapFromParent(self, ax: float, ay: float) -> QtCore.QPointF: ...
    @typing.overload
    def mapFromParent(self, ax: float, ay: float, w: float, h: float) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapFromItem(self, item: typing.Optional['QGraphicsItem'], point: typing.Union[QtCore.QPointF, QtCore.QPoint]) ->
QtCore.QPointF: ...
    @typing.overload
    def mapFromItem(self, item: typing.Optional['QGraphicsItem'], rect: QtCore.QRectF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapFromItem(self, item: typing.Optional['QGraphicsItem'], polygon: QtGui.QPolygonF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapFromItem(self, item: typing.Optional['QGraphicsItem'], path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapFromItem(self, item: typing.Optional['QGraphicsItem'], ax: float, ay: float) -> QtCore.QPointF: ...
    @typing.overload
    def mapFromItem(self, item: typing.Optional['QGraphicsItem'], ax: float, ay: float, w: float, h: float) -> QtGui.QPolygonF:
...
    @typing.overload
    def mapToScene(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.QPointF: ...
    @typing.overload
    def mapToScene(self, rect: QtCore.QRectF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToScene(self, polygon: QtGui.QPolygonF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToScene(self, path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapToScene(self, ax: float, ay: float) -> QtCore.QPointF: ...
    @typing.overload
    def mapToScene(self, ax: float, ay: float, w: float, h: float) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToParent(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.QPointF: ...
    @typing.overload
    def mapToParent(self, rect: QtCore.QRectF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToParent(self, polygon: QtGui.QPolygonF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToParent(self, path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapToParent(self, ax: float, ay: float) -> QtCore.QPointF: ...
    @typing.overload
    def mapToParent(self, ax: float, ay: float, w: float, h: float) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToItem(self, item: typing.Optional['QGraphicsItem'], point: typing.Union[QtCore.QPointF, QtCore.QPoint]) ->
QtCore.QPointF: ...
    @typing.overload
    def mapToItem(self, item: typing.Optional['QGraphicsItem'], rect: QtCore.QRectF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToItem(self, item: typing.Optional['QGraphicsItem'], polygon: QtGui.QPolygonF) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToItem(self, item: typing.Optional['QGraphicsItem'], path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapToItem(self, item: typing.Optional['QGraphicsItem'], ax: float, ay: float) -> QtCore.QPointF: ...
    @typing.overload
    def mapToItem(self, item: typing.Optional['QGraphicsItem'], ax: float, ay: float, w: float, h: float) -> QtGui.QPolygonF: ...
    @typing.overload
    def update(self, rect: QtCore.QRectF = ...) -> None: ...
    @typing.overload
    def update(self, ax: float, ay: float, width: float, height: float) -> None: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
```

```
typing.Optional[QWidget] = ...) -> None: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional['QGraphicsItem']) -> bool: ...
    def collidingItems(self, mode: QtCore.Qt.ItemSelectionMode = ...) -> typing.List['QGraphicsItem']: ...
    def collidesWithPath(self, path: QtGui.QPainterPath, mode: QtCore.Qt.ItemSelectionMode = ...) -> bool: ...
    def collidesWithItem(self, other: typing.Optional['QGraphicsItem'], mode: QtCore.Qt.ItemSelectionMode = ...) -> bool: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def sceneBoundingRect(self) -> QtCore.QRectF: ...
    def childrenBoundingRect(self) -> QtCore.QRectF: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def setZValue(self, z: float) -> None: ...
    def zValue(self) -> float: ...
    def advance(self, phase: int) -> None: ...
    @typing.overload
    def ensureVisible(self, rect: QtCore.QRectF = ..., xMargin: int = ..., yMargin: int = ...) -> None: ...
    @typing.overload
    def ensureVisible(self, x: float, y: float, w: float, h: float, xMargin: int = ..., yMargin: int = ...) -> None: ...
    def moveBy(self, dx: float, dy: float) -> None: ...
    @typing.overload
    def setPos(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    @typing.overload
    def setPos(self, ax: float, ay: float) -> None: ...
    def scenePos(self) -> QtCore.QPointF: ...
    def y(self) -> float: ...
    def x(self) -> float: ...
    def pos(self) -> QtCore.QPointF: ...
    def clearFocus(self) -> None: ...
    def setFocus(self, focusReason: QtCore.Qt.FocusReason = ...) -> None: ...
    def hasFocus(self) -> bool: ...
    def setAcceptedMouseButtons(self, buttons: typing.Union[QtCore.Qt.MouseButtons, QtCore.Qt.MouseButton]) -> None: ...
    def acceptedMouseButtons(self) -> QtCore.Qt.MouseButtons: ...
    def setAcceptDrops(self, on: bool) -> None: ...
    def acceptDrops(self) -> bool: ...
    def setSelected(self, selected: bool) -> None: ...
    def isSelected(self) -> bool: ...
    def setEnabled(self, enabled: bool) -> None: ...
    def isEnabled(self) -> bool: ...
    def show(self) -> None: ...
    def hide(self) -> None: ...
    def setVisible(self, visible: bool) -> None: ...
    def isVisible(self) -> bool: ...
    def unsetCursor(self) -> None: ...
    def hasCursor(self) -> bool: ...
    def setCursor(self, cursor: typing.Union[QtGui.QCursor, QtCore.Qt.CursorShape]) -> None: ...
    def cursor(self) -> QtGui.QCursor: ...
    def setToolTip(self, toolTip: typing.Optional[str]) -> None: ...
    def toolTip(self) -> str: ...
    def setFlags(self, flags: typing.Union['QGraphicsItem.GraphicsItemFlags', 'QGraphicsItem.GraphicsItemFlag']) -> None: ...
    def setFlag(self, flag: 'QGraphicsItem.GraphicsItemFlag', enabled: bool = ...) -> None: ...
    def flags(self) -> 'QGraphicsItem.GraphicsItemFlags': ...
    def setGroup(self, group: typing.Optional['QGraphicsItemGroup']) -> None: ...
    def group(self) -> typing.Optional['QGraphicsItemGroup']: ...
    def setParentItem(self, parent: typing.Optional['QGraphicsItem']) -> None: ...
    def topLevelItem(self) -> typing.Optional['QGraphicsItem']: ...
    def parentItem(self) -> typing.Optional['QGraphicsItem']: ...
    def scene(self) -> typing.Optional['QGraphicsScene']: ...


class QAbstractGraphicsShapeItem(QGraphicsItem):

    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def setBrush(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def brush(self) -> QtGui.QBrush: ...
    def setPen(self, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]]) -> None: ...
    def pen(self) -> QtGui.QPen: ...
```

```
class QGraphicsPathItem(QAbstractGraphicsShapeItem):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, path: QtGui.QPainterPath, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget] = ...) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def setPath(self, path: QtGui.QPainterPath) -> None: ...
    def path(self) -> QtGui.QPainterPath: ...


class QGraphicsRectItem(QAbstractGraphicsShapeItem):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, rect: QtCore.QRectF, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, x: float, y: float, w: float, h: float, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget] = ...) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    @typing.overload
    def setRect(self, rect: QtCore.QRectF) -> None: ...
    @typing.overload
    def setRect(self, ax: float, ay: float, w: float, h: float) -> None: ...
    def rect(self) -> QtCore.QRectF: ...


class QGraphicsEllipseItem(QAbstractGraphicsShapeItem):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, rect: QtCore.QRectF, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, x: float, y: float, w: float, h: float, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget] = ...) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def setSpanAngle(self, angle: int) -> None: ...
    def spanAngle(self) -> int: ...
    def setStartAngle(self, angle: int) -> None: ...
    def startAngle(self) -> int: ...
    @typing.overload
    def setRect(self, rect: QtCore.QRectF) -> None: ...
    @typing.overload
    def setRect(self, ax: float, ay: float, w: float, h: float) -> None: ...
    def rect(self) -> QtCore.QRectF: ...


class QGraphicsPolygonItem(QAbstractGraphicsShapeItem):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, polygon: QtGui.QPolygonF, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget] = ...) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def setFillRule(self, rule: QtCore.Qt.FillRule) -> None: ...
    def fillRule(self) -> QtCore.Qt.FillRule: ...
    def setPolygon(self, polygon: QtGui.QPolygonF) -> None: ...
    def polygon(self) -> QtGui.QPolygonF: ...


class QGraphicsLineItem(QGraphicsItem):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, line: QtCore.QLineF, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, x1: float, y1: float, x2: float, y2: float, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget] = ...) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    @typing.overload
    def setLine(self, line: QtCore.QLineF) -> None: ...
    @typing.overload
    def setLine(self, x1: float, y1: float, x2: float, y2: float) -> None: ...
    def line(self) -> QtCore.QLineF: ...
    def setPen(self, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]]) -> None: ...
    def pen(self) -> QtGui.QPen: ...


class QGraphicsPixmapItem(QGraphicsItem):

    class ShapeMode(int):
        MaskShape = ... # type: QGraphicsPixmapItem.ShapeMode
        BoundingRectShape = ... # type: QGraphicsPixmapItem.ShapeMode
        HeuristicMaskShape = ... # type: QGraphicsPixmapItem.ShapeMode

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, pixmap: QtGui.QPixmap, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def setShapeMode(self, mode: 'QGraphicsPixmapItem.ShapeMode') -> None: ...
    def shapeMode(self) -> 'QGraphicsPixmapItem.ShapeMode': ...
    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget]) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    @typing.overload
    def setOffset(self, offset: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    @typing.overload
    def setOffset(self, ax: float, ay: float) -> None: ...
    def offset(self) -> QtCore.QPointF: ...
    def setTransformationMode(self, mode: QtCore.Qt.TransformationMode) -> None: ...
    def transformationMode(self) -> QtCore.Qt.TransformationMode: ...
    def setPixmap(self, pixmap: QtGui.QPixmap) -> None: ...
    def pixmap(self) -> QtGui.QPixmap: ...


class QGraphicsSimpleTextItem(QAbstractGraphicsShapeItem):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget: 
typing.Optional[QWidget]) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def font(self) -> QtGui.QFont: ...
    def setFont(self, font: QtGui.QFont) -> None: ...
    def text(self) -> str: ...
    def setText(self, text: typing.Optional[str]) -> None: ...


class QGraphicsItemGroup(QGraphicsItem):

    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget: 
typing.Optional[QWidget] = ...) -> None: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def removeFromGroup(self, item: typing.Optional[QGraphicsItem]) -> None: ...
    def addToGroup(self, item: typing.Optional[QGraphicsItem]) -> None: ...


class QGraphicsObject(QtCore.QObject, QGraphicsItem):

    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def event(self, ev: typing.Optional[QtCore.QEvent]) -> bool: ...
    def updateMicroFocus(self) -> None: ...
    scaleChanged: typing.ClassVar[QtCore.pyqtSignal]
    rotationChanged: typing.ClassVar[QtCore.pyqtSignal]
    zChanged: typing.ClassVar[QtCore.pyqtSignal]
    yChanged: typing.ClassVar[QtCore.pyqtSignal]
    xChanged: typing.ClassVar[QtCore.pyqtSignal]
    enabledChanged: typing.ClassVar[QtCore.pyqtSignal]
    visibleChanged: typing.ClassVar[QtCore.pyqtSignal]
    opacityChanged: typing.ClassVar[QtCore.pyqtSignal]
    parentChanged: typing.ClassVar[QtCore.pyqtSignal]
    def ungrabGesture(self, type: QtCore.Qt.GestureType) -> None: ...
    def grabGesture(self, type: QtCore.Qt.GestureType, flags: typing.Union[QtCore.Qt.GestureFlags, QtCore.Qt.GestureFlag] = 
...) -> None: ...


class QGraphicsTextItem(QGraphicsObject):
    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QGraphicsItem] = ...) -> None: ...

    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    def hoverLeaveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def hoverMoveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def hoverEnterEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def inputMethodEvent(self, event: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def dropEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragMoveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragLeaveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragEnterEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def focusOutEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def keyReleaseEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def contextMenuEvent(self, event: typing.Optional['QGraphicsSceneContextMenuEvent']) -> None: ...
    def mouseDoubleClickEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mouseReleaseEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mouseMoveEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mousePressEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def sceneEvent(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    linkHovered: typing.ClassVar[QtCore.pyqtSignal]
    linkActivated: typing.ClassVar[QtCore.pyqtSignal]
    def textCursor(self) -> QtGui.QTextCursor: ...
    def setTextCursor(self, cursor: QtGui.QTextCursor) -> None: ...
    def openExternalLinks(self) -> bool: ...
    def setOpenExternalLinks(self, open: bool) -> None: ...
    def tabChangesFocus(self) -> bool: ...
    def setTabChangesFocus(self, b: bool) -> None: ...
    def textInteractionFlags(self) -> QtCore.Qt.TextInteractionFlags: ...
    def setTextInteractionFlags(self, flags: typing.Union[QtCore.Qt.TextInteractionFlags, QtCore.Qt.TextInteractionFlag]) ->
None: ...
    def document(self) -> typing.Optional[QtGui.QTextDocument]: ...
    def setDocument(self, document: typing.Optional[QtGui.QTextDocument]) -> None: ...
    def adjustSize(self) -> None: ...
    def textWidth(self) -> float: ...
    def setTextWidth(self, width: float) -> None: ...
    def type(self) -> int: ...
    def opaqueArea(self) -> QtGui.QPainterPath: ...
    def isObscuredBy(self, item: typing.Optional[QGraphicsItem]) -> bool: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget]) -> None: ...
    def contains(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def defaultTextColor(self) -> QtGui.QColor: ...
    def setDefaultTextColor(self, c: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    def setFont(self, font: QtGui.QFont) -> None: ...
    def font(self) -> QtGui.QFont: ...
    def setPlainText(self, text: typing.Optional[str]) -> None: ...
    def toPlainText(self) -> str: ...
    def setHtml(self, html: typing.Optional[str]) -> None: ...
    def toHtml(self) -> str: ...


class QGraphicsLinearLayout(QGraphicsLayout):

    @typing.overload
    def __init__(self, parent: typing.Optional[QGraphicsLayoutItem] = ...) -> None: ...
    @typing.overload
    def __init__(self, orientation: QtCore.Qt.Orientation, parent: typing.Optional[QGraphicsLayoutItem] = ...) -> None: ...

    def sizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def invalidate(self) -> None: ...
    def itemAt(self, index: int) -> typing.Optional[QGraphicsLayoutItem]: ...
    def count(self) -> int: ...
    def setGeometry(self, rect: QtCore.QRectF) -> None: ...
    def alignment(self, item: typing.Optional[QGraphicsLayoutItem]) -> QtCore.Qt.Alignment: ...
    def setAlignment(self, item: typing.Optional[QGraphicsLayoutItem], alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag]) -> None: ...
    def stretchFactor(self, item: typing.Optional[QGraphicsLayoutItem]) -> int: ...
    def setStretchFactor(self, item: typing.Optional[QGraphicsLayoutItem], stretch: int) -> None: ...
    def itemSpacing(self, index: int) -> float: ...
    def setItemSpacing(self, index: int, spacing: float) -> None: ...
    def spacing(self) -> float: ...
    def setSpacing(self, spacing: float) -> None: ...
    def removeAt(self, index: int) -> None: ...
    def removeItem(self, item: typing.Optional[QGraphicsLayoutItem]) -> None: ...
    def insertStretch(self, index: int, stretch: int = ...) -> None: ...
    def insertItem(self, index: int, item: typing.Optional[QGraphicsLayoutItem]) -> None: ...
    def addStretch(self, stretch: int = ...) -> None: ...
    def addItem(self, item: typing.Optional[QGraphicsLayoutItem]) -> None: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setOrientation(self, orientation: QtCore.Qt.Orientation) -> None: ...


class QGraphicsWidget(QGraphicsObject, QGraphicsLayoutItem):

    def __init__(self, parent: typing.Optional[QGraphicsItem] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    geometryChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setAutoFillBackground(self, enabled: bool) -> None: ...
    def autoFillBackground(self) -> bool: ...
    def ungrabKeyboardEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def grabKeyboardEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def ungrabMouseEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def grabMouseEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def hoverLeaveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def hoverMoveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def showEvent(self, event: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def resizeEvent(self, event: typing.Optional['QGraphicsSceneResizeEvent']) -> None: ...
    def polishEvent(self) -> None: ...
    def moveEvent(self, event: typing.Optional['QGraphicsSceneMoveEvent']) -> None: ...
    def hideEvent(self, event: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def focusOutEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def focusInEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def closeEvent(self, event: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def changeEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def windowFrameSectionAt(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.Qt.WindowFrameSection:
...
    def windowFrameEvent(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def sceneEvent(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def itemChange(self, change: QGraphicsItem.GraphicsItemChange, value: typing.Any) -> typing.Any: ...
    def updateGeometry(self) -> None: ...
    def sizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def initStyleOption(self, option: typing.Optional['QStyleOption']) -> None: ...
    def close(self) -> bool: ...
    def shape(self) -> QtGui.QPainterPath: ...
    def boundingRect(self) -> QtCore.QRectF: ...
    def paintWindowFrame(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'],
widget: typing.Optional[QWidget] = ...) -> None: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget:
typing.Optional[QWidget] = ...) -> None: ...
    def type(self) -> int: ...
    def testAttribute(self, attribute: QtCore.Qt.WidgetAttribute) -> bool: ...
    def setAttribute(self, attribute: QtCore.Qt.WidgetAttribute, on: bool = ...) -> None: ...
    def actions(self) -> typing.List[QAction]: ...
    def removeAction(self, action: typing.Optional[QAction]) -> None: ...
    def insertActions(self, before: typing.Optional[QAction], actions: typing.Iterable[QAction]) -> None: ...
    def insertAction(self, before: typing.Optional[QAction], action: typing.Optional[QAction]) -> None: ...
    def addActions(self, actions: typing.Iterable[QAction]) -> None: ...
    def addAction(self, action: typing.Optional[QAction]) -> None: ...
    def setShortcutAutoRepeat(self, id: int, enabled: bool = ...) -> None: ...
    def setShortcutEnabled(self, id: int, enabled: bool = ...) -> None: ...
    def releaseShortcut(self, id: int) -> None: ...
    def grabShortcut(self, sequence: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey,
typing.Optional[str], int], context: QtCore.Qt.ShortcutContext = ...) -> int: ...
    def focusWidget(self) -> typing.Optional['QGraphicsWidget']: ...
    @staticmethod
    def setTabOrder(first: typing.Optional['QGraphicsWidget'], second: typing.Optional['QGraphicsWidget']) -> None: ...
    def setFocusPolicy(self, policy: QtCore.Qt.FocusPolicy) -> None: ...
    def focusPolicy(self) -> QtCore.Qt.FocusPolicy: ...
    def windowTitle(self) -> str: ...
    def setWindowTitle(self, title: typing.Optional[str]) -> None: ...
    def isActiveWindow(self) -> bool: ...
    def setWindowFlags(self, wFlags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType]) -> None: ...
    def windowType(self) -> QtCore.Qt.WindowType: ...
    def windowFlags(self) -> QtCore.Qt.WindowFlags: ...
    def windowFrameRect(self) -> QtCore.QRectF: ...
    def windowFrameGeometry(self) -> QtCore.QRectF: ...
    def unsetWindowFrameMargins(self) -> None: ...
    def getWindowFrameMargins(self) -> typing.Tuple[typing.Optional[float], typing.Optional[float], typing.Optional[float],
typing.Optional[float]]: ...
    @typing.overload
    def setWindowFrameMargins(self, margins: QtCore.QMarginsF) -> None: ...
    @typing.overload
    def setWindowFrameMargins(self, left: float, top: float, right: float, bottom: float) -> None: ...
    def getContentsMargins(self) -> typing.Tuple[typing.Optional[float], typing.Optional[float], typing.Optional[float],
typing.Optional[float]]: ...
    @typing.overload
    def setContentsMargins(self, margins: QtCore.QMarginsF) -> None: ...
    @typing.overload
    def setContentsMargins(self, left: float, top: float, right: float, bottom: float) -> None: ...
    def rect(self) -> QtCore.QRectF: ...
    @typing.overload
    def setGeometry(self, rect: QtCore.QRectF) -> None: ...
    @typing.overload
    def setGeometry(self, ax: float, ay: float, aw: float, ah: float) -> None: ...
    def size(self) -> QtCore.QSizeF: ...
    @typing.overload
    def resize(self, size: QtCore.QSizeF) -> None: ...
    @typing.overload
    def resize(self, w: float, h: float) -> None: ...
    def setPalette(self, palette: QtGui.QPalette) -> None: ...
    def palette(self) -> QtGui.QPalette: ...
    def setFont(self, font: QtGui.QFont) -> None: ...
    def font(self) -> QtGui.QFont: ...
    def setStyle(self, style: typing.Optional[QStyle]) -> None: ...
    def style(self) -> typing.Optional[QStyle]: ...
    def unsetLayoutDirection(self) -> None: ...
    def setLayoutDirection(self, direction: QtCore.Qt.LayoutDirection) -> None: ...
    def layoutDirection(self) -> QtCore.Qt.LayoutDirection: ...
    def adjustSize(self) -> None: ...
    def setLayout(self, layout: typing.Optional[QGraphicsLayout]) -> None: ...
    def layout(self) -> typing.Optional[QGraphicsLayout]: ...


class QGraphicsProxyWidget(QGraphicsWidget):

    def __init__(self, parent: typing.Optional[QGraphicsItem] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def inputMethodEvent(self, event: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    def newProxyWidget(self, a0: typing.Optional[QWidget]) -> typing.Optional['QGraphicsProxyWidget']: ...
    def dropEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragMoveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragLeaveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragEnterEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def resizeEvent(self, event: typing.Optional['QGraphicsSceneResizeEvent']) -> None: ...
    def sizeHint(self, which: QtCore.Qt.SizeHint, constraint: QtCore.QSizeF = ...) -> QtCore.QSizeF: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def focusOutEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def keyReleaseEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def wheelEvent(self, event: typing.Optional['QGraphicsSceneWheelEvent']) -> None: ...
    def mouseDoubleClickEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mouseReleaseEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mousePressEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mouseMoveEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def ungrabMouseEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def grabMouseEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def hoverMoveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def hoverLeaveEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def hoverEnterEvent(self, event: typing.Optional['QGraphicsSceneHoverEvent']) -> None: ...
    def contextMenuEvent(self, event: typing.Optional['QGraphicsSceneContextMenuEvent']) -> None: ...
    def hideEvent(self, event: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def showEvent(self, event: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def itemChange(self, change: QGraphicsItem.GraphicsItemChange, value: typing.Any) -> typing.Any: ...
    def createProxyForChildWidget(self, child: typing.Optional[QWidget]) -> typing.Optional['QGraphicsProxyWidget']: ...
    def type(self) -> int: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: typing.Optional['QStyleOptionGraphicsItem'], widget: 
typing.Optional[QWidget]) -> None: ...
    def setGeometry(self, rect: QtCore.QRectF) -> None: ...
    def subWidgetRect(self, widget: typing.Optional[QWidget]) -> QtCore.QRectF: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    def setWidget(self, widget: typing.Optional[QWidget]) -> None: ...


class QGraphicsScene(QtCore.QObject):

    class SceneLayer(int):
        ItemLayer = ... # type: QGraphicsScene.SceneLayer
        BackgroundLayer = ... # type: QGraphicsScene.SceneLayer
        ForegroundLayer = ... # type: QGraphicsScene.SceneLayer
        AllLayers = ... # type: QGraphicsScene.SceneLayer

    class ItemIndexMethod(int):
        BspTreeIndex = ... # type: QGraphicsScene.ItemIndexMethod
        NoIndex = ... # type: QGraphicsScene.ItemIndexMethod

    class SceneLayers(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> 
'QGraphicsScene.SceneLayers': ...
        def __xor__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> 
'QGraphicsScene.SceneLayers': ...
        def __ior__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> 
'QGraphicsScene.SceneLayers': ...
        def __or__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> 
'QGraphicsScene.SceneLayers': ...
        def __iand__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> 
'QGraphicsScene.SceneLayers': ...
        def __and__(self, f: typing.Union['QGraphicsScene.SceneLayers', 'QGraphicsScene.SceneLayer']) -> 
'QGraphicsScene.SceneLayers': ...
        def __invert__(self) -> 'QGraphicsScene.SceneLayers': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...
    @typing.overload
    def __init__(self, sceneRect: QtCore.QRectF, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...
    @typing.overload
    def __init__(self, x: float, y: float, width: float, height: float, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def setFocusOnTouch(self, enabled: bool) -> None: ...
    def focusOnTouch(self) -> bool: ...
    focusItemChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setMinimumRenderSize(self, minSize: float) -> None: ...
    def minimumRenderSize(self) -> float: ...
    def sendEvent(self, item: typing.Optional[QGraphicsItem], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setActivePanel(self, item: typing.Optional[QGraphicsItem]) -> None: ...
    def activePanel(self) -> typing.Optional[QGraphicsItem]: ...
    def isActive(self) -> bool: ...
    @typing.overload
    def itemAt(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint], deviceTransform: QtGui.QTransform) ->
typing.Optional[QGraphicsItem]: ...
    @typing.overload
    def itemAt(self, x: float, y: float, deviceTransform: QtGui.QTransform) -> typing.Optional[QGraphicsItem]: ...
    def stickyFocus(self) -> bool: ...
    def setStickyFocus(self, enabled: bool) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def eventFilter(self, watched: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setActiveWindow(self, widget: typing.Optional[QGraphicsWidget]) -> None: ...
    def activeWindow(self) -> typing.Optional[QGraphicsWidget]: ...
    def setPalette(self, palette: QtGui.QPalette) -> None: ...
    def palette(self) -> QtGui.QPalette: ...
    def setFont(self, font: QtGui.QFont) -> None: ...
    def font(self) -> QtGui.QFont: ...
    def setStyle(self, style: typing.Optional[QStyle]) -> None: ...
    def style(self) -> typing.Optional[QStyle]: ...
    def addWidget(self, widget: typing.Optional[QWidget], flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> typing.Optional[QGraphicsProxyWidget]: ...
    def selectionArea(self) -> QtGui.QPainterPath: ...
    def setBspTreeDepth(self, depth: int) -> None: ...
    def bspTreeDepth(self) -> int: ...
    def drawForeground(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRectF) -> None: ...
    def drawBackground(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRectF) -> None: ...
    def inputMethodEvent(self, event: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def wheelEvent(self, event: typing.Optional['QGraphicsSceneWheelEvent']) -> None: ...
    def mouseDoubleClickEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mouseReleaseEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mouseMoveEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def mousePressEvent(self, event: typing.Optional['QGraphicsSceneMouseEvent']) -> None: ...
    def keyReleaseEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def helpEvent(self, event: typing.Optional['QGraphicsSceneHelpEvent']) -> None: ...
    def focusOutEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def dropEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragLeaveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragMoveEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def dragEnterEvent(self, event: typing.Optional['QGraphicsSceneDragDropEvent']) -> None: ...
    def contextMenuEvent(self, event: typing.Optional['QGraphicsSceneContextMenuEvent']) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    selectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    sceneRectChanged: typing.ClassVar[QtCore.pyqtSignal]
    changed: typing.ClassVar[QtCore.pyqtSignal]
    def clear(self) -> None: ...
    @typing.overload
    def invalidate(self, rect: QtCore.QRectF = ..., layers: typing.Union['QGraphicsScene.SceneLayers',
'QGraphicsScene.SceneLayer'] = ...) -> None: ...
    @typing.overload
    def invalidate(self, x: float, y: float, w: float, h: float, layers: typing.Union['QGraphicsScene.SceneLayers',
'QGraphicsScene.SceneLayer'] = ...) -> None: ...
    @typing.overload
    def update(self, rect: QtCore.QRectF = ...) -> None: ...
    @typing.overload
    def update(self, x: float, y: float, w: float, h: float) -> None: ...
    def advance(self) -> None: ...
    def views(self) -> typing.List['QGraphicsView']: ...
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    def setForegroundBrush(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def foregroundBrush(self) -> QtGui.QBrush: ...
    def setBackgroundBrush(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def backgroundBrush(self) -> QtGui.QBrush: ...
    def mouseGrabberItem(self) -> typing.Optional[QGraphicsItem]: ...
    def clearFocus(self) -> None: ...
    def setFocus(self, focusReason: QtCore.Qt.FocusReason = ...) -> None: ...
    def hasFocus(self) -> bool: ...
    def setFocusItem(self, item: typing.Optional[QGraphicsItem], focusReason: QtCore.Qt.FocusReason = ...) -> None: ...
    def focusItem(self) -> typing.Optional[QGraphicsItem]: ...
    def removeItem(self, item: typing.Optional[QGraphicsItem]) -> None: ...
    def addText(self, text: typing.Optional[str], font: QtGui.QFont = ...) -> typing.Optional[QGraphicsTextItem]: ...
    def addSimpleText(self, text: typing.Optional[str], font: QtGui.QFont = ...) -> typing.Optional[QGraphicsSimpleTextItem]:
...
    @typing.overload
    def addRect(self, rect: QtCore.QRectF, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]]
= ..., brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], QtGui.QGradient] = ...) ->
typing.Optional[QGraphicsRectItem]: ...
    @typing.overload
    def addRect(self, x: float, y: float, w: float, h: float, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor]] = ..., brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient] = ...) -> typing.Optional[QGraphicsRectItem]: ...
    def addPolygon(self, polygon: QtGui.QPolygonF, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor]] = ..., brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient] = ...) -> typing.Optional[QGraphicsPolygonItem]: ...
    def addPixmap(self, pixmap: QtGui.QPixmap) -> typing.Optional[QGraphicsPixmapItem]: ...
    def addPath(self, path: QtGui.QPainterPath, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor]] = ..., brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient] = ...) -> typing.Optional[QGraphicsPathItem]: ...
    @typing.overload
    def addLine(self, line: QtCore.QLineF, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]]
= ...) -> typing.Optional[QGraphicsLineItem]: ...
    @typing.overload
    def addLine(self, x1: float, y1: float, x2: float, y2: float, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor]] = ...) -> typing.Optional[QGraphicsLineItem]: ...
    @typing.overload
    def addEllipse(self, rect: QtCore.QRectF, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor]] = ..., brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient] = ...) -> typing.Optional[QGraphicsEllipseItem]: ...
    @typing.overload
    def addEllipse(self, x: float, y: float, w: float, h: float, pen: typing.Union[QtGui.QPen, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor]] = ..., brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient] = ...) -> typing.Optional[QGraphicsEllipseItem]: ...
    def addItem(self, item: typing.Optional[QGraphicsItem]) -> None: ...
    def destroyItemGroup(self, group: typing.Optional[QGraphicsItemGroup]) -> None: ...
    def createItemGroup(self, items: typing.Iterable[QGraphicsItem]) -> typing.Optional[QGraphicsItemGroup]: ...
    def clearSelection(self) -> None: ...
    @typing.overload
    def setSelectionArea(self, path: QtGui.QPainterPath, deviceTransform: QtGui.QTransform) -> None: ...
    @typing.overload
    def setSelectionArea(self, path: QtGui.QPainterPath, mode: QtCore.Qt.ItemSelectionMode = ..., deviceTransform:
QtGui.QTransform = ...) -> None: ...
    @typing.overload
    def setSelectionArea(self, path: QtGui.QPainterPath, selectionOperation: QtCore.Qt.ItemSelectionOperation, mode:
QtCore.Qt.ItemSelectionMode = ..., deviceTransform: QtGui.QTransform = ...) -> None: ...
    def selectedItems(self) -> typing.List[QGraphicsItem]: ...
    def collidingItems(self, item: typing.Optional[QGraphicsItem], mode: QtCore.Qt.ItemSelectionMode = ...) ->
typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, order: QtCore.Qt.SortOrder = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint], mode: QtCore.Qt.ItemSelectionMode = ..., order:
QtCore.Qt.SortOrder = ..., deviceTransform: QtGui.QTransform = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, rect: QtCore.QRectF, mode: QtCore.Qt.ItemSelectionMode = ..., order: QtCore.Qt.SortOrder = ...,
deviceTransform: QtGui.QTransform = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, polygon: QtGui.QPolygonF, mode: QtCore.Qt.ItemSelectionMode = ..., order: QtCore.Qt.SortOrder = ...,
deviceTransform: QtGui.QTransform = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, path: QtGui.QPainterPath, mode: QtCore.Qt.ItemSelectionMode = ..., order: QtCore.Qt.SortOrder = ...,
deviceTransform: QtGui.QTransform = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, x: float, y: float, w: float, h: float, mode: QtCore.Qt.ItemSelectionMode, order: QtCore.Qt.SortOrder,
deviceTransform: QtGui.QTransform = ...) -> typing.List[QGraphicsItem]: ...
    def itemsBoundingRect(self) -> QtCore.QRectF: ...
    def setItemIndexMethod(self, method: 'QGraphicsScene.ItemIndexMethod') -> None: ...
    def itemIndexMethod(self) -> 'QGraphicsScene.ItemIndexMethod': ...
    def render(self, painter: typing.Optional[QtGui.QPainter], target: QtCore.QRectF = ..., source: QtCore.QRectF = ..., mode:
QtCore.Qt.AspectRatioMode = ...) -> None: ...
    @typing.overload
    def setSceneRect(self, rect: QtCore.QRectF) -> None: ...
    @typing.overload
    def setSceneRect(self, x: float, y: float, w: float, h: float) -> None: ...
    def height(self) -> float: ...
    def width(self) -> float: ...
    def sceneRect(self) -> QtCore.QRectF: ...


class QGraphicsSceneEvent(QtCore.QEvent):

    def widget(self) -> typing.Optional[QWidget]: ...


class QGraphicsSceneMouseEvent(QGraphicsSceneEvent):

    def flags(self) -> QtCore.Qt.MouseEventFlags: ...
    def source(self) -> QtCore.Qt.MouseEventSource: ...
    def modifiers(self) -> QtCore.Qt.KeyboardModifiers: ...
    def button(self) -> QtCore.Qt.MouseButton: ...
    def buttons(self) -> QtCore.Qt.MouseButtons: ...
    def lastScreenPos(self) -> QtCore.QPoint: ...
    def lastScenePos(self) -> QtCore.QPointF: ...
    def lastPos(self) -> QtCore.QPointF: ...
    def buttonDownScreenPos(self, button: QtCore.Qt.MouseButton) -> QtCore.QPoint: ...
    def buttonDownScenePos(self, button: QtCore.Qt.MouseButton) -> QtCore.QPointF: ...
    def buttonDownPos(self, button: QtCore.Qt.MouseButton) -> QtCore.QPointF: ...
    def screenPos(self) -> QtCore.QPoint: ...
    def scenePos(self) -> QtCore.QPointF: ...
    def pos(self) -> QtCore.QPointF: ...


class QGraphicsSceneWheelEvent(QGraphicsSceneEvent):

    def orientation(self) -> QtCore.Qt.Orientation: ...
    def delta(self) -> int: ...
    def modifiers(self) -> QtCore.Qt.KeyboardModifiers: ...
    def buttons(self) -> QtCore.Qt.MouseButtons: ...
    def screenPos(self) -> QtCore.QPoint: ...
    def scenePos(self) -> QtCore.QPointF: ...
    def pos(self) -> QtCore.QPointF: ...


class QGraphicsSceneContextMenuEvent(QGraphicsSceneEvent):

    class Reason(int):
        Mouse = ... # type: QGraphicsSceneContextMenuEvent.Reason
        Keyboard = ... # type: QGraphicsSceneContextMenuEvent.Reason
        Other = ... # type: QGraphicsSceneContextMenuEvent.Reason

    def reason(self) -> 'QGraphicsSceneContextMenuEvent.Reason': ...
    def modifiers(self) -> QtCore.Qt.KeyboardModifiers: ...
    def screenPos(self) -> QtCore.QPoint: ...
    def scenePos(self) -> QtCore.QPointF: ...
    def pos(self) -> QtCore.QPointF: ...
```

```
class QGraphicsSceneHoverEvent(QGraphicsSceneEvent):

    def modifiers(self) -> QtCore.Qt.KeyboardModifiers: ...
    def lastScreenPos(self) -> QtCore.QPoint: ...
    def lastScenePos(self) -> QtCore.QPointF: ...
    def lastPos(self) -> QtCore.QPointF: ...
    def screenPos(self) -> QtCore.QPoint: ...
    def scenePos(self) -> QtCore.QPointF: ...
    def pos(self) -> QtCore.QPointF: ...


class QGraphicsSceneHelpEvent(QGraphicsSceneEvent):

    def screenPos(self) -> QtCore.QPoint: ...
    def scenePos(self) -> QtCore.QPointF: ...


class QGraphicsSceneDragDropEvent(QGraphicsSceneEvent):

    def mimeData(self) -> typing.Optional[QtCore.QMimeData]: ...
    def source(self) -> typing.Optional[QWidget]: ...
    def setDropAction(self, action: QtCore.Qt.DropAction) -> None: ...
    def dropAction(self) -> QtCore.Qt.DropAction: ...
    def acceptProposedAction(self) -> None: ...
    def proposedAction(self) -> QtCore.Qt.DropAction: ...
    def possibleActions(self) -> QtCore.Qt.DropActions: ...
    def modifiers(self) -> QtCore.Qt.KeyboardModifiers: ...
    def buttons(self) -> QtCore.Qt.MouseButtons: ...
    def screenPos(self) -> QtCore.QPoint: ...
    def scenePos(self) -> QtCore.QPointF: ...
    def pos(self) -> QtCore.QPointF: ...


class QGraphicsSceneResizeEvent(QGraphicsSceneEvent):

    def __init__(self) -> None: ...

    def newSize(self) -> QtCore.QSizeF: ...
    def oldSize(self) -> QtCore.QSizeF: ...


class QGraphicsSceneMoveEvent(QGraphicsSceneEvent):

    def __init__(self) -> None: ...

    def newPos(self) -> QtCore.QPointF: ...
    def oldPos(self) -> QtCore.QPointF: ...


class QGraphicsTransform(QtCore.QObject):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def update(self) -> None: ...
    def applyTo(self, matrix: typing.Optional[QtGui.QMatrix4x4]) -> None: ...


class QGraphicsScale(QGraphicsTransform):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    zScaleChanged: typing.ClassVar[QtCore.pyqtSignal]
    yScaleChanged: typing.ClassVar[QtCore.pyqtSignal]
    xScaleChanged: typing.ClassVar[QtCore.pyqtSignal]
    scaleChanged: typing.ClassVar[QtCore.pyqtSignal]
    originChanged: typing.ClassVar[QtCore.pyqtSignal]
    def applyTo(self, matrix: typing.Optional[QtGui.QMatrix4x4]) -> None: ...
    def setZScale(self, a0: float) -> None: ...
    def zScale(self) -> float: ...
    def setYScale(self, a0: float) -> None: ...
    def yScale(self) -> float: ...
    def setXScale(self, a0: float) -> None: ...
    def xScale(self) -> float: ...
    def setOrigin(self, point: QtGui.QVector3D) -> None: ...
    def origin(self) -> QtGui.QVector3D: ...


class QGraphicsRotation(QGraphicsTransform):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    axisChanged: typing.ClassVar[QtCore.pyqtSignal]
    angleChanged: typing.ClassVar[QtCore.pyqtSignal]
    originChanged: typing.ClassVar[QtCore.pyqtSignal]
    def applyTo(self, matrix: typing.Optional[QtGui.QMatrix4x4]) -> None: ...
    @typing.overload
    def setAxis(self, axis: QtGui.QVector3D) -> None: ...
    @typing.overload
    def setAxis(self, axis: QtCore.Qt.Axis) -> None: ...
    def axis(self) -> QtGui.QVector3D: ...
    def setAngle(self, a0: float) -> None: ...
    def angle(self) -> float: ...
    def setOrigin(self, point: QtGui.QVector3D) -> None: ...
    def origin(self) -> QtGui.QVector3D: ...


class QGraphicsView(QAbstractScrollArea):

    class OptimizationFlag(int):
        DontClipPainter = ... # type: QGraphicsView.OptimizationFlag
        DontSavePainterState = ... # type: QGraphicsView.OptimizationFlag
        DontAdjustForAntialiasing = ... # type: QGraphicsView.OptimizationFlag

    class ViewportUpdateMode(int):
        FullViewportUpdate = ... # type: QGraphicsView.ViewportUpdateMode
        MinimalViewportUpdate = ... # type: QGraphicsView.ViewportUpdateMode
        SmartViewportUpdate = ... # type: QGraphicsView.ViewportUpdateMode
        BoundingRectViewportUpdate = ... # type: QGraphicsView.ViewportUpdateMode
        NoViewportUpdate = ... # type: QGraphicsView.ViewportUpdateMode

    class ViewportAnchor(int):
        NoAnchor = ... # type: QGraphicsView.ViewportAnchor
        AnchorViewCenter = ... # type: QGraphicsView.ViewportAnchor
        AnchorUnderMouse = ... # type: QGraphicsView.ViewportAnchor

    class DragMode(int):
        NoDrag = ... # type: QGraphicsView.DragMode
        ScrollHandDrag = ... # type: QGraphicsView.DragMode
        RubberBandDrag = ... # type: QGraphicsView.DragMode

    class CacheModeFlag(int):
        CacheNone = ... # type: QGraphicsView.CacheModeFlag
        CacheBackground = ... # type: QGraphicsView.CacheModeFlag

    class CacheMode(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) -> 
'QGraphicsView.CacheMode': ...
        def __xor__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) -> 
'QGraphicsView.CacheMode': ...
        def __ior__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) ->
```

```
'QGraphicsView.CacheMode': ...
        def __or__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) ->
'QGraphicsView.CacheMode': ...
        def __iand__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) ->
'QGraphicsView.CacheMode': ...
        def __and__(self, f: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) ->
'QGraphicsView.CacheMode': ...
        def __invert__(self) -> 'QGraphicsView.CacheMode': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    class OptimizationFlags(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) ->
'QGraphicsView.OptimizationFlags': ...
        def __xor__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) ->
'QGraphicsView.OptimizationFlags': ...
        def __ior__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) ->
'QGraphicsView.OptimizationFlags': ...
        def __or__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) ->
'QGraphicsView.OptimizationFlags': ...
        def __iand__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) ->
'QGraphicsView.OptimizationFlags': ...
        def __and__(self, f: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) ->
'QGraphicsView.OptimizationFlags': ...
        def __invert__(self) -> 'QGraphicsView.OptimizationFlags': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, scene: typing.Optional[QGraphicsScene], parent: typing.Optional[QWidget] = ...) -> None: ...

    rubberBandChanged: typing.ClassVar[QtCore.pyqtSignal]
    def rubberBandRect(self) -> QtCore.QRect: ...
    def isTransformed(self) -> bool: ...
    def resetTransform(self) -> None: ...
    def setTransform(self, matrix: QtGui.QTransform, combine: bool = ...) -> None: ...
    def viewportTransform(self) -> QtGui.QTransform: ...
    def transform(self) -> QtGui.QTransform: ...
    def setRubberBandSelectionMode(self, mode: QtCore.Qt.ItemSelectionMode) -> None: ...
    def rubberBandSelectionMode(self) -> QtCore.Qt.ItemSelectionMode: ...
    def setOptimizationFlags(self, flags: typing.Union['QGraphicsView.OptimizationFlags', 'QGraphicsView.OptimizationFlag']) -
> None: ...
    def setOptimizationFlag(self, flag: 'QGraphicsView.OptimizationFlag', enabled: bool = ...) -> None: ...
    def optimizationFlags(self) -> 'QGraphicsView.OptimizationFlags': ...
    def setViewportUpdateMode(self, mode: 'QGraphicsView.ViewportUpdateMode') -> None: ...
    def viewportUpdateMode(self) -> 'QGraphicsView.ViewportUpdateMode': ...
    def drawForeground(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRectF) -> None: ...
    def drawBackground(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRectF) -> None: ...
    def inputMethodEvent(self, event: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def showEvent(self, event: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def resizeEvent(self, event: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def paintEvent(self, event: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def wheelEvent(self, event: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def mouseReleaseEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseDoubleClickEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def keyReleaseEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def focusOutEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, event: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def dropEvent(self, event: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragMoveEvent(self, event: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def dragLeaveEvent(self, event: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragEnterEvent(self, event: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
    def contextMenuEvent(self, event: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def viewportEvent(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setupViewport(self, widget: typing.Optional[QWidget]) -> None: ...
    def updateSceneRect(self, rect: QtCore.QRectF) -> None: ...
    def updateScene(self, rects: typing.Iterable[QtCore.QRectF]) -> None: ...
    def invalidateScene(self, rect: QtCore.QRectF = ..., layers: typing.Union[QGraphicsScene.SceneLayers,
QGraphicsScene.SceneLayer] = ...) -> None: ...
    def setForegroundBrush(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def foregroundBrush(self) -> QtGui.QBrush: ...
    def setBackgroundBrush(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def backgroundBrush(self) -> QtGui.QBrush: ...
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    @typing.overload
    def mapFromScene(self, point: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> QtCore.QPoint: ...
    @typing.overload
    def mapFromScene(self, rect: QtCore.QRectF) -> QtGui.QPolygon: ...
    @typing.overload
    def mapFromScene(self, polygon: QtGui.QPolygonF) -> QtGui.QPolygon: ...
    @typing.overload
    def mapFromScene(self, path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapFromScene(self, ax: float, ay: float) -> QtCore.QPoint: ...
    @typing.overload
    def mapFromScene(self, ax: float, ay: float, w: float, h: float) -> QtGui.QPolygon: ...
    @typing.overload
    def mapToScene(self, point: QtCore.QPoint) -> QtCore.QPointF: ...
    @typing.overload
    def mapToScene(self, rect: QtCore.QRect) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToScene(self, polygon: QtGui.QPolygon) -> QtGui.QPolygonF: ...
    @typing.overload
    def mapToScene(self, path: QtGui.QPainterPath) -> QtGui.QPainterPath: ...
    @typing.overload
    def mapToScene(self, ax: int, ay: int) -> QtCore.QPointF: ...
    @typing.overload
    def mapToScene(self, ax: int, ay: int, w: int, h: int) -> QtGui.QPolygonF: ...
    @typing.overload
    def itemAt(self, pos: QtCore.QPoint) -> typing.Optional[QGraphicsItem]: ...
    @typing.overload
    def itemAt(self, ax: int, ay: int) -> typing.Optional[QGraphicsItem]: ...
    @typing.overload
    def items(self) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, pos: QtCore.QPoint) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, x: int, y: int) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, x: int, y: int, w: int, h: int, mode: QtCore.Qt.ItemSelectionMode = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, rect: QtCore.QRect, mode: QtCore.Qt.ItemSelectionMode = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, polygon: QtGui.QPolygon, mode: QtCore.Qt.ItemSelectionMode = ...) -> typing.List[QGraphicsItem]: ...
    @typing.overload
    def items(self, path: QtGui.QPainterPath, mode: QtCore.Qt.ItemSelectionMode = ...) -> typing.List[QGraphicsItem]: ...
    def render(self, painter: typing.Optional[QtGui.QPainter], target: QtCore.QRectF = ..., source: QtCore.QRect = ..., mode:
QtCore.Qt.AspectRatioMode = ...) -> None: ...
    @typing.overload
    def fitInView(self, rect: QtCore.QRectF, mode: QtCore.Qt.AspectRatioMode = ...) -> None: ...
    @typing.overload
    def fitInView(self, item: typing.Optional[QGraphicsItem], mode: QtCore.Qt.AspectRatioMode = ...) -> None: ...
    @typing.overload
    def fitInView(self, x: float, y: float, w: float, h: float, mode: QtCore.Qt.AspectRatioMode = ...) -> None: ...
    @typing.overload
    def ensureVisible(self, rect: QtCore.QRectF, xMargin: int = ..., yMargin: int = ...) -> None: ...
    @typing.overload
    def ensureVisible(self, item: typing.Optional[QGraphicsItem], xMargin: int = ..., yMargin: int = ...) -> None: ...
    @typing.overload
    def ensureVisible(self, x: float, y: float, w: float, h: float, xMargin: int = ..., yMargin: int = ...) -> None: ...
    @typing.overload
    def centerOn(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    @typing.overload
    def centerOn(self, item: typing.Optional[QGraphicsItem]) -> None: ...
    @typing.overload
    def centerOn(self, ax: float, ay: float) -> None: ...
    def translate(self, dx: float, dy: float) -> None: ...
    def shear(self, sh: float, sv: float) -> None: ...
    def scale(self, sx: float, sy: float) -> None: ...
    def rotate(self, angle: float) -> None: ...
    @typing.overload
    def setSceneRect(self, rect: QtCore.QRectF) -> None: ...
    @typing.overload
    def setSceneRect(self, ax: float, ay: float, aw: float, ah: float) -> None: ...
    def sceneRect(self) -> QtCore.QRectF: ...
    def setScene(self, scene: typing.Optional[QGraphicsScene]) -> None: ...
    def scene(self) -> typing.Optional[QGraphicsScene]: ...
    def setInteractive(self, allowed: bool) -> None: ...
    def isInteractive(self) -> bool: ...
    def resetCachedContent(self) -> None: ...
    def setCacheMode(self, mode: typing.Union['QGraphicsView.CacheMode', 'QGraphicsView.CacheModeFlag']) -> None: ...
    def cacheMode(self) -> 'QGraphicsView.CacheMode': ...
    def setDragMode(self, mode: 'QGraphicsView.DragMode') -> None: ...
    def dragMode(self) -> 'QGraphicsView.DragMode': ...
    def setResizeAnchor(self, anchor: 'QGraphicsView.ViewportAnchor') -> None: ...
    def resizeAnchor(self) -> 'QGraphicsView.ViewportAnchor': ...
    def setTransformationAnchor(self, anchor: 'QGraphicsView.ViewportAnchor') -> None: ...
    def transformationAnchor(self) -> 'QGraphicsView.ViewportAnchor': ...
    def setAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def setRenderHints(self, hints: typing.Union[QtGui.QPainter.RenderHints, QtGui.QPainter.RenderHint]) -> None: ...
    def setRenderHint(self, hint: QtGui.QPainter.RenderHint, on: bool = ...) -> None: ...
    def renderHints(self) -> QtGui.QPainter.RenderHints: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QGridLayout(QLayout):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def __init__(self) -> None: ...

    def itemAtPosition(self, row: int, column: int) -> typing.Optional[QLayoutItem]: ...
    def spacing(self) -> int: ...
    def setSpacing(self, spacing: int) -> None: ...
    def verticalSpacing(self) -> int: ...
    def setVerticalSpacing(self, spacing: int) -> None: ...
    def horizontalSpacing(self) -> int: ...
    def setHorizontalSpacing(self, spacing: int) -> None: ...
    def getItemPosition(self, idx: int) -> typing.Tuple[typing.Optional[int], typing.Optional[int], typing.Optional[int],
typing.Optional[int]]: ...
    def setDefaultPositioning(self, n: int, orient: QtCore.Qt.Orientation) -> None: ...
    @typing.overload
    def addItem(self, item: typing.Optional[QLayoutItem], row: int, column: int, rowSpan: int = ..., columnSpan: int = ...,
alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    @typing.overload
    def addItem(self, a0: typing.Optional[QLayoutItem]) -> None: ...
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    def count(self) -> int: ...
    def takeAt(self, a0: int) -> typing.Optional[QLayoutItem]: ...
    def itemAt(self, a0: int) -> typing.Optional[QLayoutItem]: ...
    def originCorner(self) -> QtCore.Qt.Corner: ...
    def setOriginCorner(self, a0: QtCore.Qt.Corner) -> None: ...
    @typing.overload
    def addLayout(self, a0: typing.Optional[QLayout], row: int, column: int, alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    @typing.overload
    def addLayout(self, a0: typing.Optional[QLayout], row: int, column: int, rowSpan: int, columnSpan: int, alignment:
typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    @typing.overload
    def addWidget(self, w: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def addWidget(self, a0: typing.Optional[QWidget], row: int, column: int, alignment: typing.Union[QtCore.Qt.Alignment,
QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    @typing.overload
    def addWidget(self, a0: typing.Optional[QWidget], row: int, column: int, rowSpan: int, columnSpan: int, alignment:
typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag] = ...) -> None: ...
    def invalidate(self) -> None: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def minimumHeightForWidth(self, a0: int) -> int: ...
    def heightForWidth(self, a0: int) -> int: ...
    def hasHeightForWidth(self) -> bool: ...
    def cellRect(self, row: int, column: int) -> QtCore.QRect: ...
    def rowCount(self) -> int: ...
    def columnCount(self) -> int: ...
    def columnMinimumWidth(self, column: int) -> int: ...
    def rowMinimumHeight(self, row: int) -> int: ...
    def setColumnMinimumWidth(self, column: int, minSize: int) -> None: ...
    def setRowMinimumHeight(self, row: int, minSize: int) -> None: ...
    def columnStretch(self, column: int) -> int: ...
    def rowStretch(self, row: int) -> int: ...
    def setColumnStretch(self, column: int, stretch: int) -> None: ...
    def setRowStretch(self, row: int, stretch: int) -> None: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QGroupBox(QWidget):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, title: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    def mouseReleaseEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def focusInEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def childEvent(self, a0: typing.Optional[QtCore.QChildEvent]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionGroupBox']) -> None: ...
    toggled: typing.ClassVar[QtCore.pyqtSignal]
    clicked: typing.ClassVar[QtCore.pyqtSignal]
    def setChecked(self, b: bool) -> None: ...
    def isChecked(self) -> bool: ...
    def setCheckable(self, b: bool) -> None: ...
    def isCheckable(self) -> bool: ...
    def setFlat(self, b: bool) -> None: ...
    def isFlat(self) -> bool: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def setAlignment(self, a0: int) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def setTitle(self, a0: typing.Optional[str]) -> None: ...
    def title(self) -> str: ...
```

```
class QHeaderView(QAbstractItemView):

    class ResizeMode(int):
        Interactive = ... # type: QHeaderView.ResizeMode
        Fixed = ... # type: QHeaderView.ResizeMode
        Stretch = ... # type: QHeaderView.ResizeMode
        ResizeToContents = ... # type: QHeaderView.ResizeMode
        Custom = ... # type: QHeaderView.ResizeMode

    def __init__(self, orientation: QtCore.Qt.Orientation, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isFirstSectionMovable(self) -> bool: ...
    def setFirstSectionMovable(self, movable: bool) -> None: ...
    def resetDefaultSectionSize(self) -> None: ...
    def setMaximumSectionSize(self, size: int) -> None: ...
    def maximumSectionSize(self) -> int: ...
    def resizeContentsPrecision(self) -> int: ...
    def setResizeContentsPrecision(self, precision: int) -> None: ...
    def setVisible(self, v: bool) -> None: ...
    @typing.overload
    def setSectionResizeMode(self, logicalIndex: int, mode: 'QHeaderView.ResizeMode') -> None: ...
    @typing.overload
    def setSectionResizeMode(self, mode: 'QHeaderView.ResizeMode') -> None: ...
    def sectionResizeMode(self, logicalIndex: int) -> 'QHeaderView.ResizeMode': ...
    def sectionsClickable(self) -> bool: ...
    def setSectionsClickable(self, clickable: bool) -> None: ...
    def sectionsMovable(self) -> bool: ...
    def setSectionsMovable(self, movable: bool) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionHeader']) -> None: ...
    sortIndicatorChanged: typing.ClassVar[QtCore.pyqtSignal]
    sectionEntered: typing.ClassVar[QtCore.pyqtSignal]
    def setOffsetToLastSection(self) -> None: ...
    def reset(self) -> None: ...
    def restoreState(self, state: typing.Union[QtCore.QByteArray, bytes, bytearray]) -> bool: ...
    def saveState(self) -> QtCore.QByteArray: ...
    def setMinimumSectionSize(self, size: int) -> None: ...
    def minimumSectionSize(self) -> int: ...
    def setCascadingSectionResizes(self, enable: bool) -> None: ...
    def cascadingSectionResizes(self) -> bool: ...
    def swapSections(self, first: int, second: int) -> None: ...
    def sectionsHidden(self) -> bool: ...
    def setDefaultAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def defaultAlignment(self) -> QtCore.Qt.Alignment: ...
    def setDefaultSectionSize(self, size: int) -> None: ...
    def defaultSectionSize(self) -> int: ...
    def hiddenSectionCount(self) -> int: ...
    def showSection(self, alogicalIndex: int) -> None: ...
    def hideSection(self, alogicalIndex: int) -> None: ...
    def visualRegionForSelection(self, selection: QtCore.QItemSelection) -> QtGui.QRegion: ...
    def setSelection(self, rect: QtCore.QRect, flags: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def moveCursor(self, a0: QAbstractItemView.CursorAction, a1: typing.Union[QtCore.Qt.KeyboardModifiers,
QtCore.Qt.KeyboardModifier]) -> QtCore.QModelIndex: ...
    def isIndexHidden(self, index: QtCore.QModelIndex) -> bool: ...
    def indexAt(self, p: QtCore.QPoint) -> QtCore.QModelIndex: ...
    def scrollTo(self, index: QtCore.QModelIndex, hint: QAbstractItemView.ScrollHint) -> None: ...
    def visualRect(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def rowsInserted(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def dataChanged(self, topLeft: QtCore.QModelIndex, bottomRight: QtCore.QModelIndex, roles: typing.Iterable[int] = ...) -
> None: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def updateGeometries(self) -> None: ...
    def verticalOffset(self) -> int: ...
    def horizontalOffset(self) -> int: ...
    def sectionSizeFromContents(self, logicalIndex: int) -> QtCore.QSize: ...
    def paintSection(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRect, logicalIndex: int) -> None: ...
    def mouseDoubleClickEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def viewportEvent(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def currentChanged(self, current: QtCore.QModelIndex, old: QtCore.QModelIndex) -> None: ...
    @typing.overload
    def initializeSections(self) -> None: ...
    @typing.overload
    def initializeSections(self, start: int, end: int) -> None: ...
    def initialize(self) -> None: ...
    def sectionsAboutToBeRemoved(self, parent: QtCore.QModelIndex, logicalFirst: int, logicalLast: int) -> None: ...
    def sectionsInserted(self, parent: QtCore.QModelIndex, logicalFirst: int, logicalLast: int) -> None: ...
    @typing.overload
    def resizeSections(self) -> None: ...
    @typing.overload
    def resizeSections(self, mode: 'QHeaderView.ResizeMode') -> None: ...
    def updateSection(self, logicalIndex: int) -> None: ...
    sectionHandleDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    sectionCountChanged: typing.ClassVar[QtCore.pyqtSignal]
    sectionDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    sectionClicked: typing.ClassVar[QtCore.pyqtSignal]
    sectionPressed: typing.ClassVar[QtCore.pyqtSignal]
    sectionResized: typing.ClassVar[QtCore.pyqtSignal]
    sectionMoved: typing.ClassVar[QtCore.pyqtSignal]
    geometriesChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setOffsetToSectionPosition(self, visualIndex: int) -> None: ...
    def headerDataChanged(self, orientation: QtCore.Qt.Orientation, logicalFirst: int, logicalLast: int) -> None: ...
    def setOffset(self, offset: int) -> None: ...
    def sectionsMoved(self) -> bool: ...
    def setStretchLastSection(self, stretch: bool) -> None: ...
    def stretchLastSection(self) -> bool: ...
    def sortIndicatorOrder(self) -> QtCore.Qt.SortOrder: ...
    def sortIndicatorSection(self) -> int: ...
    def setSortIndicator(self, logicalIndex: int, order: QtCore.Qt.SortOrder) -> None: ...
    def isSortIndicatorShown(self) -> bool: ...
    def setSortIndicatorShown(self, show: bool) -> None: ...
    def stretchSectionCount(self) -> int: ...
    def highlightSections(self) -> bool: ...
    def setHighlightSections(self, highlight: bool) -> None: ...
    def logicalIndex(self, visualIndex: int) -> int: ...
    def visualIndex(self, logicalIndex: int) -> int: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def setSectionHidden(self, logicalIndex: int, hide: bool) -> None: ...
    def isSectionHidden(self, logicalIndex: int) -> bool: ...
    def resizeSection(self, logicalIndex: int, size: int) -> None: ...
    def moveSection(self, from_: int, to: int) -> None: ...
    def sectionViewportPosition(self, logicalIndex: int) -> int: ...
    def sectionPosition(self, logicalIndex: int) -> int: ...
    def sectionSize(self, logicalIndex: int) -> int: ...
    @typing.overload
    def logicalIndexAt(self, position: int) -> int: ...
    @typing.overload
    def logicalIndexAt(self, ax: int, ay: int) -> int: ...
    @typing.overload
    def logicalIndexAt(self, apos: QtCore.QPoint) -> int: ...
    def visualIndexAt(self, position: int) -> int: ...
    def sectionSizeHint(self, logicalIndex: int) -> int: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def length(self) -> int: ...
    def offset(self) -> int: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...


class QInputDialog(QDialog):

    class InputMode(int):
        TextInput = ... # type: QInputDialog.InputMode
        IntInput = ... # type: QInputDialog.InputMode
        DoubleInput = ... # type: QInputDialog.InputMode

    class InputDialogOption(int):
        NoButtons = ... # type: QInputDialog.InputDialogOption
        UseListViewForComboBoxItems = ... # type: QInputDialog.InputDialogOption
        UsePlainTextEditForTextInput = ... # type: QInputDialog.InputDialogOption

    class InputDialogOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> 
'QInputDialog.InputDialogOptions': ...
        def __xor__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> 
'QInputDialog.InputDialogOptions': ...
        def __ior__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> 
'QInputDialog.InputDialogOptions': ...
        def __or__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> 
'QInputDialog.InputDialogOptions': ...
        def __iand__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> 
'QInputDialog.InputDialogOptions': ...
        def __and__(self, f: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> 
'QInputDialog.InputDialogOptions': ...
        def __invert__(self) -> 'QInputDialog.InputDialogOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags, 
QtCore.Qt.WindowType] = ...) -> None: ...

    def doubleStep(self) -> float: ...
    def setDoubleStep(self, step: float) -> None: ...
    doubleValueSelected: typing.ClassVar[QtCore.pyqtSignal]
    doubleValueChanged: typing.ClassVar[QtCore.pyqtSignal]
    intValueSelected: typing.ClassVar[QtCore.pyqtSignal]
    intValueChanged: typing.ClassVar[QtCore.pyqtSignal]
    textValueSelected: typing.ClassVar[QtCore.pyqtSignal]
    textValueChanged: typing.ClassVar[QtCore.pyqtSignal]
    def done(self, result: int) -> None: ...
    def setVisible(self, visible: bool) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    @typing.overload
    def open(self) -> None: ...
    @typing.overload
    def open(self, slot: PYQT_SLOT) -> None: ...
    def cancelButtonText(self) -> str: ...
    def setCancelButtonText(self, text: typing.Optional[str]) -> None: ...
    def okButtonText(self) -> str: ...
    def setOkButtonText(self, text: typing.Optional[str]) -> None: ...
    def doubleDecimals(self) -> int: ...
    def setDoubleDecimals(self, decimals: int) -> None: ...
    def setDoubleRange(self, min: float, max: float) -> None: ...
    def doubleMaximum(self) -> float: ...
    def setDoubleMaximum(self, max: float) -> None: ...
    def doubleMinimum(self) -> float: ...
    def setDoubleMinimum(self, min: float) -> None: ...
    def doubleValue(self) -> float: ...
    def setDoubleValue(self, value: float) -> None: ...
    def intStep(self) -> int: ...
    def setIntStep(self, step: int) -> None: ...
    def setIntRange(self, min: int, max: int) -> None: ...
    def intMaximum(self) -> int: ...
    def setIntMaximum(self, max: int) -> None: ...
    def intMinimum(self) -> int: ...
    def setIntMinimum(self, min: int) -> None: ...
    def intValue(self) -> int: ...
    def setIntValue(self, value: int) -> None: ...
    def comboBoxItems(self) -> typing.List[str]: ...
    def setComboBoxItems(self, items: typing.Iterable[typing.Optional[str]]) -> None: ...
    def isComboBoxEditable(self) -> bool: ...
    def setComboBoxEditable(self, editable: bool) -> None: ...
    def textEchoMode(self) -> 'QLineEdit.EchoMode': ...
    def setTextEchoMode(self, mode: 'QLineEdit.EchoMode') -> None: ...
    def textValue(self) -> str: ...
    def setTextValue(self, text: typing.Optional[str]) -> None: ...
    def options(self) -> 'QInputDialog.InputDialogOptions': ...
    def setOptions(self, options: typing.Union['QInputDialog.InputDialogOptions', 'QInputDialog.InputDialogOption']) -> None:
...
    def testOption(self, option: 'QInputDialog.InputDialogOption') -> bool: ...
    def setOption(self, option: 'QInputDialog.InputDialogOption', on: bool = ...) -> None: ...
    def labelText(self) -> str: ...
    def setLabelText(self, text: typing.Optional[str]) -> None: ...
    def inputMode(self) -> 'QInputDialog.InputMode': ...
    def setInputMode(self, mode: 'QInputDialog.InputMode') -> None: ...
    @staticmethod
    def getMultiLineText(parent: typing.Optional[QWidget], title: typing.Optional[str], label: typing.Optional[str], text:
typing.Optional[str] = ..., flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ..., inputMethodHints:
typing.Union[QtCore.Qt.InputMethodHints, QtCore.Qt.InputMethodHint] = ...) -> typing.Tuple[str, typing.Optional[bool]]: ...
    @staticmethod
    def getItem(parent: typing.Optional[QWidget], title: typing.Optional[str], label: typing.Optional[str], items:
typing.Iterable[typing.Optional[str]], current: int = ..., editable: bool = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ..., inputMethodHints: typing.Union[QtCore.Qt.InputMethodHints, QtCore.Qt.InputMethodHint] =
...) -> typing.Tuple[str, typing.Optional[bool]]: ...
    @typing.overload
    @staticmethod
    def getDouble(parent: typing.Optional[QWidget], title: typing.Optional[str], label: typing.Optional[str], value: float = ...,
min: float = ..., max: float = ..., decimals: int = ..., flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] =
...) -> typing.Tuple[float, typing.Optional[bool]]: ...
    @typing.overload
    @staticmethod
    def getDouble(parent: typing.Optional[QWidget], title: typing.Optional[str], label: typing.Optional[str], value: float,
minValue: float, maxValue: float, decimals: int, flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType], step:
float) -> typing.Tuple[float, typing.Optional[bool]]: ...
    @staticmethod
    def getInt(parent: typing.Optional[QWidget], title: typing.Optional[str], label: typing.Optional[str], value: int = ..., min: int
= ..., max: int = ..., step: int = ..., flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) ->
typing.Tuple[int, typing.Optional[bool]]: ...
    @staticmethod
    def getText(parent: typing.Optional[QWidget], title: typing.Optional[str], label: typing.Optional[str], echo:
'QLineEdit.EchoMode' = ..., text: typing.Optional[str] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ..., inputMethodHints: typing.Union[QtCore.Qt.InputMethodHints, QtCore.Qt.InputMethodHint] =
...) -> typing.Tuple[str, typing.Optional[bool]]: ...


class QItemDelegate(QAbstractItemDelegate):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def editorEvent(self, event: typing.Optional[QtCore.QEvent], model: typing.Optional[QtCore.QAbstractItemModel], option:
'QStyleOptionViewItem', index: QtCore.QModelIndex) -> bool: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def drawFocus(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', rect: QtCore.QRect) ->
None: ...
    def drawDisplay(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', rect: QtCore.QRect, text:
typing.Optional[str]) -> None: ...
    def drawDecoration(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', rect: QtCore.QRect,
pixmap: QtGui.QPixmap) -> None: ...
    def drawCheck(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', rect: QtCore.QRect, state:
QtCore.Qt.CheckState) -> None: ...
    def drawBackground(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', index:
QtCore.QModelIndex) -> None: ...
    def setClipping(self, clip: bool) -> None: ...
    def hasClipping(self) -> bool: ...
    def setItemEditorFactory(self, factory: typing.Optional['QItemEditorFactory']) -> None: ...
    def itemEditorFactory(self) -> typing.Optional['QItemEditorFactory']: ...
    def updateEditorGeometry(self, editor: typing.Optional[QWidget], option: 'QStyleOptionViewItem', index: 
QtCore.QModelIndex) -> None: ...
    def setModelData(self, editor: typing.Optional[QWidget], model: typing.Optional[QtCore.QAbstractItemModel], index: 
QtCore.QModelIndex) -> None: ...
    def setEditorData(self, editor: typing.Optional[QWidget], index: QtCore.QModelIndex) -> None: ...
    def createEditor(self, parent: typing.Optional[QWidget], option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> 
typing.Optional[QWidget]: ...
    def sizeHint(self, option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> QtCore.QSize: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> 
None: ...


class QItemEditorCreatorBase(PyQt5.sip.wrapper):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QItemEditorCreatorBase') -> None: ...

    def valuePropertyName(self) -> QtCore.QByteArray: ...
    def createWidget(self, parent: typing.Optional[QWidget]) -> typing.Optional[QWidget]: ...


class QItemEditorFactory(PyQt5.sip.wrapper):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QItemEditorFactory') -> None: ...

    @staticmethod
    def setDefaultFactory(factory: typing.Optional['QItemEditorFactory']) -> None: ...
    @staticmethod
    def defaultFactory() -> typing.Optional['QItemEditorFactory']: ...
    def registerEditor(self, userType: int, creator: typing.Optional[QItemEditorCreatorBase]) -> None: ...
    def valuePropertyName(self, userType: int) -> QtCore.QByteArray: ...
    def createEditor(self, userType: int, parent: typing.Optional[QWidget]) -> typing.Optional[QWidget]: ...


class QKeyEventTransition(QtCore.QEventTransition):

    @typing.overload
    def __init__(self, sourceState: typing.Optional[QtCore.QState] = ...) -> None: ...
    @typing.overload
    def __init__(self, object: typing.Optional[QtCore.QObject], type: QtCore.QEvent.Type, key: int, sourceState: 
typing.Optional[QtCore.QState] = ...) -> None: ...

    def eventTest(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def onTransition(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def setModifierMask(self, modifiers: typing.Union[QtCore.Qt.KeyboardModifiers, QtCore.Qt.KeyboardModifier]) -> None: ...
    def modifierMask(self) -> QtCore.Qt.KeyboardModifiers: ...
    def setKey(self, key: int) -> None: ...
    def key(self) -> int: ...


class QKeySequenceEdit(QWidget):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, keySequence: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, 
typing.Optional[str], int], parent: typing.Optional[QWidget] = ...) -> None: ...

    def timerEvent(self, a0: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def keyReleaseEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    keySequenceChanged: typing.ClassVar[QtCore.pyqtSignal]
    editingFinished: typing.ClassVar[QtCore.pyqtSignal]
    def clear(self) -> None: ...
    def setKeySequence(self, keySequence: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey,
typing.Optional[str], int]) -> None: ...
    def keySequence(self) -> QtGui.QKeySequence: ...


class QLabel(QFrame):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ..., flags:
typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> None: ...

    def selectionStart(self) -> int: ...
    def selectedText(self) -> str: ...
    def hasSelectedText(self) -> bool: ...
    def setSelection(self, a0: int, a1: int) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def focusOutEvent(self, ev: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, ev: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def contextMenuEvent(self, ev: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def mouseReleaseEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def keyPressEvent(self, ev: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    linkHovered: typing.ClassVar[QtCore.pyqtSignal]
    linkActivated: typing.ClassVar[QtCore.pyqtSignal]
    def setText(self, a0: typing.Optional[str]) -> None: ...
    def setPixmap(self, a0: QtGui.QPixmap) -> None: ...
    def setPicture(self, a0: QtGui.QPicture) -> None: ...
    @typing.overload
    def setNum(self, a0: float) -> None: ...
    @typing.overload
    def setNum(self, a0: int) -> None: ...
    def setMovie(self, movie: typing.Optional[QtGui.QMovie]) -> None: ...
    def clear(self) -> None: ...
    def setOpenExternalLinks(self, open: bool) -> None: ...
    def textInteractionFlags(self) -> QtCore.Qt.TextInteractionFlags: ...
    def setTextInteractionFlags(self, flags: typing.Union[QtCore.Qt.TextInteractionFlags, QtCore.Qt.TextInteractionFlag]) ->
None: ...
    def openExternalLinks(self) -> bool: ...
    def heightForWidth(self, a0: int) -> int: ...
    def buddy(self) -> typing.Optional[QWidget]: ...
    def setBuddy(self, a0: typing.Optional[QWidget]) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setScaledContents(self, a0: bool) -> None: ...
    def hasScaledContents(self) -> bool: ...
    def setMargin(self, a0: int) -> None: ...
    def margin(self) -> int: ...
    def setIndent(self, a0: int) -> None: ...
    def indent(self) -> int: ...
    def wordWrap(self) -> bool: ...
    def setWordWrap(self, on: bool) -> None: ...
    def setAlignment(self, a0: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def setTextFormat(self, a0: QtCore.Qt.TextFormat) -> None: ...
    def textFormat(self) -> QtCore.Qt.TextFormat: ...
    def movie(self) -> typing.Optional[QtGui.QMovie]: ...
    def picture(self) -> typing.Optional[QtGui.QPicture]: ...
    def pixmap(self) -> typing.Optional[QtGui.QPixmap]: ...
    def text(self) -> str: ...
```

```
class QSpacerItem(QLayoutItem):

    @typing.overload
    def __init__(self, w: int, h: int, hPolicy: 'QSizePolicy.Policy' = ..., vPolicy: 'QSizePolicy.Policy' = ...) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QSpacerItem') -> None: ...

    def sizePolicy(self) -> 'QSizePolicy': ...
    def spacerItem(self) -> typing.Optional['QSpacerItem']: ...
    def geometry(self) -> QtCore.QRect: ...
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    def isEmpty(self) -> bool: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def changeSize(self, w: int, h: int, hPolicy: 'QSizePolicy.Policy' = ..., vPolicy: 'QSizePolicy.Policy' = ...) -> None: ...


class QWidgetItem(QLayoutItem):

    def __init__(self, w: typing.Optional[QWidget]) -> None: ...

    def controlTypes(self) -> 'QSizePolicy.ControlTypes': ...
    def heightForWidth(self, a0: int) -> int: ...
    def hasHeightForWidth(self) -> bool: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    def geometry(self) -> QtCore.QRect: ...
    def setGeometry(self, a0: QtCore.QRect) -> None: ...
    def isEmpty(self) -> bool: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def maximumSize(self) -> QtCore.QSize: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QLCDNumber(QFrame):

    class SegmentStyle(int):
        Outline = ... # type: QLCDNumber.SegmentStyle
        Filled = ... # type: QLCDNumber.SegmentStyle
        Flat = ... # type: QLCDNumber.SegmentStyle

    class Mode(int):
        Hex = ... # type: QLCDNumber.Mode
        Dec = ... # type: QLCDNumber.Mode
        Oct = ... # type: QLCDNumber.Mode
        Bin = ... # type: QLCDNumber.Mode

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, numDigits: int, parent: typing.Optional[QWidget] = ...) -> None: ...

    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    overflow: typing.ClassVar[QtCore.pyqtSignal]
    def setSmallDecimalPoint(self, a0: bool) -> None: ...
    def setBinMode(self) -> None: ...
    def setOctMode(self) -> None: ...
    def setDecMode(self) -> None: ...
    def setHexMode(self) -> None: ...
    @typing.overload
    def display(self, str: typing.Optional[str]) -> None: ...
    @typing.overload
    def display(self, num: float) -> None: ...
    @typing.overload
    def display(self, num: int) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def intValue(self) -> int: ...
    def value(self) -> float: ...
    def setSegmentStyle(self, a0: 'QLCDNumber.SegmentStyle') -> None: ...
    def segmentStyle(self) -> 'QLCDNumber.SegmentStyle': ...
    def setMode(self, a0: 'QLCDNumber.Mode') -> None: ...
    def mode(self) -> 'QLCDNumber.Mode': ...
    @typing.overload
    def checkOverflow(self, num: float) -> bool: ...
    @typing.overload
    def checkOverflow(self, num: int) -> bool: ...
    def setNumDigits(self, nDigits: int) -> None: ...
    def setDigitCount(self, nDigits: int) -> None: ...
    def digitCount(self) -> int: ...
    def smallDecimalPoint(self) -> bool: ...


class QLineEdit(QWidget):

    class ActionPosition(int):
        LeadingPosition = ... # type: QLineEdit.ActionPosition
        TrailingPosition = ... # type: QLineEdit.ActionPosition

    class EchoMode(int):
        Normal = ... # type: QLineEdit.EchoMode
        NoEcho = ... # type: QLineEdit.EchoMode
        Password = ... # type: QLineEdit.EchoMode
        PasswordEchoOnEdit = ... # type: QLineEdit.EchoMode

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, contents: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    inputRejected: typing.ClassVar[QtCore.pyqtSignal]
    def selectionLength(self) -> int: ...
    def selectionEnd(self) -> int: ...
    @typing.overload
    def addAction(self, action: typing.Optional[QAction]) -> None: ...
    @typing.overload
    def addAction(self, action: typing.Optional[QAction], position: 'QLineEdit.ActionPosition') -> None: ...
    @typing.overload
    def addAction(self, icon: QtGui.QIcon, position: 'QLineEdit.ActionPosition') -> typing.Optional[QAction]: ...
    def isClearButtonEnabled(self) -> bool: ...
    def setClearButtonEnabled(self, enable: bool) -> None: ...
    def cursorMoveStyle(self) -> QtCore.Qt.CursorMoveStyle: ...
    def setCursorMoveStyle(self, style: QtCore.Qt.CursorMoveStyle) -> None: ...
    def setPlaceholderText(self, a0: typing.Optional[str]) -> None: ...
    def placeholderText(self) -> str: ...
    def textMargins(self) -> QtCore.QMargins: ...
    def getTextMargins(self) -> typing.Tuple[typing.Optional[int], typing.Optional[int], typing.Optional[int],
typing.Optional[int]]: ...
    @typing.overload
    def setTextMargins(self, left: int, top: int, right: int, bottom: int) -> None: ...
    @typing.overload
    def setTextMargins(self, margins: QtCore.QMargins) -> None: ...
    def completer(self) -> typing.Optional[QCompleter]: ...
    def setCompleter(self, completer: typing.Optional[QCompleter]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    @typing.overload
    def inputMethodQuery(self, a0: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    @typing.overload
    def inputMethodQuery(self, property: QtCore.Qt.InputMethodQuery, argument: typing.Any) -> typing.Any: ...
    def cursorRect(self) -> QtCore.QRect: ...
    def inputMethodEvent(self, a0: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def contextMenuEvent(self, a0: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def dropEvent(self, a0: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragLeaveEvent(self, e: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragMoveEvent(self, e: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def dragEnterEvent(self, a0: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def focusOutEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def mouseDoubleClickEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionFrame']) -> None: ...
    selectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    editingFinished: typing.ClassVar[QtCore.pyqtSignal]
    returnPressed: typing.ClassVar[QtCore.pyqtSignal]
    cursorPositionChanged: typing.ClassVar[QtCore.pyqtSignal]
    textEdited: typing.ClassVar[QtCore.pyqtSignal]
    textChanged: typing.ClassVar[QtCore.pyqtSignal]
    def createStandardContextMenu(self) -> typing.Optional['QMenu']: ...
    def insert(self, a0: typing.Optional[str]) -> None: ...
    def deselect(self) -> None: ...
    def paste(self) -> None: ...
    def copy(self) -> None: ...
    def cut(self) -> None: ...
    def redo(self) -> None: ...
    def undo(self) -> None: ...
    def selectAll(self) -> None: ...
    def clear(self) -> None: ...
    def setText(self, a0: typing.Optional[str]) -> None: ...
    def hasAcceptableInput(self) -> bool: ...
    def setInputMask(self, inputMask: typing.Optional[str]) -> None: ...
    def inputMask(self) -> str: ...
    def dragEnabled(self) -> bool: ...
    def setDragEnabled(self, b: bool) -> None: ...
    def isRedoAvailable(self) -> bool: ...
    def isUndoAvailable(self) -> bool: ...
    def selectionStart(self) -> int: ...
    def selectedText(self) -> str: ...
    def hasSelectedText(self) -> bool: ...
    def setSelection(self, a0: int, a1: int) -> None: ...
    def setModified(self, a0: bool) -> None: ...
    def isModified(self) -> bool: ...
    def end(self, mark: bool) -> None: ...
    def home(self, mark: bool) -> None: ...
    def del_(self) -> None: ...
    def backspace(self) -> None: ...
    def cursorWordBackward(self, mark: bool) -> None: ...
    def cursorWordForward(self, mark: bool) -> None: ...
    def cursorBackward(self, mark: bool, steps: int = ...) -> None: ...
    def cursorForward(self, mark: bool, steps: int = ...) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def setAlignment(self, flag: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def cursorPositionAt(self, pos: QtCore.QPoint) -> int: ...
    def setCursorPosition(self, a0: int) -> None: ...
    def cursorPosition(self) -> int: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def validator(self) -> typing.Optional[QtGui.QValidator]: ...
    def setValidator(self, a0: typing.Optional[QtGui.QValidator]) -> None: ...
    def setReadOnly(self, a0: bool) -> None: ...
    def isReadOnly(self) -> bool: ...
    def setEchoMode(self, a0: 'QLineEdit.EchoMode') -> None: ...
    def echoMode(self) -> 'QLineEdit.EchoMode': ...
    def hasFrame(self) -> bool: ...
    def setFrame(self, a0: bool) -> None: ...
    def setMaxLength(self, a0: int) -> None: ...
    def maxLength(self) -> int: ...
    def displayText(self) -> str: ...
    def text(self) -> str: ...


class QListView(QAbstractItemView):
    class ViewMode(int):
        ListMode = ... # type: QListView.ViewMode
        IconMode = ... # type: QListView.ViewMode

    class LayoutMode(int):
        SinglePass = ... # type: QListView.LayoutMode
        Batched = ... # type: QListView.LayoutMode

    class ResizeMode(int):
        Fixed = ... # type: QListView.ResizeMode
        Adjust = ... # type: QListView.ResizeMode

    class Flow(int):
        LeftToRight = ... # type: QListView.Flow
        TopToBottom = ... # type: QListView.Flow

    class Movement(int):
        Static = ... # type: QListView.Movement
        Free = ... # type: QListView.Movement
        Snap = ... # type: QListView.Movement

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def itemAlignment(self) -> QtCore.Qt.Alignment: ...
    def setItemAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def currentChanged(self, current: QtCore.QModelIndex, previous: QtCore.QModelIndex) -> None: ...
    def selectionChanged(self, selected: QtCore.QItemSelection, deselected: QtCore.QItemSelection) -> None: ...
    def isSelectionRectVisible(self) -> bool: ...
    def setSelectionRectVisible(self, show: bool) -> None: ...
    def wordWrap(self) -> bool: ...
    def setWordWrap(self, on: bool) -> None: ...
    def batchSize(self) -> int: ...
    def setBatchSize(self, batchSize: int) -> None: ...
    def viewportSizeHint(self) -> QtCore.QSize: ...
    def isIndexHidden(self, index: QtCore.QModelIndex) -> bool: ...
    def updateGeometries(self) -> None: ...
    def selectedIndexes(self) -> typing.List[QtCore.QModelIndex]: ...
    def visualRegionForSelection(self, selection: QtCore.QItemSelection) -> QtGui.QRegion: ...
    def setSelection(self, rect: QtCore.QRect, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def setPositionForIndex(self, position: QtCore.QPoint, index: QtCore.QModelIndex) -> None: ...
    def rectForIndex(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def moveCursor(self, cursorAction: QAbstractItemView.CursorAction, modifiers: typing.Union[QtCore.Qt.KeyboardModifiers,
QtCore.Qt.KeyboardModifier]) -> QtCore.QModelIndex: ...
    def verticalOffset(self) -> int: ...
    def horizontalOffset(self) -> int: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def viewOptions(self) -> 'QStyleOptionViewItem': ...
    def startDrag(self, supportedActions: typing.Union[QtCore.Qt.DropActions, QtCore.Qt.DropAction]) -> None: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def dropEvent(self, e: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragLeaveEvent(self, e: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragMoveEvent(self, e: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def timerEvent(self, e: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def rowsAboutToBeRemoved(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def rowsInserted(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def dataChanged(self, topLeft: QtCore.QModelIndex, bottomRight: QtCore.QModelIndex, roles: typing.Iterable[int] = ...) -
> None: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    indexesMoved: typing.ClassVar[QtCore.pyqtSignal]
    def setRootIndex(self, index: QtCore.QModelIndex) -> None: ...
    def reset(self) -> None: ...
    def indexAt(self, p: QtCore.QPoint) -> QtCore.QModelIndex: ...
    def scrollTo(self, index: QtCore.QModelIndex, hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def visualRect(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def uniformItemSizes(self) -> bool: ...
    def setUniformItemSizes(self, enable: bool) -> None: ...
    def modelColumn(self) -> int: ...
    def setModelColumn(self, column: int) -> None: ...
    def setRowHidden(self, row: int, hide: bool) -> None: ...
    def isRowHidden(self, row: int) -> bool: ...
    def clearPropertyFlags(self) -> None: ...
    def viewMode(self) -> 'QListView.ViewMode': ...
    def setViewMode(self, mode: 'QListView.ViewMode') -> None: ...
    def gridSize(self) -> QtCore.QSize: ...
    def setGridSize(self, size: QtCore.QSize) -> None: ...
    def spacing(self) -> int: ...
    def setSpacing(self, space: int) -> None: ...
    def layoutMode(self) -> 'QListView.LayoutMode': ...
    def setLayoutMode(self, mode: 'QListView.LayoutMode') -> None: ...
    def resizeMode(self) -> 'QListView.ResizeMode': ...
    def setResizeMode(self, mode: 'QListView.ResizeMode') -> None: ...
    def isWrapping(self) -> bool: ...
    def setWrapping(self, enable: bool) -> None: ...
    def flow(self) -> 'QListView.Flow': ...
    def setFlow(self, flow: 'QListView.Flow') -> None: ...
    def movement(self) -> 'QListView.Movement': ...
    def setMovement(self, movement: 'QListView.Movement') -> None: ...


class QListWidgetItem(PyQt5.sip.wrapper):

    class ItemType(int):
        Type = ... # type: QListWidgetItem.ItemType
        UserType = ... # type: QListWidgetItem.ItemType

    @typing.overload
    def __init__(self, parent: typing.Optional['QListWidget'] = ..., type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional['QListWidget'] = ..., type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, icon: QtGui.QIcon, text: typing.Optional[str], parent: typing.Optional['QListWidget'] = ..., type: int = ...) -
> None: ...
    @typing.overload
    def __init__(self, other: 'QListWidgetItem') -> None: ...

    def __ge__(self, other: 'QListWidgetItem') -> bool: ...
    def isHidden(self) -> bool: ...
    def setHidden(self, ahide: bool) -> None: ...
    def isSelected(self) -> bool: ...
    def setSelected(self, aselect: bool) -> None: ...
    def setForeground(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def foreground(self) -> QtGui.QBrush: ...
    def setBackground(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def background(self) -> QtGui.QBrush: ...
    def setFont(self, afont: QtGui.QFont) -> None: ...
    def setWhatsThis(self, awhatsThis: typing.Optional[str]) -> None: ...
    def setToolTip(self, atoolTip: typing.Optional[str]) -> None: ...
    def setStatusTip(self, astatusTip: typing.Optional[str]) -> None: ...
    def setIcon(self, aicon: QtGui.QIcon) -> None: ...
    def setText(self, atext: typing.Optional[str]) -> None: ...
    def setFlags(self, aflags: typing.Union[QtCore.Qt.ItemFlags, QtCore.Qt.ItemFlag]) -> None: ...
    def type(self) -> int: ...
    def write(self, out: QtCore.QDataStream) -> None: ...
    def read(self, in_: QtCore.QDataStream) -> None: ...
    def __lt__(self, other: 'QListWidgetItem') -> bool: ...
    def setData(self, role: int, value: typing.Any) -> None: ...
    def data(self, role: int) -> typing.Any: ...
    def setSizeHint(self, size: QtCore.QSize) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setCheckState(self, state: QtCore.Qt.CheckState) -> None: ...
    def checkState(self) -> QtCore.Qt.CheckState: ...
    def setTextAlignment(self, alignment: int) -> None: ...
    def textAlignment(self) -> int: ...
    def font(self) -> QtGui.QFont: ...
    def whatsThis(self) -> str: ...
    def toolTip(self) -> str: ...
    def statusTip(self) -> str: ...
    def icon(self) -> QtGui.QIcon: ...
    def text(self) -> str: ...
    def flags(self) -> QtCore.Qt.ItemFlags: ...
    def listWidget(self) -> typing.Optional['QListWidget']: ...
    def clone(self) -> typing.Optional['QListWidgetItem']: ...


class QListWidget(QListView):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isPersistentEditorOpen(self, item: typing.Optional[QListWidgetItem]) -> bool: ...
    def setSelectionModel(self, selectionModel: typing.Optional[QtCore.QItemSelectionModel]) -> None: ...
    def removeItemWidget(self, aItem: typing.Optional[QListWidgetItem]) -> None: ...
    def dropEvent(self, event: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def isSortingEnabled(self) -> bool: ...
    def setSortingEnabled(self, enable: bool) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def itemFromIndex(self, index: QtCore.QModelIndex) -> typing.Optional[QListWidgetItem]: ...
    def indexFromItem(self, item: typing.Optional[QListWidgetItem]) -> QtCore.QModelIndex: ...
    def items(self, data: typing.Optional[QtCore.QMimeData]) -> typing.List[QListWidgetItem]: ...
    def supportedDropActions(self) -> QtCore.Qt.DropActions: ...
    def dropMimeData(self, index: int, data: typing.Optional[QtCore.QMimeData], action: QtCore.Qt.DropAction) -> bool: ...
    def mimeData(self, items: typing.Iterable[QListWidgetItem]) -> typing.Optional[QtCore.QMimeData]: ...
    def mimeTypes(self) -> typing.List[str]: ...
    itemSelectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentRowChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentItemChanged: typing.ClassVar[QtCore.pyqtSignal]
    itemChanged: typing.ClassVar[QtCore.pyqtSignal]
    itemEntered: typing.ClassVar[QtCore.pyqtSignal]
    itemActivated: typing.ClassVar[QtCore.pyqtSignal]
    itemDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    itemClicked: typing.ClassVar[QtCore.pyqtSignal]
    itemPressed: typing.ClassVar[QtCore.pyqtSignal]
    def scrollToItem(self, item: typing.Optional[QListWidgetItem], hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def clear(self) -> None: ...
    def findItems(self, text: typing.Optional[str], flags: typing.Union[QtCore.Qt.MatchFlags, QtCore.Qt.MatchFlag]) -> 
typing.List[QListWidgetItem]: ...
    def selectedItems(self) -> typing.List[QListWidgetItem]: ...
    def closePersistentEditor(self, item: typing.Optional[QListWidgetItem]) -> None: ...
    def openPersistentEditor(self, item: typing.Optional[QListWidgetItem]) -> None: ...
    def editItem(self, item: typing.Optional[QListWidgetItem]) -> None: ...
    def sortItems(self, order: QtCore.Qt.SortOrder = ...) -> None: ...
    def visualItemRect(self, item: typing.Optional[QListWidgetItem]) -> QtCore.QRect: ...
    def setItemWidget(self, item: typing.Optional[QListWidgetItem], widget: typing.Optional[QWidget]) -> None: ...
    def itemWidget(self, item: typing.Optional[QListWidgetItem]) -> typing.Optional[QWidget]: ...
    @typing.overload
    def itemAt(self, p: QtCore.QPoint) -> typing.Optional[QListWidgetItem]: ...
    @typing.overload
    def itemAt(self, ax: int, ay: int) -> typing.Optional[QListWidgetItem]: ...
    @typing.overload
    def setCurrentRow(self, row: int) -> None: ...
    @typing.overload
    def setCurrentRow(self, row: int, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags, 
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def currentRow(self) -> int: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QListWidgetItem]) -> None: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QListWidgetItem], command: 
typing.Union[QtCore.QItemSelectionModel.SelectionFlags, QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def currentItem(self) -> typing.Optional[QListWidgetItem]: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def takeItem(self, row: int) -> typing.Optional[QListWidgetItem]: ...
    def addItems(self, labels: typing.Iterable[typing.Optional[str]]) -> None: ...
    @typing.overload
    def addItem(self, aitem: typing.Optional[QListWidgetItem]) -> None: ...
    @typing.overload
    def addItem(self, label: typing.Optional[str]) -> None: ...
    def insertItems(self, row: int, labels: typing.Iterable[typing.Optional[str]]) -> None: ...
    @typing.overload
    def insertItem(self, row: int, item: typing.Optional[QListWidgetItem]) -> None: ...
    @typing.overload
    def insertItem(self, row: int, label: typing.Optional[str]) -> None: ...
    def row(self, item: typing.Optional[QListWidgetItem]) -> int: ...
    def item(self, row: int) -> typing.Optional[QListWidgetItem]: ...


class QMainWindow(QWidget):

    class DockOption(int):
        AnimatedDocks = ... # type: QMainWindow.DockOption
        AllowNestedDocks = ... # type: QMainWindow.DockOption
        AllowTabbedDocks = ... # type: QMainWindow.DockOption
        ForceTabbedDocks = ... # type: QMainWindow.DockOption
        VerticalTabs = ... # type: QMainWindow.DockOption
        GroupedDragging = ... # type: QMainWindow.DockOption

    class DockOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) ->
'QMainWindow.DockOptions': ...
        def __xor__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) ->
'QMainWindow.DockOptions': ...
        def __ior__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) ->
'QMainWindow.DockOptions': ...
        def __or__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) ->
'QMainWindow.DockOptions': ...
        def __iand__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) ->
'QMainWindow.DockOptions': ...
        def __and__(self, f: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) ->
'QMainWindow.DockOptions': ...
        def __invert__(self) -> 'QMainWindow.DockOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def resizeDocks(self, docks: typing.Iterable[QDockWidget], sizes: typing.Iterable[int], orientation: QtCore.Qt.Orientation) -
> None: ...
    def takeCentralWidget(self) -> typing.Optional[QWidget]: ...
    def tabifiedDockWidgets(self, dockwidget: typing.Optional[QDockWidget]) -> typing.List[QDockWidget]: ...
    def setTabPosition(self, areas: typing.Union[QtCore.Qt.DockWidgetAreas, QtCore.Qt.DockWidgetArea], tabPosition:
'QTabWidget.TabPosition') -> None: ...
    def tabPosition(self, area: QtCore.Qt.DockWidgetArea) -> 'QTabWidget.TabPosition': ...
    def setTabShape(self, tabShape: 'QTabWidget.TabShape') -> None: ...
    def tabShape(self) -> 'QTabWidget.TabShape': ...
    def setDocumentMode(self, enabled: bool) -> None: ...
    def documentMode(self) -> bool: ...
    def restoreDockWidget(self, dockwidget: typing.Optional[QDockWidget]) -> bool: ...
    def unifiedTitleAndToolBarOnMac(self) -> bool: ...
    def setUnifiedTitleAndToolBarOnMac(self, set: bool) -> None: ...
    def toolBarBreak(self, toolbar: typing.Optional['QToolBar']) -> bool: ...
    def removeToolBarBreak(self, before: typing.Optional['QToolBar']) -> None: ...
    def dockOptions(self) -> 'QMainWindow.DockOptions': ...
    def setDockOptions(self, options: typing.Union['QMainWindow.DockOptions', 'QMainWindow.DockOption']) -> None: ...
    def tabifyDockWidget(self, first: typing.Optional[QDockWidget], second: typing.Optional[QDockWidget]) -> None: ...
    def setMenuWidget(self, menubar: typing.Optional[QWidget]) -> None: ...
    def menuWidget(self) -> typing.Optional[QWidget]: ...
    def isSeparator(self, pos: QtCore.QPoint) -> bool: ...
    def isDockNestingEnabled(self) -> bool: ...
    def isAnimated(self) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def contextMenuEvent(self, event: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    tabifiedDockWidgetActivated: typing.ClassVar[QtCore.pyqtSignal]
    toolButtonStyleChanged: typing.ClassVar[QtCore.pyqtSignal]
    iconSizeChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setDockNestingEnabled(self, enabled: bool) -> None: ...
    def setAnimated(self, enabled: bool) -> None: ...
    def createPopupMenu(self) -> typing.Optional['QMenu']: ...
    def restoreState(self, state: typing.Union[QtCore.QByteArray, bytes, bytearray], version: int = ...) -> bool: ...
    def saveState(self, version: int = ...) -> QtCore.QByteArray: ...
    def dockWidgetArea(self, dockwidget: typing.Optional[QDockWidget]) -> QtCore.Qt.DockWidgetArea: ...
    def removeDockWidget(self, dockwidget: typing.Optional[QDockWidget]) -> None: ...
    def splitDockWidget(self, after: typing.Optional[QDockWidget], dockwidget: typing.Optional[QDockWidget], orientation:
QtCore.Qt.Orientation) -> None: ...
    @typing.overload
    def addDockWidget(self, area: QtCore.Qt.DockWidgetArea, dockwidget: typing.Optional[QDockWidget]) -> None: ...
    @typing.overload
    def addDockWidget(self, area: QtCore.Qt.DockWidgetArea, dockwidget: typing.Optional[QDockWidget], orientation:
QtCore.Qt.Orientation) -> None: ...
    def toolBarArea(self, toolbar: typing.Optional['QToolBar']) -> QtCore.Qt.ToolBarArea: ...
    def removeToolBar(self, toolbar: typing.Optional['QToolBar']) -> None: ...
    def insertToolBar(self, before: typing.Optional['QToolBar'], toolbar: typing.Optional['QToolBar']) -> None: ...
    @typing.overload
    def addToolBar(self, area: QtCore.Qt.ToolBarArea, toolbar: typing.Optional['QToolBar']) -> None: ...
    @typing.overload
    def addToolBar(self, toolbar: typing.Optional['QToolBar']) -> None: ...
    @typing.overload
    def addToolBar(self, title: typing.Optional[str]) -> typing.Optional['QToolBar']: ...
    def insertToolBarBreak(self, before: typing.Optional['QToolBar']) -> None: ...
    def addToolBarBreak(self, area: QtCore.Qt.ToolBarArea = ...) -> None: ...
    def corner(self, corner: QtCore.Qt.Corner) -> QtCore.Qt.DockWidgetArea: ...
    def setCorner(self, corner: QtCore.Qt.Corner, area: QtCore.Qt.DockWidgetArea) -> None: ...
    def setCentralWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def centralWidget(self) -> typing.Optional[QWidget]: ...
    def setStatusBar(self, statusbar: typing.Optional['QStatusBar']) -> None: ...
    def statusBar(self) -> typing.Optional['QStatusBar']: ...
    def setMenuBar(self, menubar: typing.Optional['QMenuBar']) -> None: ...
    def menuBar(self) -> typing.Optional['QMenuBar']: ...
    def setToolButtonStyle(self, toolButtonStyle: QtCore.Qt.ToolButtonStyle) -> None: ...
    def toolButtonStyle(self) -> QtCore.Qt.ToolButtonStyle: ...
    def setIconSize(self, iconSize: QtCore.QSize) -> None: ...
    def iconSize(self) -> QtCore.QSize: ...


class QMdiArea(QAbstractScrollArea):

    class WindowOrder(int):
        CreationOrder = ... # type: QMdiArea.WindowOrder
        StackingOrder = ... # type: QMdiArea.WindowOrder
        ActivationHistoryOrder = ... # type: QMdiArea.WindowOrder

    class ViewMode(int):
        SubWindowView = ... # type: QMdiArea.ViewMode
        TabbedView = ... # type: QMdiArea.ViewMode

    class AreaOption(int):
        DontMaximizeSubWindowOnActivation = ... # type: QMdiArea.AreaOption

    class AreaOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> 'QMdiArea.AreaOptions': ...
        def __xor__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> 'QMdiArea.AreaOptions': ...
        def __ior__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> 'QMdiArea.AreaOptions': ...
        def __or__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> 'QMdiArea.AreaOptions': ...
        def __iand__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> 'QMdiArea.AreaOptions': ...
        def __and__(self, f: typing.Union['QMdiArea.AreaOptions', 'QMdiArea.AreaOption']) -> 'QMdiArea.AreaOptions': ...
        def __invert__(self) -> 'QMdiArea.AreaOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def tabsMovable(self) -> bool: ...
    def setTabsMovable(self, movable: bool) -> None: ...
    def tabsClosable(self) -> bool: ...
    def setTabsClosable(self, closable: bool) -> None: ...
    def setDocumentMode(self, enabled: bool) -> None: ...
    def documentMode(self) -> bool: ...
    def tabPosition(self) -> 'QTabWidget.TabPosition': ...
    def setTabPosition(self, position: 'QTabWidget.TabPosition') -> None: ...
    def tabShape(self) -> 'QTabWidget.TabShape': ...
    def setTabShape(self, shape: 'QTabWidget.TabShape') -> None: ...
    def viewMode(self) -> 'QMdiArea.ViewMode': ...
    def setViewMode(self, mode: 'QMdiArea.ViewMode') -> None: ...
    def setActivationOrder(self, order: 'QMdiArea.WindowOrder') -> None: ...
    def activationOrder(self) -> 'QMdiArea.WindowOrder': ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def viewportEvent(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def showEvent(self, showEvent: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def timerEvent(self, timerEvent: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def resizeEvent(self, resizeEvent: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def childEvent(self, childEvent: typing.Optional[QtCore.QChildEvent]) -> None: ...
    def paintEvent(self, paintEvent: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def setupViewport(self, viewport: typing.Optional[QWidget]) -> None: ...
    def activatePreviousSubWindow(self) -> None: ...
    def activateNextSubWindow(self) -> None: ...
    def closeAllSubWindows(self) -> None: ...
    def closeActiveSubWindow(self) -> None: ...
    def cascadeSubWindows(self) -> None: ...
    def tileSubWindows(self) -> None: ...
    def setActiveSubWindow(self, window: typing.Optional['QMdiSubWindow']) -> None: ...
    subWindowActivated: typing.ClassVar[QtCore.pyqtSignal]
    def testOption(self, opton: 'QMdiArea.AreaOption') -> bool: ...
    def setOption(self, option: 'QMdiArea.AreaOption', on: bool = ...) -> None: ...
    def setBackground(self, background: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], 
QtGui.QGradient]) -> None: ...
    def background(self) -> QtGui.QBrush: ...
    def removeSubWindow(self, widget: typing.Optional[QWidget]) -> None: ...
    def currentSubWindow(self) -> typing.Optional['QMdiSubWindow']: ...
    def subWindowList(self, order: 'QMdiArea.WindowOrder' = ...) -> typing.List['QMdiSubWindow']: ...
    def addSubWindow(self, widget: typing.Optional[QWidget], flags: typing.Union[QtCore.Qt.WindowFlags, 
QtCore.Qt.WindowType] = ...) -> typing.Optional['QMdiSubWindow']: ...
    def activeSubWindow(self) -> typing.Optional['QMdiSubWindow']: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QMdiSubWindow(QWidget):

    class SubWindowOption(int):
        RubberBandResize = ... # type: QMdiSubWindow.SubWindowOption
        RubberBandMove = ... # type: QMdiSubWindow.SubWindowOption

    class SubWindowOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
'QMdiSubWindow.SubWindowOptions': ...
        def __xor__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
'QMdiSubWindow.SubWindowOptions': ...
        def __ior__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
'QMdiSubWindow.SubWindowOptions': ...
        def __or__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
'QMdiSubWindow.SubWindowOptions': ...
        def __iand__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
'QMdiSubWindow.SubWindowOptions': ...
        def __and__(self, f: typing.Union['QMdiSubWindow.SubWindowOptions', 'QMdiSubWindow.SubWindowOption']) ->
'QMdiSubWindow.SubWindowOptions': ...
        def __invert__(self) -> 'QMdiSubWindow.SubWindowOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def childEvent(self, childEvent: typing.Optional[QtCore.QChildEvent]) -> None: ...
    def focusOutEvent(self, focusOutEvent: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, focusInEvent: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def contextMenuEvent(self, contextMenuEvent: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def keyPressEvent(self, keyEvent: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def mouseMoveEvent(self, mouseEvent: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, mouseEvent: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseDoubleClickEvent(self, mouseEvent: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, mouseEvent: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, paintEvent: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def moveEvent(self, moveEvent: typing.Optional[QtGui.QMoveEvent]) -> None: ...
    def timerEvent(self, timerEvent: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def resizeEvent(self, resizeEvent: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def leaveEvent(self, leaveEvent: typing.Optional[QtCore.QEvent]) -> None: ...
    def closeEvent(self, closeEvent: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def changeEvent(self, changeEvent: typing.Optional[QtCore.QEvent]) -> None: ...
    def hideEvent(self, hideEvent: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def showEvent(self, showEvent: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def showShaded(self) -> None: ...
    def showSystemMenu(self) -> None: ...
    aboutToActivate: typing.ClassVar[QtCore.pyqtSignal]
    windowStateChanged: typing.ClassVar[QtCore.pyqtSignal]
    def mdiArea(self) -> typing.Optional[QMdiArea]: ...
    def systemMenu(self) -> typing.Optional['QMenu']: ...
    def setSystemMenu(self, systemMenu: typing.Optional['QMenu']) -> None: ...
    def keyboardPageStep(self) -> int: ...
    def setKeyboardPageStep(self, step: int) -> None: ...
    def keyboardSingleStep(self) -> int: ...
    def setKeyboardSingleStep(self, step: int) -> None: ...
    def testOption(self, a0: 'QMdiSubWindow.SubWindowOption') -> bool: ...
    def setOption(self, option: 'QMdiSubWindow.SubWindowOption', on: bool = ...) -> None: ...
    def isShaded(self) -> bool: ...
    def widget(self) -> typing.Optional[QWidget]: ...
    def setWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QMenu(QWidget):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, title: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    @typing.overload
    def showTearOffMenu(self) -> None: ...
    @typing.overload
    def showTearOffMenu(self, pos: QtCore.QPoint) -> None: ...
    def setToolTipsVisible(self, visible: bool) -> None: ...
    def toolTipsVisible(self) -> bool: ...
    @typing.overload
    def insertSection(self, before: typing.Optional[QAction], text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def insertSection(self, before: typing.Optional[QAction], icon: QtGui.QIcon, text: typing.Optional[str]) ->
typing.Optional[QAction]: ...
    @typing.overload
    def addSection(self, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addSection(self, icon: QtGui.QIcon, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    def setSeparatorsCollapsible(self, collapse: bool) -> None: ...
    def separatorsCollapsible(self) -> bool: ...
    def isEmpty(self) -> bool: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def timerEvent(self, a0: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def actionEvent(self, a0: typing.Optional[QtGui.QActionEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def hideEvent(self, a0: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def leaveEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def enterEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def wheelEvent(self, a0: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionMenuItem'], action: typing.Optional[QAction]) -> None: ...
    def columnCount(self) -> int: ...
    triggered: typing.ClassVar[QtCore.pyqtSignal]
    hovered: typing.ClassVar[QtCore.pyqtSignal]
    aboutToShow: typing.ClassVar[QtCore.pyqtSignal]
    aboutToHide: typing.ClassVar[QtCore.pyqtSignal]
    def setNoReplayFor(self, widget: typing.Optional[QWidget]) -> None: ...
    def setIcon(self, icon: QtGui.QIcon) -> None: ...
    def icon(self) -> QtGui.QIcon: ...
    def setTitle(self, title: typing.Optional[str]) -> None: ...
    def title(self) -> str: ...
    def menuAction(self) -> typing.Optional[QAction]: ...
    def actionAt(self, a0: QtCore.QPoint) -> typing.Optional[QAction]: ...
    def actionGeometry(self, a0: typing.Optional[QAction]) -> QtCore.QRect: ...
    def sizeHint(self) -> QtCore.QSize: ...
    @typing.overload
    def exec(self) -> typing.Optional[QAction]: ...
    @typing.overload
    def exec(self, pos: QtCore.QPoint, action: typing.Optional[QAction] = ...) -> typing.Optional[QAction]: ...
    @typing.overload
    @staticmethod
    def exec(actions: typing.Iterable[QAction], pos: QtCore.QPoint, at: typing.Optional[QAction] = ..., parent:
typing.Optional[QWidget] = ...) -> typing.Optional[QAction]: ...
    @typing.overload
    def exec_(self) -> typing.Optional[QAction]: ...
    @typing.overload
    def exec_(self, p: QtCore.QPoint, action: typing.Optional[QAction] = ...) -> typing.Optional[QAction]: ...
    @typing.overload
    @staticmethod
    def exec_(actions: typing.Iterable[QAction], pos: QtCore.QPoint, at: typing.Optional[QAction] = ..., parent:
typing.Optional[QWidget] = ...) -> typing.Optional[QAction]: ...
    def popup(self, p: QtCore.QPoint, action: typing.Optional[QAction] = ...) -> None: ...
    def activeAction(self) -> typing.Optional[QAction]: ...
    def setActiveAction(self, act: typing.Optional[QAction]) -> None: ...
    def defaultAction(self) -> typing.Optional[QAction]: ...
    def setDefaultAction(self, a0: typing.Optional[QAction]) -> None: ...
    def hideTearOffMenu(self) -> None: ...
    def isTearOffMenuVisible(self) -> bool: ...
    def isTearOffEnabled(self) -> bool: ...
    def setTearOffEnabled(self, a0: bool) -> None: ...
    def clear(self) -> None: ...
    def insertSeparator(self, before: typing.Optional[QAction]) -> typing.Optional[QAction]: ...
    def insertMenu(self, before: typing.Optional[QAction], menu: typing.Optional['QMenu']) -> typing.Optional[QAction]: ...
    def addSeparator(self) -> typing.Optional[QAction]: ...
    @typing.overload
    def addMenu(self, menu: typing.Optional['QMenu']) -> typing.Optional[QAction]: ...
    @typing.overload
    def addMenu(self, title: typing.Optional[str]) -> typing.Optional['QMenu']: ...
    @typing.overload
    def addMenu(self, icon: QtGui.QIcon, title: typing.Optional[str]) -> typing.Optional['QMenu']: ...
    @typing.overload
    def addAction(self, action: typing.Optional[QAction]) -> None: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, icon: QtGui.QIcon, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str], slot: PYQT_SLOT, shortcut: typing.Union[QtGui.QKeySequence,
QtGui.QKeySequence.StandardKey, typing.Optional[str], int] = ...) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, icon: QtGui.QIcon, text: typing.Optional[str], slot: PYQT_SLOT, shortcut:
typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, typing.Optional[str], int] = ...) ->
typing.Optional[QAction]: ...


class QMenuBar(QWidget):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def setNativeMenuBar(self, nativeMenuBar: bool) -> None: ...
    def isNativeMenuBar(self) -> bool: ...
    def timerEvent(self, a0: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def focusInEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusOutEvent(self, a0: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def actionEvent(self, a0: typing.Optional[QtGui.QActionEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def leaveEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionMenuItem'], action: typing.Optional[QAction]) -> None: ...
    hovered: typing.ClassVar[QtCore.pyqtSignal]
    triggered: typing.ClassVar[QtCore.pyqtSignal]
    def setVisible(self, visible: bool) -> None: ...
    def cornerWidget(self, corner: QtCore.Qt.Corner = ...) -> typing.Optional[QWidget]: ...
    def setCornerWidget(self, widget: typing.Optional[QWidget], corner: QtCore.Qt.Corner = ...) -> None: ...
    def actionAt(self, a0: QtCore.QPoint) -> typing.Optional[QAction]: ...
    def actionGeometry(self, a0: typing.Optional[QAction]) -> QtCore.QRect: ...
    def heightForWidth(self, a0: int) -> int: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def isDefaultUp(self) -> bool: ...
    def setDefaultUp(self, a0: bool) -> None: ...
    def setActiveAction(self, action: typing.Optional[QAction]) -> None: ...
    def activeAction(self) -> typing.Optional[QAction]: ...
    def clear(self) -> None: ...
    def insertSeparator(self, before: typing.Optional[QAction]) -> typing.Optional[QAction]: ...
    def insertMenu(self, before: typing.Optional[QAction], menu: typing.Optional[QMenu]) -> typing.Optional[QAction]: ...
    def addSeparator(self) -> typing.Optional[QAction]: ...
    @typing.overload
    def addMenu(self, menu: typing.Optional[QMenu]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addMenu(self, title: typing.Optional[str]) -> typing.Optional[QMenu]: ...
    @typing.overload
    def addMenu(self, icon: QtGui.QIcon, title: typing.Optional[str]) -> typing.Optional[QMenu]: ...
    @typing.overload
    def addAction(self, action: typing.Optional[QAction]) -> None: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str], slot: PYQT_SLOT) -> typing.Optional[QAction]: ...


class QMessageBox(QDialog):

    class StandardButton(int):
        NoButton = ... # type: QMessageBox.StandardButton
        Ok = ... # type: QMessageBox.StandardButton
        Save = ... # type: QMessageBox.StandardButton
        SaveAll = ... # type: QMessageBox.StandardButton
        Open = ... # type: QMessageBox.StandardButton
        Yes = ... # type: QMessageBox.StandardButton
        YesToAll = ... # type: QMessageBox.StandardButton
        No = ... # type: QMessageBox.StandardButton
        NoToAll = ... # type: QMessageBox.StandardButton
        Abort = ... # type: QMessageBox.StandardButton
        Retry = ... # type: QMessageBox.StandardButton
        Ignore = ... # type: QMessageBox.StandardButton
        Close = ... # type: QMessageBox.StandardButton
        Cancel = ... # type: QMessageBox.StandardButton
        Discard = ... # type: QMessageBox.StandardButton
        Help = ... # type: QMessageBox.StandardButton
        Apply = ... # type: QMessageBox.StandardButton
        Reset = ... # type: QMessageBox.StandardButton
        RestoreDefaults = ... # type: QMessageBox.StandardButton
        FirstButton = ... # type: QMessageBox.StandardButton
        LastButton = ... # type: QMessageBox.StandardButton
        YesAll = ... # type: QMessageBox.StandardButton
        NoAll = ... # type: QMessageBox.StandardButton
        Default = ... # type: QMessageBox.StandardButton
        Escape = ... # type: QMessageBox.StandardButton
        FlagMask = ... # type: QMessageBox.StandardButton
        ButtonMask = ... # type: QMessageBox.StandardButton

    class Icon(int):
        NoIcon = ... # type: QMessageBox.Icon
        Information = ... # type: QMessageBox.Icon
        Warning = ... # type: QMessageBox.Icon
        Critical = ... # type: QMessageBox.Icon
        Question = ... # type: QMessageBox.Icon

    class ButtonRole(int):
        InvalidRole = ... # type: QMessageBox.ButtonRole
        AcceptRole = ... # type: QMessageBox.ButtonRole
        RejectRole = ... # type: QMessageBox.ButtonRole
        DestructiveRole = ... # type: QMessageBox.ButtonRole
        ActionRole = ... # type: QMessageBox.ButtonRole
        HelpRole = ... # type: QMessageBox.ButtonRole
        YesRole = ... # type: QMessageBox.ButtonRole
        NoRole = ... # type: QMessageBox.ButtonRole
        ResetRole = ... # type: QMessageBox.ButtonRole
        ApplyRole = ... # type: QMessageBox.ButtonRole
    class StandardButtons(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
'QMessageBox.StandardButtons': ...
        def __xor__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
'QMessageBox.StandardButtons': ...
        def __ior__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
'QMessageBox.StandardButtons': ...
        def __or__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
'QMessageBox.StandardButtons': ...
        def __iand__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
'QMessageBox.StandardButtons': ...
        def __and__(self, f: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
'QMessageBox.StandardButtons': ...
        def __invert__(self) -> 'QMessageBox.StandardButtons': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, icon: 'QMessageBox.Icon', title: typing.Optional[str], text: typing.Optional[str], buttons:
typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton'] = ..., parent: typing.Optional[QWidget] = ...,
flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> None: ...

    def checkBox(self) -> typing.Optional[QCheckBox]: ...
    def setCheckBox(self, cb: typing.Optional[QCheckBox]) -> None: ...
    def textInteractionFlags(self) -> QtCore.Qt.TextInteractionFlags: ...
    def setTextInteractionFlags(self, flags: typing.Union[QtCore.Qt.TextInteractionFlags, QtCore.Qt.TextInteractionFlag]) ->
None: ...
    buttonClicked: typing.ClassVar[QtCore.pyqtSignal]
    def buttonRole(self, button: typing.Optional[QAbstractButton]) -> 'QMessageBox.ButtonRole': ...
    def buttons(self) -> typing.List[QAbstractButton]: ...
    @typing.overload
    def open(self) -> None: ...
    @typing.overload
    def open(self, slot: PYQT_SLOT) -> None: ...
    def setWindowModality(self, windowModality: QtCore.Qt.WindowModality) -> None: ...
    def setWindowTitle(self, title: typing.Optional[str]) -> None: ...
    def setDetailedText(self, text: typing.Optional[str]) -> None: ...
    def detailedText(self) -> str: ...
    def setInformativeText(self, text: typing.Optional[str]) -> None: ...
    def informativeText(self) -> str: ...
    def clickedButton(self) -> typing.Optional[QAbstractButton]: ...
    @typing.overload
    def setEscapeButton(self, button: typing.Optional[QAbstractButton]) -> None: ...
    @typing.overload
    def setEscapeButton(self, button: 'QMessageBox.StandardButton') -> None: ...
    def escapeButton(self) -> typing.Optional[QAbstractButton]: ...
    @typing.overload
    def setDefaultButton(self, button: typing.Optional[QPushButton]) -> None: ...
    @typing.overload
    def setDefaultButton(self, button: 'QMessageBox.StandardButton') -> None: ...
    def defaultButton(self) -> typing.Optional[QPushButton]: ...
    def button(self, which: 'QMessageBox.StandardButton') -> typing.Optional[QAbstractButton]: ...
    def standardButton(self, button: typing.Optional[QAbstractButton]) -> 'QMessageBox.StandardButton': ...
    def standardButtons(self) -> 'QMessageBox.StandardButtons': ...
    def setStandardButtons(self, buttons: typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton']) ->
None: ...
    def removeButton(self, button: typing.Optional[QAbstractButton]) -> None: ...
    @typing.overload
    def addButton(self, button: typing.Optional[QAbstractButton], role: 'QMessageBox.ButtonRole') -> None: ...
    @typing.overload
    def addButton(self, text: typing.Optional[str], role: 'QMessageBox.ButtonRole') -> typing.Optional[QPushButton]: ...
    @typing.overload
    def addButton(self, button: 'QMessageBox.StandardButton') -> typing.Optional[QPushButton]: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def closeEvent(self, a0: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    @staticmethod
    def standardIcon(icon: 'QMessageBox.Icon') -> QtGui.QPixmap: ...
    @staticmethod
    def aboutQt(parent: typing.Optional[QWidget], title: typing.Optional[str] = ...) -> None: ...
    @staticmethod
    def about(parent: typing.Optional[QWidget], caption: typing.Optional[str], text: typing.Optional[str]) -> None: ...
    @staticmethod
    def critical(parent: typing.Optional[QWidget], title: typing.Optional[str], text: typing.Optional[str], buttons:
typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton'] = ..., defaultButton:
'QMessageBox.StandardButton' = ...) -> 'QMessageBox.StandardButton': ...
    @staticmethod
    def warning(parent: typing.Optional[QWidget], title: typing.Optional[str], text: typing.Optional[str], buttons:
typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton'] = ..., defaultButton:
'QMessageBox.StandardButton' = ...) -> 'QMessageBox.StandardButton': ...
    @staticmethod
    def question(parent: typing.Optional[QWidget], title: typing.Optional[str], text: typing.Optional[str], buttons:
typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton'] = ..., defaultButton:
'QMessageBox.StandardButton' = ...) -> 'QMessageBox.StandardButton': ...
    @staticmethod
    def information(parent: typing.Optional[QWidget], title: typing.Optional[str], text: typing.Optional[str], buttons:
typing.Union['QMessageBox.StandardButtons', 'QMessageBox.StandardButton'] = ..., defaultButton:
'QMessageBox.StandardButton' = ...) -> 'QMessageBox.StandardButton': ...
    def setTextFormat(self, a0: QtCore.Qt.TextFormat) -> None: ...
    def textFormat(self) -> QtCore.Qt.TextFormat: ...
    def setIconPixmap(self, a0: QtGui.QPixmap) -> None: ...
    def iconPixmap(self) -> QtGui.QPixmap: ...
    def setIcon(self, a0: 'QMessageBox.Icon') -> None: ...
    def icon(self) -> 'QMessageBox.Icon': ...
    def setText(self, a0: typing.Optional[str]) -> None: ...
    def text(self) -> str: ...


class QMouseEventTransition(QtCore.QEventTransition):

    @typing.overload
    def __init__(self, sourceState: typing.Optional[QtCore.QState] = ...) -> None: ...
    @typing.overload
    def __init__(self, object: typing.Optional[QtCore.QObject], type: QtCore.QEvent.Type, button: QtCore.Qt.MouseButton,
sourceState: typing.Optional[QtCore.QState] = ...) -> None: ...

    def eventTest(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def onTransition(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def setHitTestPath(self, path: QtGui.QPainterPath) -> None: ...
    def hitTestPath(self) -> QtGui.QPainterPath: ...
    def setModifierMask(self, modifiers: typing.Union[QtCore.Qt.KeyboardModifiers, QtCore.Qt.KeyboardModifier]) -> None: ...
    def modifierMask(self) -> QtCore.Qt.KeyboardModifiers: ...
    def setButton(self, button: QtCore.Qt.MouseButton) -> None: ...
    def button(self) -> QtCore.Qt.MouseButton: ...


class QOpenGLWidget(QWidget):

    class UpdateBehavior(int):
        NoPartialUpdate = ... # type: QOpenGLWidget.UpdateBehavior
        PartialUpdate = ... # type: QOpenGLWidget.UpdateBehavior

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...
```

```
    def setTextureFormat(self, texFormat: int) -> None: ...
    def textureFormat(self) -> int: ...
    def updateBehavior(self) -> 'QOpenGLWidget.UpdateBehavior': ...
    def setUpdateBehavior(self, updateBehavior: 'QOpenGLWidget.UpdateBehavior') -> None: ...
    def paintEngine(self) -> typing.Optional[QtGui.QPaintEngine]: ...
    def metric(self, metric: QtGui.QPaintDevice.PaintDeviceMetric) -> int: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def paintGL(self) -> None: ...
    def resizeGL(self, w: int, h: int) -> None: ...
    def initializeGL(self) -> None: ...
    resized: typing.ClassVar[QtCore.pyqtSignal]
    aboutToResize: typing.ClassVar[QtCore.pyqtSignal]
    frameSwapped: typing.ClassVar[QtCore.pyqtSignal]
    aboutToCompose: typing.ClassVar[QtCore.pyqtSignal]
    def grabFramebuffer(self) -> QtGui.QImage: ...
    def defaultFramebufferObject(self) -> int: ...
    def context(self) -> typing.Optional[QtGui.QOpenGLContext]: ...
    def doneCurrent(self) -> None: ...
    def makeCurrent(self) -> None: ...
    def isValid(self) -> bool: ...
    def format(self) -> QtGui.QSurfaceFormat: ...
    def setFormat(self, format: QtGui.QSurfaceFormat) -> None: ...


class QPlainTextEdit(QAbstractScrollArea):

    class LineWrapMode(int):
        NoWrap = ... # type: QPlainTextEdit.LineWrapMode
        WidgetWidth = ... # type: QPlainTextEdit.LineWrapMode

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    def setTabStopDistance(self, distance: float) -> None: ...
    def tabStopDistance(self) -> float: ...
    def placeholderText(self) -> str: ...
    def setPlaceholderText(self, placeholderText: typing.Optional[str]) -> None: ...
    def zoomOut(self, range: int = ...) -> None: ...
    def zoomIn(self, range: int = ...) -> None: ...
    def anchorAt(self, pos: QtCore.QPoint) -> str: ...
    def getPaintContext(self) -> QtGui.QAbstractTextDocumentLayout.PaintContext: ...
    def blockBoundingGeometry(self, block: QtGui.QTextBlock) -> QtCore.QRectF: ...
    def blockBoundingRect(self, block: QtGui.QTextBlock) -> QtCore.QRectF: ...
    def contentOffset(self) -> QtCore.QPointF: ...
    def firstVisibleBlock(self) -> QtGui.QTextBlock: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def insertFromMimeData(self, source: typing.Optional[QtCore.QMimeData]) -> None: ...
    def canInsertFromMimeData(self, source: typing.Optional[QtCore.QMimeData]) -> bool: ...
    def createMimeDataFromSelection(self) -> typing.Optional[QtCore.QMimeData]: ...
    @typing.overload
    def inputMethodQuery(self, property: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    @typing.overload
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery, argument: typing.Any) -> typing.Any: ...
    def inputMethodEvent(self, a0: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def focusOutEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def dropEvent(self, e: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragMoveEvent(self, e: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def dragLeaveEvent(self, e: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragEnterEvent(self, e: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
    def contextMenuEvent(self, e: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def mouseDoubleClickEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, e: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def keyReleaseEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def timerEvent(self, e: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    modificationChanged: typing.ClassVar[QtCore.pyqtSignal]
    blockCountChanged: typing.ClassVar[QtCore.pyqtSignal]
    updateRequest: typing.ClassVar[QtCore.pyqtSignal]
    cursorPositionChanged: typing.ClassVar[QtCore.pyqtSignal]
    selectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    copyAvailable: typing.ClassVar[QtCore.pyqtSignal]
    redoAvailable: typing.ClassVar[QtCore.pyqtSignal]
    undoAvailable: typing.ClassVar[QtCore.pyqtSignal]
    textChanged: typing.ClassVar[QtCore.pyqtSignal]
    def centerCursor(self) -> None: ...
    def appendHtml(self, html: typing.Optional[str]) -> None: ...
    def appendPlainText(self, text: typing.Optional[str]) -> None: ...
    def insertPlainText(self, text: typing.Optional[str]) -> None: ...
    def selectAll(self) -> None: ...
    def clear(self) -> None: ...
    def redo(self) -> None: ...
    def undo(self) -> None: ...
    def paste(self) -> None: ...
    def copy(self) -> None: ...
    def cut(self) -> None: ...
    def setPlainText(self, text: typing.Optional[str]) -> None: ...
    def blockCount(self) -> int: ...
    def print(self, printer: typing.Optional[QtGui.QPagedPaintDevice]) -> None: ...
    def print_(self, printer: typing.Optional[QtGui.QPagedPaintDevice]) -> None: ...
    def canPaste(self) -> bool: ...
    def moveCursor(self, operation: QtGui.QTextCursor.MoveOperation, mode: QtGui.QTextCursor.MoveMode = ...) -> None:
...
    def extraSelections(self) -> typing.List['QTextEdit.ExtraSelection']: ...
    def setExtraSelections(self, selections: typing.Iterable['QTextEdit.ExtraSelection']) -> None: ...
    def setCursorWidth(self, width: int) -> None: ...
    def cursorWidth(self) -> int: ...
    def setTabStopWidth(self, width: int) -> None: ...
    def tabStopWidth(self) -> int: ...
    def setOverwriteMode(self, overwrite: bool) -> None: ...
    def overwriteMode(self) -> bool: ...
    @typing.overload
    def cursorRect(self, cursor: QtGui.QTextCursor) -> QtCore.QRect: ...
    @typing.overload
    def cursorRect(self) -> QtCore.QRect: ...
    def cursorForPosition(self, pos: QtCore.QPoint) -> QtGui.QTextCursor: ...
    @typing.overload
    def createStandardContextMenu(self) -> typing.Optional[QMenu]: ...
    @typing.overload
    def createStandardContextMenu(self, position: QtCore.QPoint) -> typing.Optional[QMenu]: ...
    def loadResource(self, type: int, name: QtCore.QUrl) -> typing.Any: ...
    def ensureCursorVisible(self) -> None: ...
    def toPlainText(self) -> str: ...
    @typing.overload
    def find(self, exp: typing.Optional[str], options: typing.Union[QtGui.QTextDocument.FindFlags,
QtGui.QTextDocument.FindFlag] = ...) -> bool: ...
    @typing.overload
    def find(self, exp: QtCore.QRegExp, options: typing.Union[QtGui.QTextDocument.FindFlags,
QtGui.QTextDocument.FindFlag] = ...) -> bool: ...
    @typing.overload
    def find(self, exp: QtCore.QRegularExpression, options: typing.Union[QtGui.QTextDocument.FindFlags,
QtGui.QTextDocument.FindFlag] = ...) -> bool: ...
    def centerOnScroll(self) -> bool: ...
    def setCenterOnScroll(self, enabled: bool) -> None: ...
    def backgroundVisible(self) -> bool: ...
    def setBackgroundVisible(self, visible: bool) -> None: ...
    def setWordWrapMode(self, policy: QtGui.QTextOption.WrapMode) -> None: ...
    def wordWrapMode(self) -> QtGui.QTextOption.WrapMode: ...
    def setLineWrapMode(self, mode: 'QPlainTextEdit.LineWrapMode') -> None: ...
    def lineWrapMode(self) -> 'QPlainTextEdit.LineWrapMode': ...
    def maximumBlockCount(self) -> int: ...
    def setMaximumBlockCount(self, maximum: int) -> None: ...
    def setUndoRedoEnabled(self, enable: bool) -> None: ...
    def isUndoRedoEnabled(self) -> bool: ...
    def documentTitle(self) -> str: ...
    def setDocumentTitle(self, title: typing.Optional[str]) -> None: ...
    def setTabChangesFocus(self, b: bool) -> None: ...
    def tabChangesFocus(self) -> bool: ...
    def currentCharFormat(self) -> QtGui.QTextCharFormat: ...
    def setCurrentCharFormat(self, format: QtGui.QTextCharFormat) -> None: ...
    def mergeCurrentCharFormat(self, modifier: QtGui.QTextCharFormat) -> None: ...
    def textInteractionFlags(self) -> QtCore.Qt.TextInteractionFlags: ...
    def setTextInteractionFlags(self, flags: typing.Union[QtCore.Qt.TextInteractionFlags, QtCore.Qt.TextInteractionFlag]) ->
None: ...
    def setReadOnly(self, ro: bool) -> None: ...
    def isReadOnly(self) -> bool: ...
    def textCursor(self) -> QtGui.QTextCursor: ...
    def setTextCursor(self, cursor: QtGui.QTextCursor) -> None: ...
    def document(self) -> typing.Optional[QtGui.QTextDocument]: ...
    def setDocument(self, document: typing.Optional[QtGui.QTextDocument]) -> None: ...


class QPlainTextDocumentLayout(QtGui.QAbstractTextDocumentLayout):

    def __init__(self, document: typing.Optional[QtGui.QTextDocument]) -> None: ...

    def documentChanged(self, from_: int, a1: int, charsAdded: int) -> None: ...
    def requestUpdate(self) -> None: ...
    def cursorWidth(self) -> int: ...
    def setCursorWidth(self, width: int) -> None: ...
    def ensureBlockLayout(self, block: QtGui.QTextBlock) -> None: ...
    def blockBoundingRect(self, block: QtGui.QTextBlock) -> QtCore.QRectF: ...
    def frameBoundingRect(self, a0: typing.Optional[QtGui.QTextFrame]) -> QtCore.QRectF: ...
    def documentSize(self) -> QtCore.QSizeF: ...
    def pageCount(self) -> int: ...
    def hitTest(self, a0: typing.Union[QtCore.QPointF, QtCore.QPoint], a1: QtCore.Qt.HitTestAccuracy) -> int: ...
    def draw(self, a0: typing.Optional[QtGui.QPainter], a1: QtGui.QAbstractTextDocumentLayout.PaintContext) -> None: ...


class QProgressBar(QWidget):

    class Direction(int):
        TopToBottom = ... # type: QProgressBar.Direction
        BottomToTop = ... # type: QProgressBar.Direction

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionProgressBar']) -> None: ...
    valueChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setOrientation(self, a0: QtCore.Qt.Orientation) -> None: ...
    def setValue(self, value: int) -> None: ...
    def setMaximum(self, maximum: int) -> None: ...
    def setMinimum(self, minimum: int) -> None: ...
    def reset(self) -> None: ...
    def resetFormat(self) -> None: ...
    def format(self) -> str: ...
    def setFormat(self, format: typing.Optional[str]) -> None: ...
    def setTextDirection(self, textDirection: 'QProgressBar.Direction') -> None: ...
    def setInvertedAppearance(self, invert: bool) -> None: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setAlignment(self, alignment: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def isTextVisible(self) -> bool: ...
    def setTextVisible(self, visible: bool) -> None: ...
    def text(self) -> str: ...
    def value(self) -> int: ...
    def setRange(self, minimum: int, maximum: int) -> None: ...
    def maximum(self) -> int: ...
    def minimum(self) -> int: ...


class QProgressDialog(QDialog):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...
    @typing.overload
    def __init__(self, labelText: typing.Optional[str], cancelButtonText: typing.Optional[str], minimum: int, maximum: int,
parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> None:
...

    @typing.overload
    def open(self) -> None: ...
    @typing.overload
    def open(self, slot: PYQT_SLOT) -> None: ...
    def forceShow(self) -> None: ...
    def showEvent(self, e: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def closeEvent(self, a0: typing.Optional[QtGui.QCloseEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    canceled: typing.ClassVar[QtCore.pyqtSignal]
    def setMinimumDuration(self, ms: int) -> None: ...
    def setCancelButtonText(self, a0: typing.Optional[str]) -> None: ...
    def setLabelText(self, a0: typing.Optional[str]) -> None: ...
    def setValue(self, progress: int) -> None: ...
    def setMinimum(self, minimum: int) -> None: ...
    def setMaximum(self, maximum: int) -> None: ...
    def reset(self) -> None: ...
    def cancel(self) -> None: ...
    def autoClose(self) -> bool: ...
    def setAutoClose(self, b: bool) -> None: ...
    def autoReset(self) -> bool: ...
    def setAutoReset(self, b: bool) -> None: ...
    def minimumDuration(self) -> int: ...
    def labelText(self) -> str: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def value(self) -> int: ...
    def setRange(self, minimum: int, maximum: int) -> None: ...
    def maximum(self) -> int: ...
    def minimum(self) -> int: ...
    def wasCanceled(self) -> bool: ...
    def setBar(self, bar: typing.Optional[QProgressBar]) -> None: ...
    def setCancelButton(self, button: typing.Optional[QPushButton]) -> None: ...
    def setLabel(self, label: typing.Optional[QLabel]) -> None: ...


class QProxyStyle(QCommonStyle):

    @typing.overload
    def __init__(self, style: typing.Optional[QStyle] = ...) -> None: ...
    @typing.overload
    def __init__(self, key: typing.Optional[str]) -> None: ...

    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    @typing.overload
    def unpolish(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def unpolish(self, app: typing.Optional[QApplication]) -> None: ...
    @typing.overload
    def polish(self, widget: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def polish(self, pal: QtGui.QPalette) -> QtGui.QPalette: ...
    @typing.overload
    def polish(self, app: typing.Optional[QApplication]) -> None: ...
    def standardPalette(self) -> QtGui.QPalette: ...
    def generatedIconPixmap(self, iconMode: QtGui.QIcon.Mode, pixmap: QtGui.QPixmap, opt:
typing.Optional['QStyleOption']) -> QtGui.QPixmap: ...
    def standardPixmap(self, standardPixmap: QStyle.StandardPixmap, opt: typing.Optional['QStyleOption'], widget:
typing.Optional[QWidget] = ...) -> QtGui.QPixmap: ...
    def standardIcon(self, standardIcon: QStyle.StandardPixmap, option: typing.Optional['QStyleOption'] = ..., widget:
typing.Optional[QWidget] = ...) -> QtGui.QIcon: ...
    def layoutSpacing(self, control1: 'QSizePolicy.ControlType', control2: 'QSizePolicy.ControlType', orientation:
QtCore.Qt.Orientation, option: typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] = ...) -> int: ...
    def pixelMetric(self, metric: QStyle.PixelMetric, option: typing.Optional['QStyleOption'] = ..., widget:
typing.Optional[QWidget] = ...) -> int: ...
    def styleHint(self, hint: QStyle.StyleHint, option: typing.Optional['QStyleOption'] = ..., widget: typing.Optional[QWidget] =
..., returnData: typing.Optional['QStyleHintReturn'] = ...) -> int: ...
    def hitTestComplexControl(self, control: QStyle.ComplexControl, option: typing.Optional['QStyleOptionComplex'], pos:
QtCore.QPoint, widget: typing.Optional[QWidget] = ...) -> QStyle.SubControl: ...
    def itemPixmapRect(self, r: QtCore.QRect, flags: int, pixmap: QtGui.QPixmap) -> QtCore.QRect: ...
    def itemTextRect(self, fm: QtGui.QFontMetrics, r: QtCore.QRect, flags: int, enabled: bool, text: typing.Optional[str]) ->
QtCore.QRect: ...
    def subControlRect(self, cc: QStyle.ComplexControl, opt: typing.Optional['QStyleOptionComplex'], sc: QStyle.SubControl,
widget: typing.Optional[QWidget]) -> QtCore.QRect: ...
    def subElementRect(self, element: QStyle.SubElement, option: typing.Optional['QStyleOption'], widget:
typing.Optional[QWidget]) -> QtCore.QRect: ...
    def sizeFromContents(self, type: QStyle.ContentsType, option: typing.Optional['QStyleOption'], size: QtCore.QSize, widget:
typing.Optional[QWidget]) -> QtCore.QSize: ...
    def drawItemPixmap(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRect, alignment: int, pixmap:
QtGui.QPixmap) -> None: ...
    def drawItemText(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRect, flags: int, pal: QtGui.QPalette,
enabled: bool, text: typing.Optional[str], textRole: QtGui.QPalette.ColorRole = ...) -> None: ...
    def drawComplexControl(self, control: QStyle.ComplexControl, option: typing.Optional['QStyleOptionComplex'], painter:
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def drawControl(self, element: QStyle.ControlElement, option: typing.Optional['QStyleOption'], painter:
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def drawPrimitive(self, element: QStyle.PrimitiveElement, option: typing.Optional['QStyleOption'], painter:
typing.Optional[QtGui.QPainter], widget: typing.Optional[QWidget] = ...) -> None: ...
    def setBaseStyle(self, style: typing.Optional[QStyle]) -> None: ...
    def baseStyle(self) -> typing.Optional[QStyle]: ...


class QRadioButton(QAbstractButton):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def hitButton(self, a0: QtCore.QPoint) -> bool: ...
    def initStyleOption(self, button: typing.Optional['QStyleOptionButton']) -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QRubberBand(QWidget):

    class Shape(int):
        Line = ... # type: QRubberBand.Shape
        Rectangle = ... # type: QRubberBand.Shape

    def __init__(self, a0: 'QRubberBand.Shape', parent: typing.Optional[QWidget] = ...) -> None: ...

    def moveEvent(self, a0: typing.Optional[QtGui.QMoveEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionRubberBand']) -> None: ...
    @typing.overload
    def resize(self, w: int, h: int) -> None: ...
    @typing.overload
    def resize(self, s: QtCore.QSize) -> None: ...
    @typing.overload
    def move(self, p: QtCore.QPoint) -> None: ...
    @typing.overload
    def move(self, ax: int, ay: int) -> None: ...
    @typing.overload
    def setGeometry(self, r: QtCore.QRect) -> None: ...
    @typing.overload
    def setGeometry(self, ax: int, ay: int, aw: int, ah: int) -> None: ...
    def shape(self) -> 'QRubberBand.Shape': ...


class QScrollArea(QAbstractScrollArea):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def viewportSizeHint(self) -> QtCore.QSize: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def ensureWidgetVisible(self, childWidget: typing.Optional[QWidget], xMargin: int = ..., yMargin: int = ...) -> None: ...
    def ensureVisible(self, x: int, y: int, xMargin: int = ..., yMargin: int = ...) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setAlignment(self, a0: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def setWidgetResizable(self, resizable: bool) -> None: ...
    def widgetResizable(self) -> bool: ...
    def takeWidget(self) -> typing.Optional[QWidget]: ...
    def setWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def widget(self) -> typing.Optional[QWidget]: ...


class QScrollBar(QAbstractSlider):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, orientation: QtCore.Qt.Orientation, parent: typing.Optional[QWidget] = ...) -> None: ...

    def sliderChange(self, change: QAbstractSlider.SliderChange) -> None: ...
    def wheelEvent(self, a0: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def contextMenuEvent(self, a0: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def hideEvent(self, a0: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionSlider']) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QScroller(QtCore.QObject):

    class Input(int):
        InputPress = ... # type: QScroller.Input
        InputMove = ... # type: QScroller.Input
        InputRelease = ... # type: QScroller.Input

    class ScrollerGestureType(int):
        TouchGesture = ... # type: QScroller.ScrollerGestureType
        LeftMouseButtonGesture = ... # type: QScroller.ScrollerGestureType
        RightMouseButtonGesture = ... # type: QScroller.ScrollerGestureType
        MiddleMouseButtonGesture = ... # type: QScroller.ScrollerGestureType
    class State(int):
        Inactive = ... # type: QScroller.State
        Pressed = ... # type: QScroller.State
        Dragging = ... # type: QScroller.State
        Scrolling = ... # type: QScroller.State

    scrollerPropertiesChanged: typing.ClassVar[QtCore.pyqtSignal]
    stateChanged: typing.ClassVar[QtCore.pyqtSignal]
    def resendPrepareEvent(self) -> None: ...
    @typing.overload
    def ensureVisible(self, rect: QtCore.QRectF, xmargin: float, ymargin: float) -> None: ...
    @typing.overload
    def ensureVisible(self, rect: QtCore.QRectF, xmargin: float, ymargin: float, scrollTime: int) -> None: ...
    @typing.overload
    def scrollTo(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint]) -> None: ...
    @typing.overload
    def scrollTo(self, pos: typing.Union[QtCore.QPointF, QtCore.QPoint], scrollTime: int) -> None: ...
    def setScrollerProperties(self, prop: 'QScrollerProperties') -> None: ...
    @typing.overload
    def setSnapPositionsY(self, positions: typing.Iterable[float]) -> None: ...
    @typing.overload
    def setSnapPositionsY(self, first: float, interval: float) -> None: ...
    @typing.overload
    def setSnapPositionsX(self, positions: typing.Iterable[float]) -> None: ...
    @typing.overload
    def setSnapPositionsX(self, first: float, interval: float) -> None: ...
    def scrollerProperties(self) -> 'QScrollerProperties': ...
    def pixelPerMeter(self) -> QtCore.QPointF: ...
    def finalPosition(self) -> QtCore.QPointF: ...
    def velocity(self) -> QtCore.QPointF: ...
    def stop(self) -> None: ...
    def handleInput(self, input: 'QScroller.Input', position: typing.Union[QtCore.QPointF, QtCore.QPoint], timestamp: int = ...)
-> bool: ...
    def state(self) -> 'QScroller.State': ...
    def target(self) -> typing.Optional[QtCore.QObject]: ...
    @staticmethod
    def activeScrollers() -> typing.List['QScroller']: ...
    @staticmethod
    def ungrabGesture(target: typing.Optional[QtCore.QObject]) -> None: ...
    @staticmethod
    def grabbedGesture(target: typing.Optional[QtCore.QObject]) -> QtCore.Qt.GestureType: ...
    @staticmethod
    def grabGesture(target: typing.Optional[QtCore.QObject], scrollGestureType: 'QScroller.ScrollerGestureType' = ...) ->
QtCore.Qt.GestureType: ...
    @staticmethod
    def scroller(target: typing.Optional[QtCore.QObject]) -> typing.Optional['QScroller']: ...
    @staticmethod
    def hasScroller(target: typing.Optional[QtCore.QObject]) -> bool: ...


class QScrollerProperties(PyQt5.sipsimplewrapper):

    class ScrollMetric(int):
        MousePressEventDelay = ... # type: QScrollerProperties.ScrollMetric
        DragStartDistance = ... # type: QScrollerProperties.ScrollMetric
        DragVelocitySmoothingFactor = ... # type: QScrollerProperties.ScrollMetric
        AxisLockThreshold = ... # type: QScrollerProperties.ScrollMetric
        ScrollingCurve = ... # type: QScrollerProperties.ScrollMetric
        DecelerationFactor = ... # type: QScrollerProperties.ScrollMetric
        MinimumVelocity = ... # type: QScrollerProperties.ScrollMetric
        MaximumVelocity = ... # type: QScrollerProperties.ScrollMetric
        MaximumClickThroughVelocity = ... # type: QScrollerProperties.ScrollMetric
        AcceleratingFlickMaximumTime = ... # type: QScrollerProperties.ScrollMetric
        AcceleratingFlickSpeedupFactor = ... # type: QScrollerProperties.ScrollMetric
        SnapPositionRatio = ... # type: QScrollerProperties.ScrollMetric
        SnapTime = ... # type: QScrollerProperties.ScrollMetric
        OvershootDragResistanceFactor = ... # type: QScrollerProperties.ScrollMetric
        OvershootDragDistanceFactor = ... # type: QScrollerProperties.ScrollMetric
        OvershootScrollDistanceFactor = ... # type: QScrollerProperties.ScrollMetric
        OvershootScrollTime = ... # type: QScrollerProperties.ScrollMetric
        HorizontalOvershootPolicy = ... # type: QScrollerProperties.ScrollMetric
        VerticalOvershootPolicy = ... # type: QScrollerProperties.ScrollMetric
        FrameRate = ... # type: QScrollerProperties.ScrollMetric
        ScrollMetricCount = ... # type: QScrollerProperties.ScrollMetric

    class FrameRates(int):
        Standard = ... # type: QScrollerProperties.FrameRates
        Fps60 = ... # type: QScrollerProperties.FrameRates
        Fps30 = ... # type: QScrollerProperties.FrameRates
        Fps20 = ... # type: QScrollerProperties.FrameRates

    class OvershootPolicy(int):
        OvershootWhenScrollable = ... # type: QScrollerProperties.OvershootPolicy
        OvershootAlwaysOff = ... # type: QScrollerProperties.OvershootPolicy
        OvershootAlwaysOn = ... # type: QScrollerProperties.OvershootPolicy

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, sp: 'QScrollerProperties') -> None: ...

    def setScrollMetric(self, metric: 'QScrollerProperties.ScrollMetric', value: typing.Any) -> None: ...
    def scrollMetric(self, metric: 'QScrollerProperties.ScrollMetric') -> typing.Any: ...
    @staticmethod
    def unsetDefaultScrollerProperties() -> None: ...
    @staticmethod
    def setDefaultScrollerProperties(sp: 'QScrollerProperties') -> None: ...
    def __ne__(self, other: object): ...
    def __eq__(self, other: object): ...


class QShortcut(QtCore.QObject):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def __init__(self, key: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, typing.Optional[str], int],
parent: typing.Optional[QWidget], member: PYQT_SLOT = ..., ambiguousMember: PYQT_SLOT = ..., context:
QtCore.Qt.ShortcutContext = ...) -> None: ...

    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    activatedAmbiguously: typing.ClassVar[QtCore.pyqtSignal]
    activated: typing.ClassVar[QtCore.pyqtSignal]
    def autoRepeat(self) -> bool: ...
    def setAutoRepeat(self, on: bool) -> None: ...
    def parentWidget(self) -> typing.Optional[QWidget]: ...
    def id(self) -> int: ...
    def whatsThis(self) -> str: ...
    def setWhatsThis(self, text: typing.Optional[str]) -> None: ...
    def context(self) -> QtCore.Qt.ShortcutContext: ...
    def setContext(self, context: QtCore.Qt.ShortcutContext) -> None: ...
    def isEnabled(self) -> bool: ...
    def setEnabled(self, enable: bool) -> None: ...
    def key(self) -> QtGui.QKeySequence: ...
    def setKey(self, key: typing.Union[QtGui.QKeySequence, QtGui.QKeySequence.StandardKey, typing.Optional[str], int]) ->
None: ...


class QSizeGrip(QWidget):

    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...

    def hideEvent(self, hideEvent: typing.Optional[QtGui.QHideEvent]) -> None: ...
    def showEvent(self, showEvent: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def moveEvent(self, moveEvent: typing.Optional[QtGui.QMoveEvent]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, mouseEvent: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def setVisible(self, a0: bool) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QSizePolicy(PyQt5.sipsimplewrapper):

    class ControlType(int):
        DefaultType = ... # type: QSizePolicy.ControlType
        ButtonBox = ... # type: QSizePolicy.ControlType
        CheckBox = ... # type: QSizePolicy.ControlType
        ComboBox = ... # type: QSizePolicy.ControlType
        Frame = ... # type: QSizePolicy.ControlType
        GroupBox = ... # type: QSizePolicy.ControlType
        Label = ... # type: QSizePolicy.ControlType
        Line = ... # type: QSizePolicy.ControlType
        LineEdit = ... # type: QSizePolicy.ControlType
        PushButton = ... # type: QSizePolicy.ControlType
        RadioButton = ... # type: QSizePolicy.ControlType
        Slider = ... # type: QSizePolicy.ControlType
        SpinBox = ... # type: QSizePolicy.ControlType
        TabWidget = ... # type: QSizePolicy.ControlType
        ToolButton = ... # type: QSizePolicy.ControlType

    class Policy(int):
        Fixed = ... # type: QSizePolicy.Policy
        Minimum = ... # type: QSizePolicy.Policy
        Maximum = ... # type: QSizePolicy.Policy
        Preferred = ... # type: QSizePolicy.Policy
        MinimumExpanding = ... # type: QSizePolicy.Policy
        Expanding = ... # type: QSizePolicy.Policy
        Ignored = ... # type: QSizePolicy.Policy

    class PolicyFlag(int):
        GrowFlag = ... # type: QSizePolicy.PolicyFlag
        ExpandFlag = ... # type: QSizePolicy.PolicyFlag
        ShrinkFlag = ... # type: QSizePolicy.PolicyFlag
        IgnoreFlag = ... # type: QSizePolicy.PolicyFlag

    class ControlTypes(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> 'QSizePolicy.ControlTypes':
...
        def __xor__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> 'QSizePolicy.ControlTypes':
...
        def __ior__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> 'QSizePolicy.ControlTypes': ...
        def __or__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> 'QSizePolicy.ControlTypes': ...
        def __iand__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> 'QSizePolicy.ControlTypes':
...
        def __and__(self, f: typing.Union['QSizePolicy.ControlTypes', 'QSizePolicy.ControlType']) -> 'QSizePolicy.ControlTypes':
...
        def __invert__(self) -> 'QSizePolicy.ControlTypes': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, horizontal: 'QSizePolicy.Policy', vertical: 'QSizePolicy.Policy', type: 'QSizePolicy.ControlType' = ...) ->
None: ...
    @typing.overload
    def __init__(self, variant: typing.Any) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QSizePolicy') -> None: ...

    def __hash__(self) -> int: ...
    def setRetainSizeWhenHidden(self, retainSize: bool) -> None: ...
    def retainSizeWhenHidden(self) -> bool: ...
    def hasWidthForHeight(self) -> bool: ...
    def setWidthForHeight(self, b: bool) -> None: ...
    def setControlType(self, type: 'QSizePolicy.ControlType') -> None: ...
    def controlType(self) -> 'QSizePolicy.ControlType': ...
    def transposed(self) -> 'QSizePolicy': ...
    def transpose(self) -> None: ...
    def setVerticalStretch(self, stretchFactor: int) -> None: ...
    def setHorizontalStretch(self, stretchFactor: int) -> None: ...
    def verticalStretch(self) -> int: ...
    def horizontalStretch(self) -> int: ...
    def __ne__(self, other: object): ...
    def __eq__(self, other: object): ...
    def hasHeightForWidth(self) -> bool: ...
    def setHeightForWidth(self, b: bool) -> None: ...
    def expandingDirections(self) -> QtCore.Qt.Orientations: ...
    def setVerticalPolicy(self, d: 'QSizePolicy.Policy') -> None: ...
    def setHorizontalPolicy(self, d: 'QSizePolicy.Policy') -> None: ...
    def verticalPolicy(self) -> 'QSizePolicy.Policy': ...
    def horizontalPolicy(self) -> 'QSizePolicy.Policy': ...


class QSlider(QAbstractSlider):

    class TickPosition(int):
        NoTicks = ... # type: QSlider.TickPosition
        TicksAbove = ... # type: QSlider.TickPosition
        TicksLeft = ... # type: QSlider.TickPosition
        TicksBelow = ... # type: QSlider.TickPosition
        TicksRight = ... # type: QSlider.TickPosition
        TicksBothSides = ... # type: QSlider.TickPosition

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, orientation: QtCore.Qt.Orientation, parent: typing.Optional[QWidget] = ...) -> None: ...

    def mouseMoveEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, ev: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionSlider']) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def tickInterval(self) -> int: ...
    def setTickInterval(self, ti: int) -> None: ...
    def tickPosition(self) -> 'QSlider.TickPosition': ...
    def setTickPosition(self, position: 'QSlider.TickPosition') -> None: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QSpinBox(QAbstractSpinBox):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def setStepType(self, stepType: QAbstractSpinBox.StepType) -> None: ...
    def stepType(self) -> QAbstractSpinBox.StepType: ...
    def setDisplayIntegerBase(self, base: int) -> None: ...
    def displayIntegerBase(self) -> int: ...
    textChanged: typing.ClassVar[QtCore.pyqtSignal]
    valueChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setValue(self, val: int) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def fixup(self, str: typing.Optional[str]) -> str: ...
    def textFromValue(self, v: int) -> str: ...
    def valueFromText(self, text: typing.Optional[str]) -> int: ...
    def validate(self, input: typing.Optional[str], pos: int) -> typing.Tuple[QtGui.QValidator.State, str, int]: ...
    def setRange(self, min: int, max: int) -> None: ...
    def setMaximum(self, max: int) -> None: ...
    def maximum(self) -> int: ...
    def setMinimum(self, min: int) -> None: ...
    def minimum(self) -> int: ...
    def setSingleStep(self, val: int) -> None: ...
    def singleStep(self) -> int: ...
    def cleanText(self) -> str: ...
    def setSuffix(self, s: typing.Optional[str]) -> None: ...
    def suffix(self) -> str: ...
    def setPrefix(self, p: typing.Optional[str]) -> None: ...
    def prefix(self) -> str: ...
    def value(self) -> int: ...


class QDoubleSpinBox(QAbstractSpinBox):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def setStepType(self, stepType: QAbstractSpinBox.StepType) -> None: ...
    def stepType(self) -> QAbstractSpinBox.StepType: ...
    textChanged: typing.ClassVar[QtCore.pyqtSignal]
    valueChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setValue(self, val: float) -> None: ...
    def fixup(self, str: typing.Optional[str]) -> str: ...
    def textFromValue(self, v: float) -> str: ...
    def valueFromText(self, text: typing.Optional[str]) -> float: ...
    def validate(self, input: typing.Optional[str], pos: int) -> typing.Tuple[QtGui.QValidator.State, str, int]: ...
    def setDecimals(self, prec: int) -> None: ...
    def decimals(self) -> int: ...
    def setRange(self, min: float, max: float) -> None: ...
    def setMaximum(self, max: float) -> None: ...
    def maximum(self) -> float: ...
    def setMinimum(self, min: float) -> None: ...
    def minimum(self) -> float: ...
    def setSingleStep(self, val: float) -> None: ...
    def singleStep(self) -> float: ...
    def cleanText(self) -> str: ...
    def setSuffix(self, s: typing.Optional[str]) -> None: ...
    def suffix(self) -> str: ...
    def setPrefix(self, p: typing.Optional[str]) -> None: ...
    def prefix(self) -> str: ...
    def value(self) -> float: ...


class QSplashScreen(QWidget):

    @typing.overload
    def __init__(self, pixmap: QtGui.QPixmap = ..., flags: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...)
-> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget], pixmap: QtGui.QPixmap = ..., flags:
typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> None: ...
    @typing.overload
    def __init__(self, screen: typing.Optional[QtGui.QScreen], pixmap: QtGui.QPixmap = ..., flags:
typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType] = ...) -> None: ...

    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def drawContents(self, painter: typing.Optional[QtGui.QPainter]) -> None: ...
    messageChanged: typing.ClassVar[QtCore.pyqtSignal]
    def clearMessage(self) -> None: ...
    def showMessage(self, message: typing.Optional[str], alignment: int = ..., color: typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor] = ...) -> None: ...
    def message(self) -> str: ...
    def repaint(self) -> None: ...
    def finish(self, w: typing.Optional[QWidget]) -> None: ...
    def pixmap(self) -> QtGui.QPixmap: ...
    def setPixmap(self, pixmap: QtGui.QPixmap) -> None: ...


class QSplitter(QFrame):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, orientation: QtCore.Qt.Orientation, parent: typing.Optional[QWidget] = ...) -> None: ...

    def closestLegalPosition(self, a0: int, a1: int) -> int: ...
    def setRubberBand(self, position: int) -> None: ...
    def moveSplitter(self, pos: int, index: int) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def childEvent(self, a0: typing.Optional[QtCore.QChildEvent]) -> None: ...
    def createHandle(self) -> typing.Optional['QSplitterHandle']: ...
    splitterMoved: typing.ClassVar[QtCore.pyqtSignal]
    def replaceWidget(self, index: int, widget: typing.Optional[QWidget]) -> typing.Optional[QWidget]: ...
    def setStretchFactor(self, index: int, stretch: int) -> None: ...
    def handle(self, index: int) -> typing.Optional['QSplitterHandle']: ...
    def getRange(self, index: int) -> typing.Tuple[typing.Optional[int], typing.Optional[int]]: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def widget(self, index: int) -> typing.Optional[QWidget]: ...
    def indexOf(self, w: typing.Optional[QWidget]) -> int: ...
    def setHandleWidth(self, a0: int) -> None: ...
    def handleWidth(self) -> int: ...
    def restoreState(self, state: typing.Union[QtCore.QByteArray, bytes, bytearray]) -> bool: ...
    def saveState(self) -> QtCore.QByteArray: ...
    def setSizes(self, list: typing.Iterable[int]) -> None: ...
    def sizes(self) -> typing.List[int]: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def refresh(self) -> None: ...
    def opaqueResize(self) -> bool: ...
    def setOpaqueResize(self, opaque: bool = ...) -> None: ...
    def isCollapsible(self, index: int) -> bool: ...
    def setCollapsible(self, index: int, a1: bool) -> None: ...
    def childrenCollapsible(self) -> bool: ...
    def setChildrenCollapsible(self, a0: bool) -> None: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setOrientation(self, a0: QtCore.Qt.Orientation) -> None: ...
    def insertWidget(self, index: int, widget: typing.Optional[QWidget]) -> None: ...
    def addWidget(self, widget: typing.Optional[QWidget]) -> None: ...


class QSplitterHandle(QWidget):

    def __init__(self, o: QtCore.Qt.Orientation, parent: typing.Optional[QSplitter]) -> None: ...

    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def closestLegalPosition(self, p: int) -> int: ...
    def moveSplitter(self, p: int) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def splitter(self) -> typing.Optional[QSplitter]: ...
    def opaqueResize(self) -> bool: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setOrientation(self, o: QtCore.Qt.Orientation) -> None: ...


class QStackedLayout(QLayout):
    class StackingMode(int):
        StackOne = ... # type: QStackedLayout.StackingMode
        StackAll = ... # type: QStackedLayout.StackingMode

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def __init__(self, parentLayout: typing.Optional[QLayout]) -> None: ...

    def heightForWidth(self, width: int) -> int: ...
    def hasHeightForWidth(self) -> bool: ...
    def setStackingMode(self, stackingMode: 'QStackedLayout.StackingMode') -> None: ...
    def stackingMode(self) -> 'QStackedLayout.StackingMode': ...
    def setCurrentWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def setCurrentIndex(self, index: int) -> None: ...
    currentChanged: typing.ClassVar[QtCore.pyqtSignal]
    widgetRemoved: typing.ClassVar[QtCore.pyqtSignal]
    def setGeometry(self, rect: QtCore.QRect) -> None: ...
    def takeAt(self, a0: int) -> typing.Optional[QLayoutItem]: ...
    def itemAt(self, a0: int) -> typing.Optional[QLayoutItem]: ...
    def minimumSize(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def addItem(self, item: typing.Optional[QLayoutItem]) -> None: ...
    def count(self) -> int: ...
    @typing.overload
    def widget(self, a0: int) -> typing.Optional[QWidget]: ...
    @typing.overload
    def widget(self) -> typing.Optional[QWidget]: ...
    def currentIndex(self) -> int: ...
    def currentWidget(self) -> typing.Optional[QWidget]: ...
    def insertWidget(self, index: int, w: typing.Optional[QWidget]) -> int: ...
    def addWidget(self, w: typing.Optional[QWidget]) -> int: ...


class QStackedWidget(QFrame):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    widgetRemoved: typing.ClassVar[QtCore.pyqtSignal]
    currentChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setCurrentWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def setCurrentIndex(self, index: int) -> None: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def widget(self, a0: int) -> typing.Optional[QWidget]: ...
    def indexOf(self, a0: typing.Optional[QWidget]) -> int: ...
    def currentIndex(self) -> int: ...
    def currentWidget(self) -> typing.Optional[QWidget]: ...
    def removeWidget(self, w: typing.Optional[QWidget]) -> None: ...
    def insertWidget(self, index: int, w: typing.Optional[QWidget]) -> int: ...
    def addWidget(self, w: typing.Optional[QWidget]) -> int: ...


class QStatusBar(QWidget):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def hideOrShow(self) -> None: ...
    def reformat(self) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    messageChanged: typing.ClassVar[QtCore.pyqtSignal]
    def clearMessage(self) -> None: ...
    def showMessage(self, message: typing.Optional[str], msecs: int = ...) -> None: ...
    def insertPermanentWidget(self, index: int, widget: typing.Optional[QWidget], stretch: int = ...) -> int: ...
    def insertWidget(self, index: int, widget: typing.Optional[QWidget], stretch: int = ...) -> int: ...
    def currentMessage(self) -> str: ...
    def isSizeGripEnabled(self) -> bool: ...
    def setSizeGripEnabled(self, a0: bool) -> None: ...
    def removeWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def addPermanentWidget(self, widget: typing.Optional[QWidget], stretch: int = ...) -> None: ...
    def addWidget(self, widget: typing.Optional[QWidget], stretch: int = ...) -> None: ...


class QStyledItemDelegate(QAbstractItemDelegate):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def editorEvent(self, event: typing.Optional[QtCore.QEvent], model: typing.Optional[QtCore.QAbstractItemModel], option: 
'QStyleOptionViewItem', index: QtCore.QModelIndex) -> bool: ...
    def eventFilter(self, object: typing.Optional[QtCore.QObject], event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional['QStyleOptionViewItem'], index: QtCore.QModelIndex) -> None: ...
    def displayText(self, value: typing.Any, locale: QtCore.QLocale) -> str: ...
    def setItemEditorFactory(self, factory: typing.Optional[QItemEditorFactory]) -> None: ...
    def itemEditorFactory(self) -> typing.Optional[QItemEditorFactory]: ...
    def updateEditorGeometry(self, editor: typing.Optional[QWidget], option: 'QStyleOptionViewItem', index: 
QtCore.QModelIndex) -> None: ...
    def setModelData(self, editor: typing.Optional[QWidget], model: typing.Optional[QtCore.QAbstractItemModel], index: 
QtCore.QModelIndex) -> None: ...
    def setEditorData(self, editor: typing.Optional[QWidget], index: QtCore.QModelIndex) -> None: ...
    def createEditor(self, parent: typing.Optional[QWidget], option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> 
typing.Optional[QWidget]: ...
    def sizeHint(self, option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> QtCore.QSize: ...
    def paint(self, painter: typing.Optional[QtGui.QPainter], option: 'QStyleOptionViewItem', index: QtCore.QModelIndex) -> 
None: ...


class QStyleFactory(PyQt5.sipsimplewrapper):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QStyleFactory') -> None: ...

    @staticmethod
    def create(a0: typing.Optional[str]) -> typing.Optional[QStyle]: ...
    @staticmethod
    def keys() -> typing.List[str]: ...


class QStyleOption(PyQt5.sipsimplewrapper):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOption.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOption.StyleOptionType

    class OptionType(int):
        SO_Default = ... # type: QStyleOption.OptionType
        SO_FocusRect = ... # type: QStyleOption.OptionType
        SO_Button = ... # type: QStyleOption.OptionType
        SO_Tab = ... # type: QStyleOption.OptionType
        SO_MenuItem = ... # type: QStyleOption.OptionType
        SO_Frame = ... # type: QStyleOption.OptionType
        SO_ProgressBar = ... # type: QStyleOption.OptionType
        SO_ToolBox = ... # type: QStyleOption.OptionType
        SO_Header = ... # type: QStyleOption.OptionType
        SO_DockWidget = ... # type: QStyleOption.OptionType
        SO_ViewItem = ... # type: QStyleOption.OptionType
        SO_TabWidgetFrame = ... # type: QStyleOption.OptionType
        SO_TabBarBase = ... # type: QStyleOption.OptionType
        SO_RubberBand = ... # type: QStyleOption.OptionType
        SO_ToolBar = ... # type: QStyleOption.OptionType
        SO_Complex = ... # type: QStyleOption.OptionType
        SO_Slider = ... # type: QStyleOption.OptionType
        SO_SpinBox = ... # type: QStyleOption.OptionType
        SO_ToolButton = ... # type: QStyleOption.OptionType
        SO_ComboBox = ... # type: QStyleOption.OptionType
        SO_TitleBar = ... # type: QStyleOption.OptionType
        SO_GroupBox = ... # type: QStyleOption.OptionType
        SO_ComplexCustomBase = ... # type: QStyleOption.OptionType
        SO_GraphicsItem = ... # type: QStyleOption.OptionType
        SO_SizeGrip = ... # type: QStyleOption.OptionType
        SO_CustomBase = ... # type: QStyleOption.OptionType

    direction = ... # type: QtCore.Qt.LayoutDirection
    fontMetrics = ... # type: QtGui.QFontMetrics
    palette = ... # type: QtGui.QPalette
    rect = ... # type: QtCore.QRect
    state = ... # type: typing.Union[QStyle.State, QStyle.StateFlag]
    styleObject = ... # type: QtCore.QObject
    type = ... # type: int
    version = ... # type: int

    @typing.overload
    def __init__(self, version: int = ..., type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOption') -> None: ...

    def initFrom(self, w: typing.Optional[QWidget]) -> None: ...


class QStyleOptionFocusRect(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionFocusRect.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionFocusRect.StyleOptionType

    backgroundColor = ... # type: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionFocusRect') -> None: ...


class QStyleOptionFrame(QStyleOption):

    class FrameFeature(int):
        None_ = ... # type: QStyleOptionFrame.FrameFeature
        Flat = ... # type: QStyleOptionFrame.FrameFeature
        Rounded = ... # type: QStyleOptionFrame.FrameFeature

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionFrame.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionFrame.StyleOptionType

    class FrameFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) ->
'QStyleOptionFrame.FrameFeatures': ...
```

```
        def __xor__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) ->
'QStyleOptionFrame.FrameFeatures': ...
        def __ior__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) ->
'QStyleOptionFrame.FrameFeatures': ...
        def __or__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) ->
'QStyleOptionFrame.FrameFeatures': ...
        def __iand__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) ->
'QStyleOptionFrame.FrameFeatures': ...
        def __and__(self, f: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']) ->
'QStyleOptionFrame.FrameFeatures': ...
        def __invert__(self) -> 'QStyleOptionFrame.FrameFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    features = ... # type: typing.Union['QStyleOptionFrame.FrameFeatures', 'QStyleOptionFrame.FrameFeature']
    frameShape = ... # type: QFrame.Shape
    lineWidth = ... # type: int
    midLineWidth = ... # type: int

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionFrame') -> None: ...


class QStyleOptionTabWidgetFrame(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionTabWidgetFrame.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionTabWidgetFrame.StyleOptionType

    leftCornerWidgetSize = ... # type: QtCore.QSize
    lineWidth = ... # type: int
    midLineWidth = ... # type: int
    rightCornerWidgetSize = ... # type: QtCore.QSize
    selectedTabRect = ... # type: QtCore.QRect
    shape = ... # type: 'QTabBar.Shape'
    tabBarRect = ... # type: QtCore.QRect
    tabBarSize = ... # type: QtCore.QSize

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionTabWidgetFrame') -> None: ...


class QStyleOptionTabBarBase(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionTabBarBase.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionTabBarBase.StyleOptionType

    documentMode = ... # type: bool
    selectedTabRect = ... # type: QtCore.QRect
    shape = ... # type: 'QTabBar.Shape'
    tabBarRect = ... # type: QtCore.QRect

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionTabBarBase') -> None: ...


class QStyleOptionHeader(QStyleOption):

    class SortIndicator(int):
        None_ = ... # type: QStyleOptionHeader.SortIndicator
        SortUp = ... # type: QStyleOptionHeader.SortIndicator
        SortDown = ... # type: QStyleOptionHeader.SortIndicator

    class SelectedPosition(int):
        NotAdjacent = ... # type: QStyleOptionHeader.SelectedPosition
        NextIsSelected = ... # type: QStyleOptionHeader.SelectedPosition
        PreviousIsSelected = ... # type: QStyleOptionHeader.SelectedPosition
        NextAndPreviousAreSelected = ... # type: QStyleOptionHeader.SelectedPosition

    class SectionPosition(int):
        Beginning = ... # type: QStyleOptionHeader.SectionPosition
        Middle = ... # type: QStyleOptionHeader.SectionPosition
        End = ... # type: QStyleOptionHeader.SectionPosition
        OnlyOneSection = ... # type: QStyleOptionHeader.SectionPosition

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionHeader.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionHeader.StyleOptionType

    icon = ... # type: QtGui.QIcon
    iconAlignment = ... # type: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]
    orientation = ... # type: QtCore.Qt.Orientation
    position = ... # type: 'QStyleOptionHeader.SectionPosition'
    section = ... # type: int
    selectedPosition = ... # type: 'QStyleOptionHeader.SelectedPosition'
    sortIndicator = ... # type: 'QStyleOptionHeader.SortIndicator'
    text = ... # type: typing.Optional[str]
    textAlignment = ... # type: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionHeader') -> None: ...


class QStyleOptionButton(QStyleOption):

    class ButtonFeature(int):
        None_ = ... # type: QStyleOptionButton.ButtonFeature
        Flat = ... # type: QStyleOptionButton.ButtonFeature
        HasMenu = ... # type: QStyleOptionButton.ButtonFeature
        DefaultButton = ... # type: QStyleOptionButton.ButtonFeature
        AutoDefaultButton = ... # type: QStyleOptionButton.ButtonFeature
        CommandLinkButton = ... # type: QStyleOptionButton.ButtonFeature

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionButton.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionButton.StyleOptionType

    class ButtonFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) -> None:
...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) ->
'QStyleOptionButton.ButtonFeatures': ...
        def __xor__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) ->
'QStyleOptionButton.ButtonFeatures': ...
```

```
        def __ior__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) ->
'QStyleOptionButton.ButtonFeatures': ...
        def __or__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) ->
'QStyleOptionButton.ButtonFeatures': ...
        def __iand__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) ->
'QStyleOptionButton.ButtonFeatures': ...
        def __and__(self, f: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']) ->
'QStyleOptionButton.ButtonFeatures': ...
        def __invert__(self) -> 'QStyleOptionButton.ButtonFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    features = ... # type: typing.Union['QStyleOptionButton.ButtonFeatures', 'QStyleOptionButton.ButtonFeature']
    icon = ... # type: QtGui.QIcon
    iconSize = ... # type: QtCore.QSize
    text = ... # type: typing.Optional[str]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionButton') -> None: ...


class QStyleOptionTab(QStyleOption):

    class TabFeature(int):
        None_ = ... # type: QStyleOptionTab.TabFeature
        HasFrame = ... # type: QStyleOptionTab.TabFeature

    class CornerWidget(int):
        NoCornerWidgets = ... # type: QStyleOptionTab.CornerWidget
        LeftCornerWidget = ... # type: QStyleOptionTab.CornerWidget
        RightCornerWidget = ... # type: QStyleOptionTab.CornerWidget

    class SelectedPosition(int):
        NotAdjacent = ... # type: QStyleOptionTab.SelectedPosition
        NextIsSelected = ... # type: QStyleOptionTab.SelectedPosition
        PreviousIsSelected = ... # type: QStyleOptionTab.SelectedPosition

    class TabPosition(int):
        Beginning = ... # type: QStyleOptionTab.TabPosition
        Middle = ... # type: QStyleOptionTab.TabPosition
        End = ... # type: QStyleOptionTab.TabPosition
        OnlyOneTab = ... # type: QStyleOptionTab.TabPosition

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionTab.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionTab.StyleOptionType

    class CornerWidgets(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) ->
'QStyleOptionTab.CornerWidgets': ...
        def __xor__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) ->
'QStyleOptionTab.CornerWidgets': ...
        def __ior__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) ->
'QStyleOptionTab.CornerWidgets': ...
        def __or__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) ->
'QStyleOptionTab.CornerWidgets': ...
```

```
        def __iand__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) ->
'QStyleOptionTab.CornerWidgets': ...
        def __and__(self, f: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']) ->
'QStyleOptionTab.CornerWidgets': ...
        def __invert__(self) -> 'QStyleOptionTab.CornerWidgets': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    class TabFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) ->
'QStyleOptionTab.TabFeatures': ...
        def __xor__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) ->
'QStyleOptionTab.TabFeatures': ...
        def __ior__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) ->
'QStyleOptionTab.TabFeatures': ...
        def __or__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) ->
'QStyleOptionTab.TabFeatures': ...
        def __iand__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) ->
'QStyleOptionTab.TabFeatures': ...
        def __and__(self, f: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']) ->
'QStyleOptionTab.TabFeatures': ...
        def __invert__(self) -> 'QStyleOptionTab.TabFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    cornerWidgets = ... # type: typing.Union['QStyleOptionTab.CornerWidgets', 'QStyleOptionTab.CornerWidget']
    documentMode = ... # type: bool
    features = ... # type: typing.Union['QStyleOptionTab.TabFeatures', 'QStyleOptionTab.TabFeature']
    icon = ... # type: QtGui.QIcon
    iconSize = ... # type: QtCore.QSize
    leftButtonSize = ... # type: QtCore.QSize
    position = ... # type: 'QStyleOptionTab.TabPosition'
    rightButtonSize = ... # type: QtCore.QSize
    row = ... # type: int
    selectedPosition = ... # type: 'QStyleOptionTab.SelectedPosition'
    shape = ... # type: 'QTabBar.Shape'
    text = ... # type: typing.Optional[str]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionTab') -> None: ...


class QStyleOptionTabV4(QStyleOptionTab):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionTabV4.StyleOptionVersion

    tabIndex = ... # type: int

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QStyleOptionTabV4') -> None: ...


class QStyleOptionProgressBar(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionProgressBar.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionProgressBar.StyleOptionType

    bottomToTop = ... # type: bool
    invertedAppearance = ... # type: bool
    maximum = ... # type: int
    minimum = ... # type: int
    orientation = ... # type: QtCore.Qt.Orientation
    progress = ... # type: int
    text = ... # type: typing.Optional[str]
    textAlignment = ... # type: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]
    textVisible = ... # type: bool

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionProgressBar') -> None: ...


class QStyleOptionMenuItem(QStyleOption):

    class CheckType(int):
        NotCheckable = ... # type: QStyleOptionMenuItem.CheckType
        Exclusive = ... # type: QStyleOptionMenuItem.CheckType
        NonExclusive = ... # type: QStyleOptionMenuItem.CheckType

    class MenuItemType(int):
        Normal = ... # type: QStyleOptionMenuItem.MenuItemType
        DefaultItem = ... # type: QStyleOptionMenuItem.MenuItemType
        Separator = ... # type: QStyleOptionMenuItem.MenuItemType
        SubMenu = ... # type: QStyleOptionMenuItem.MenuItemType
        Scroller = ... # type: QStyleOptionMenuItem.MenuItemType
        TearOff = ... # type: QStyleOptionMenuItem.MenuItemType
        Margin = ... # type: QStyleOptionMenuItem.MenuItemType
        EmptyArea = ... # type: QStyleOptionMenuItem.MenuItemType

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionMenuItem.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionMenuItem.StyleOptionType

    checkType = ... # type: 'QStyleOptionMenuItem.CheckType'
    checked = ... # type: bool
    font = ... # type: QtGui.QFont
    icon = ... # type: QtGui.QIcon
    maxIconWidth = ... # type: int
    menuHasCheckableItems = ... # type: bool
    menuItemType = ... # type: 'QStyleOptionMenuItem.MenuItemType'
    menuRect = ... # type: QtCore.QRect
    tabWidth = ... # type: int
    text = ... # type: typing.Optional[str]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionMenuItem') -> None: ...


class QStyleOptionDockWidget(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionDockWidget.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionDockWidget.StyleOptionType

    closable = ... # type: bool
    floatable = ... # type: bool
    movable = ... # type: bool
    title = ... # type: typing.Optional[str]
    verticalTitleBar = ... # type: bool

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionDockWidget') -> None: ...


class QStyleOptionViewItem(QStyleOption):

    class ViewItemPosition(int):
        Invalid = ... # type: QStyleOptionViewItem.ViewItemPosition
        Beginning = ... # type: QStyleOptionViewItem.ViewItemPosition
        Middle = ... # type: QStyleOptionViewItem.ViewItemPosition
        End = ... # type: QStyleOptionViewItem.ViewItemPosition
        OnlyOne = ... # type: QStyleOptionViewItem.ViewItemPosition

    class ViewItemFeature(int):
        None_ = ... # type: QStyleOptionViewItem.ViewItemFeature
        WrapText = ... # type: QStyleOptionViewItem.ViewItemFeature
        Alternate = ... # type: QStyleOptionViewItem.ViewItemFeature
        HasCheckIndicator = ... # type: QStyleOptionViewItem.ViewItemFeature
        HasDisplay = ... # type: QStyleOptionViewItem.ViewItemFeature
        HasDecoration = ... # type: QStyleOptionViewItem.ViewItemFeature

    class Position(int):
        Left = ... # type: QStyleOptionViewItem.Position
        Right = ... # type: QStyleOptionViewItem.Position
        Top = ... # type: QStyleOptionViewItem.Position
        Bottom = ... # type: QStyleOptionViewItem.Position

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionViewItem.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionViewItem.StyleOptionType

    class ViewItemFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature'])
-> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature'])
-> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __xor__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature'])
-> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __ior__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature'])
-> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __or__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature']) -
> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __iand__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures',
'QStyleOptionViewItem.ViewItemFeature']) -> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __and__(self, f: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature'])
-> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __invert__(self) -> 'QStyleOptionViewItem.ViewItemFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    backgroundBrush = ... # type: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]
```

```
    checkState = ... # type: QtCore.Qt.CheckState
    decorationAlignment = ... # type: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]
    decorationPosition = ... # type: 'QStyleOptionViewItem.Position'
    decorationSize = ... # type: QtCore.QSize
    displayAlignment = ... # type: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]
    features = ... # type: typing.Union['QStyleOptionViewItem.ViewItemFeatures', 'QStyleOptionViewItem.ViewItemFeature']
    font = ... # type: QtGui.QFont
    icon = ... # type: QtGui.QIcon
    index = ... # type: QtCore.QModelIndex
    locale = ... # type: QtCore.QLocale
    showDecorationSelected = ... # type: bool
    text = ... # type: typing.Optional[str]
    textElideMode = ... # type: QtCore.Qt.TextElideMode
    viewItemPosition = ... # type: 'QStyleOptionViewItem.ViewItemPosition'
    widget = ... # type: QWidget

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionViewItem') -> None: ...


class QStyleOptionToolBox(QStyleOption):

    class SelectedPosition(int):
        NotAdjacent = ... # type: QStyleOptionToolBox.SelectedPosition
        NextIsSelected = ... # type: QStyleOptionToolBox.SelectedPosition
        PreviousIsSelected = ... # type: QStyleOptionToolBox.SelectedPosition

    class TabPosition(int):
        Beginning = ... # type: QStyleOptionToolBox.TabPosition
        Middle = ... # type: QStyleOptionToolBox.TabPosition
        End = ... # type: QStyleOptionToolBox.TabPosition
        OnlyOneTab = ... # type: QStyleOptionToolBox.TabPosition

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionToolBox.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionToolBox.StyleOptionType

    icon = ... # type: QtGui.QIcon
    position = ... # type: 'QStyleOptionToolBox.TabPosition'
    selectedPosition = ... # type: 'QStyleOptionToolBox.SelectedPosition'
    text = ... # type: typing.Optional[str]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionToolBox') -> None: ...


class QStyleOptionRubberBand(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionRubberBand.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionRubberBand.StyleOptionType

    opaque = ... # type: bool
    shape = ... # type: QRubberBand.Shape

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionRubberBand') -> None: ...


class QStyleOptionComplex(QStyleOption):
    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionComplex.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionComplex.StyleOptionType

    activeSubControls = ... # type: typing.Union[QStyle.SubControls, QStyle.SubControl]
    subControls = ... # type: typing.Union[QStyle.SubControls, QStyle.SubControl]

    @typing.overload
    def __init__(self, version: int = ..., type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionComplex') -> None: ...


class QStyleOptionSlider(QStyleOptionComplex):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionSlider.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionSlider.StyleOptionType

    dialWrapping = ... # type: bool
    maximum = ... # type: int
    minimum = ... # type: int
    notchTarget = ... # type: float
    orientation = ... # type: QtCore.Qt.Orientation
    pageStep = ... # type: int
    singleStep = ... # type: int
    sliderPosition = ... # type: int
    sliderValue = ... # type: int
    tickInterval = ... # type: int
    tickPosition = ... # type: QSlider.TickPosition
    upsideDown = ... # type: bool

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionSlider') -> None: ...


class QStyleOptionSpinBox(QStyleOptionComplex):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionSpinBox.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionSpinBox.StyleOptionType

    buttonSymbols = ... # type: QAbstractSpinBox.ButtonSymbols
    frame = ... # type: bool
    stepEnabled = ... # type: typing.Union[QAbstractSpinBox.StepEnabled, QAbstractSpinBox.StepEnabledFlag]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionSpinBox') -> None: ...


class QStyleOptionToolButton(QStyleOptionComplex):

    class ToolButtonFeature(int):
        None_ = ... # type: QStyleOptionToolButton.ToolButtonFeature
        Arrow = ... # type: QStyleOptionToolButton.ToolButtonFeature
        Menu = ... # type: QStyleOptionToolButton.ToolButtonFeature
        PopupDelay = ... # type: QStyleOptionToolButton.ToolButtonFeature
        MenuButtonPopup = ... # type: QStyleOptionToolButton.ToolButtonFeature
        HasMenu = ... # type: QStyleOptionToolButton.ToolButtonFeature
    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionToolButton.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionToolButton.StyleOptionType

    class ToolButtonFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __xor__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __ior__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __or__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __iand__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __and__(self, f: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __invert__(self) -> 'QStyleOptionToolButton.ToolButtonFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    arrowType = ... # type: QtCore.Qt.ArrowType
    features = ... # type: typing.Union['QStyleOptionToolButton.ToolButtonFeatures',
'QStyleOptionToolButton.ToolButtonFeature']
    font = ... # type: QtGui.QFont
    icon = ... # type: QtGui.QIcon
    iconSize = ... # type: QtCore.QSize
    pos = ... # type: QtCore.QPoint
    text = ... # type: typing.Optional[str]
    toolButtonStyle = ... # type: QtCore.Qt.ToolButtonStyle

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionToolButton') -> None: ...


class QStyleOptionComboBox(QStyleOptionComplex):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionComboBox.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionComboBox.StyleOptionType

    currentIcon = ... # type: QtGui.QIcon
    currentText = ... # type: typing.Optional[str]
    editable = ... # type: bool
    frame = ... # type: bool
    iconSize = ... # type: QtCore.QSize
    popupRect = ... # type: QtCore.QRect

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionComboBox') -> None: ...
```

```
class QStyleOptionTitleBar(QStyleOptionComplex):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionTitleBar.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionTitleBar.StyleOptionType

    icon = ... # type: QtGui.QIcon
    text = ... # type: typing.Optional[str]
    titleBarFlags = ... # type: typing.Union[QtCore.Qt.WindowFlags, QtCore.Qt.WindowType]
    titleBarState = ... # type: int

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionTitleBar') -> None: ...


class QStyleHintReturn(PyQt5.sipsimplewrapper):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleHintReturn.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleHintReturn.StyleOptionType

    class HintReturnType(int):
        SH_Default = ... # type: QStyleHintReturn.HintReturnType
        SH_Mask = ... # type: QStyleHintReturn.HintReturnType
        SH_Variant = ... # type: QStyleHintReturn.HintReturnType

    type = ... # type: int
    version = ... # type: int

    @typing.overload
    def __init__(self, version: int = ..., type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QStyleHintReturn') -> None: ...


class QStyleHintReturnMask(QStyleHintReturn):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleHintReturnMask.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleHintReturnMask.StyleOptionType

    region = ... # type: QtGui.QRegion

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QStyleHintReturnMask') -> None: ...


class QStyleOptionToolBar(QStyleOption):

    class ToolBarFeature(int):
        None_ = ... # type: QStyleOptionToolBar.ToolBarFeature
        Movable = ... # type: QStyleOptionToolBar.ToolBarFeature

    class ToolBarPosition(int):
        Beginning = ... # type: QStyleOptionToolBar.ToolBarPosition
        Middle = ... # type: QStyleOptionToolBar.ToolBarPosition
        End = ... # type: QStyleOptionToolBar.ToolBarPosition
        OnlyOne = ... # type: QStyleOptionToolBar.ToolBarPosition
    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionToolBar.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionToolBar.StyleOptionType

    class ToolBarFeatures(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
'QStyleOptionToolBar.ToolBarFeatures': ...
        def __xor__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
'QStyleOptionToolBar.ToolBarFeatures': ...
        def __ior__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
'QStyleOptionToolBar.ToolBarFeatures': ...
        def __or__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
'QStyleOptionToolBar.ToolBarFeatures': ...
        def __iand__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
'QStyleOptionToolBar.ToolBarFeatures': ...
        def __and__(self, f: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']) -> 
'QStyleOptionToolBar.ToolBarFeatures': ...
        def __invert__(self) -> 'QStyleOptionToolBar.ToolBarFeatures': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    features = ... # type: typing.Union['QStyleOptionToolBar.ToolBarFeatures', 'QStyleOptionToolBar.ToolBarFeature']
    lineWidth = ... # type: int
    midLineWidth = ... # type: int
    positionOfLine = ... # type: 'QStyleOptionToolBar.ToolBarPosition'
    positionWithinLine = ... # type: 'QStyleOptionToolBar.ToolBarPosition'
    toolBarArea = ... # type: QtCore.Qt.ToolBarArea

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionToolBar') -> None: ...


class QStyleOptionGroupBox(QStyleOptionComplex):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionGroupBox.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionGroupBox.StyleOptionType

    features = ... # type: typing.Union[QStyleOptionFrame.FrameFeatures, QStyleOptionFrame.FrameFeature]
    lineWidth = ... # type: int
    midLineWidth = ... # type: int
    text = ... # type: typing.Optional[str]
    textAlignment = ... # type: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]
    textColor = ... # type: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionGroupBox') -> None: ...


class QStyleOptionSizeGrip(QStyleOptionComplex):
    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionSizeGrip.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionSizeGrip.StyleOptionType

    corner = ... # type: QtCore.Qt.Corner

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionSizeGrip') -> None: ...


class QStyleOptionGraphicsItem(QStyleOption):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleOptionGraphicsItem.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleOptionGraphicsItem.StyleOptionType

    exposedRect = ... # type: QtCore.QRectF

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, other: 'QStyleOptionGraphicsItem') -> None: ...

    @staticmethod
    def levelOfDetailFromTransform(worldTransform: QtGui.QTransform) -> float: ...


class QStyleHintReturnVariant(QStyleHintReturn):

    class StyleOptionVersion(int):
        Version = ... # type: QStyleHintReturnVariant.StyleOptionVersion

    class StyleOptionType(int):
        Type = ... # type: QStyleHintReturnVariant.StyleOptionType

    variant = ... # type: typing.Any

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, a0: 'QStyleHintReturnVariant') -> None: ...


class QStylePainter(QtGui.QPainter):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, w: typing.Optional[QWidget]) -> None: ...
    @typing.overload
    def __init__(self, pd: typing.Optional[QtGui.QPaintDevice], w: typing.Optional[QWidget]) -> None: ...

    def drawItemPixmap(self, r: QtCore.QRect, flags: int, pixmap: QtGui.QPixmap) -> None: ...
    def drawItemText(self, rect: QtCore.QRect, flags: int, pal: QtGui.QPalette, enabled: bool, text: typing.Optional[str],
textRole: QtGui.QPalette.ColorRole = ...) -> None: ...
    def drawComplexControl(self, cc: QStyle.ComplexControl, opt: QStyleOptionComplex) -> None: ...
    def drawControl(self, ce: QStyle.ControlElement, opt: QStyleOption) -> None: ...
    def drawPrimitive(self, pe: QStyle.PrimitiveElement, opt: QStyleOption) -> None: ...
    def style(self) -> typing.Optional[QStyle]: ...
    @typing.overload
    def begin(self, w: typing.Optional[QWidget]) -> bool: ...
    @typing.overload
    def begin(self, pd: typing.Optional[QtGui.QPaintDevice], w: typing.Optional[QWidget]) -> bool: ...
```

```
class QSystemTrayIcon(QtCore.QObject):

    class MessageIcon(int):
        NoIcon = ... # type: QSystemTrayIcon.MessageIcon
        Information = ... # type: QSystemTrayIcon.MessageIcon
        Warning = ... # type: QSystemTrayIcon.MessageIcon
        Critical = ... # type: QSystemTrayIcon.MessageIcon

    class ActivationReason(int):
        Unknown = ... # type: QSystemTrayIcon.ActivationReason
        Context = ... # type: QSystemTrayIcon.ActivationReason
        DoubleClick = ... # type: QSystemTrayIcon.ActivationReason
        Trigger = ... # type: QSystemTrayIcon.ActivationReason
        MiddleClick = ... # type: QSystemTrayIcon.ActivationReason

    @typing.overload
    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...
    @typing.overload
    def __init__(self, icon: QtGui.QIcon, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    messageClicked: typing.ClassVar[QtCore.pyqtSignal]
    activated: typing.ClassVar[QtCore.pyqtSignal]
    def show(self) -> None: ...
    def setVisible(self, visible: bool) -> None: ...
    def hide(self) -> None: ...
    def isVisible(self) -> bool: ...
    @typing.overload
    def showMessage(self, title: typing.Optional[str], msg: typing.Optional[str], icon: 'QSystemTrayIcon.MessageIcon' = ...,
msecs: int = ...) -> None: ...
    @typing.overload
    def showMessage(self, title: typing.Optional[str], msg: typing.Optional[str], icon: QtGui.QIcon, msecs: int = ...) -> None:
...
    @staticmethod
    def supportsMessages() -> bool: ...
    @staticmethod
    def isSystemTrayAvailable() -> bool: ...
    def setToolTip(self, tip: typing.Optional[str]) -> None: ...
    def toolTip(self) -> str: ...
    def setIcon(self, icon: QtGui.QIcon) -> None: ...
    def icon(self) -> QtGui.QIcon: ...
    def geometry(self) -> QtCore.QRect: ...
    def contextMenu(self) -> typing.Optional[QMenu]: ...
    def setContextMenu(self, menu: typing.Optional[QMenu]) -> None: ...


class QTabBar(QWidget):

    class SelectionBehavior(int):
        SelectLeftTab = ... # type: QTabBar.SelectionBehavior
        SelectRightTab = ... # type: QTabBar.SelectionBehavior
        SelectPreviousTab = ... # type: QTabBar.SelectionBehavior

    class ButtonPosition(int):
        LeftSide = ... # type: QTabBar.ButtonPosition
        RightSide = ... # type: QTabBar.ButtonPosition

    class Shape(int):
        RoundedNorth = ... # type: QTabBar.Shape
        RoundedSouth = ... # type: QTabBar.Shape
        RoundedWest = ... # type: QTabBar.Shape
        RoundedEast = ... # type: QTabBar.Shape
        TriangularNorth = ... # type: QTabBar.Shape
        TriangularSouth = ... # type: QTabBar.Shape
        TriangularWest = ... # type: QTabBar.Shape
        TriangularEast = ... # type: QTabBar.Shape

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
def setTabVisible(self, index: int, visible: bool) -> None: ...
def isTabVisible(self, index: int) -> bool: ...
def setAccessibleTabName(self, index: int, name: typing.Optional[str]) -> None: ...
def accessibleTabName(self, index: int) -> str: ...
def timerEvent(self, event: typing.Optional[QtCore.QTimerEvent]) -> None: ...
def setChangeCurrentOnDrag(self, change: bool) -> None: ...
def changeCurrentOnDrag(self) -> bool: ...
def setAutoHide(self, hide: bool) -> None: ...
def autoHide(self) -> bool: ...
tabBarDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
tabBarClicked: typing.ClassVar[QtCore.pyqtSignal]
def minimumTabSizeHint(self, index: int) -> QtCore.QSize: ...
def wheelEvent(self, event: typing.Optional[QtGui.QWheelEvent]) -> None: ...
def hideEvent(self, a0: typing.Optional[QtGui.QHideEvent]) -> None: ...
tabMoved: typing.ClassVar[QtCore.pyqtSignal]
tabCloseRequested: typing.ClassVar[QtCore.pyqtSignal]
def setDocumentMode(self, set: bool) -> None: ...
def documentMode(self) -> bool: ...
def setMovable(self, movable: bool) -> None: ...
def isMovable(self) -> bool: ...
def setExpanding(self, enabled: bool) -> None: ...
def expanding(self) -> bool: ...
def setSelectionBehaviorOnRemove(self, behavior: 'QTabBar.SelectionBehavior') -> None: ...
def selectionBehaviorOnRemove(self) -> 'QTabBar.SelectionBehavior': ...
def tabButton(self, index: int, position: 'QTabBar.ButtonPosition') -> typing.Optional[QWidget]: ...
def setTabButton(self, index: int, position: 'QTabBar.ButtonPosition', widget: typing.Optional[QWidget]) -> None: ...
def setTabsClosable(self, closable: bool) -> None: ...
def tabsClosable(self) -> bool: ...
def moveTab(self, from_: int, to: int) -> None: ...
def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def mouseMoveEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
def tabLayoutChange(self) -> None: ...
def tabRemoved(self, index: int) -> None: ...
def tabInserted(self, index: int) -> None: ...
def tabSizeHint(self, index: int) -> QtCore.QSize: ...
def initStyleOption(self, option: typing.Optional[QStyleOptionTab], tabIndex: int) -> None: ...
currentChanged: typing.ClassVar[QtCore.pyqtSignal]
def setCurrentIndex(self, index: int) -> None: ...
def usesScrollButtons(self) -> bool: ...
def setUsesScrollButtons(self, useButtons: bool) -> None: ...
def setElideMode(self, a0: QtCore.Qt.TextElideMode) -> None: ...
def elideMode(self) -> QtCore.Qt.TextElideMode: ...
def setIconSize(self, size: QtCore.QSize) -> None: ...
def iconSize(self) -> QtCore.QSize: ...
def drawBase(self) -> bool: ...
def setDrawBase(self, drawTheBase: bool) -> None: ...
def minimumSizeHint(self) -> QtCore.QSize: ...
def sizeHint(self) -> QtCore.QSize: ...
def __len__(self) -> int: ...
def count(self) -> int: ...
def currentIndex(self) -> int: ...
def tabRect(self, index: int) -> QtCore.QRect: ...
def tabAt(self, pos: QtCore.QPoint) -> int: ...
def tabData(self, index: int) -> typing.Any: ...
def setTabData(self, index: int, data: typing.Any) -> None: ...
def tabWhatsThis(self, index: int) -> str: ...
def setTabWhatsThis(self, index: int, text: typing.Optional[str]) -> None: ...
def tabToolTip(self, index: int) -> str: ...
def setTabToolTip(self, index: int, tip: typing.Optional[str]) -> None: ...
def setTabIcon(self, index: int, icon: QtGui.QIcon) -> None: ...
def tabIcon(self, index: int) -> QtGui.QIcon: ...
def setTabTextColor(self, index: int, color: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
```

```
    def tabTextColor(self, index: int) -> QtGui.QColor: ...
    def setTabText(self, index: int, text: typing.Optional[str]) -> None: ...
    def tabText(self, index: int) -> str: ...
    def setTabEnabled(self, index: int, a1: bool) -> None: ...
    def isTabEnabled(self, index: int) -> bool: ...
    def removeTab(self, index: int) -> None: ...
    @typing.overload
    def insertTab(self, index: int, text: typing.Optional[str]) -> int: ...
    @typing.overload
    def insertTab(self, index: int, icon: QtGui.QIcon, text: typing.Optional[str]) -> int: ...
    @typing.overload
    def addTab(self, text: typing.Optional[str]) -> int: ...
    @typing.overload
    def addTab(self, icon: QtGui.QIcon, text: typing.Optional[str]) -> int: ...
    def setShape(self, shape: 'QTabBar.Shape') -> None: ...
    def shape(self) -> 'QTabBar.Shape': ...


class QTableView(QAbstractItemView):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def currentChanged(self, current: QtCore.QModelIndex, previous: QtCore.QModelIndex) -> None: ...
    def selectionChanged(self, selected: QtCore.QItemSelection, deselected: QtCore.QItemSelection) -> None: ...
    def clearSpans(self) -> None: ...
    def isCornerButtonEnabled(self) -> bool: ...
    def setCornerButtonEnabled(self, enable: bool) -> None: ...
    def wordWrap(self) -> bool: ...
    def setWordWrap(self, on: bool) -> None: ...
    def sortByColumn(self, column: int, order: QtCore.Qt.SortOrder) -> None: ...
    def columnSpan(self, row: int, column: int) -> int: ...
    def rowSpan(self, row: int, column: int) -> int: ...
    def setSpan(self, row: int, column: int, rowSpan: int, columnSpan: int) -> None: ...
    def isSortingEnabled(self) -> bool: ...
    def setSortingEnabled(self, enable: bool) -> None: ...
    def viewportSizeHint(self) -> QtCore.QSize: ...
    def isIndexHidden(self, index: QtCore.QModelIndex) -> bool: ...
    def horizontalScrollbarAction(self, action: int) -> None: ...
    def verticalScrollbarAction(self, action: int) -> None: ...
    def sizeHintForColumn(self, column: int) -> int: ...
    def sizeHintForRow(self, row: int) -> int: ...
    def updateGeometries(self) -> None: ...
    def selectedIndexes(self) -> typing.List[QtCore.QModelIndex]: ...
    def visualRegionForSelection(self, selection: QtCore.QItemSelection) -> QtGui.QRegion: ...
    def setSelection(self, rect: QtCore.QRect, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def moveCursor(self, cursorAction: QAbstractItemView.CursorAction, modifiers: typing.Union[QtCore.Qt.KeyboardModifiers,
QtCore.Qt.KeyboardModifier]) -> QtCore.QModelIndex: ...
    def verticalOffset(self) -> int: ...
    def horizontalOffset(self) -> int: ...
    def timerEvent(self, event: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def viewOptions(self) -> QStyleOptionViewItem: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def columnCountChanged(self, oldCount: int, newCount: int) -> None: ...
    def rowCountChanged(self, oldCount: int, newCount: int) -> None: ...
    def columnResized(self, column: int, oldWidth: int, newWidth: int) -> None: ...
    def rowResized(self, row: int, oldHeight: int, newHeight: int) -> None: ...
    def columnMoved(self, column: int, oldIndex: int, newIndex: int) -> None: ...
    def rowMoved(self, row: int, oldIndex: int, newIndex: int) -> None: ...
    def resizeColumnsToContents(self) -> None: ...
    def resizeColumnToContents(self, column: int) -> None: ...
    def resizeRowsToContents(self) -> None: ...
    def resizeRowToContents(self, row: int) -> None: ...
    def showColumn(self, column: int) -> None: ...
    def showRow(self, row: int) -> None: ...
    def hideColumn(self, column: int) -> None: ...
    def hideRow(self, row: int) -> None: ...
    def selectColumn(self, column: int) -> None: ...
    def selectRow(self, row: int) -> None: ...
    def indexAt(self, p: QtCore.QPoint) -> QtCore.QModelIndex: ...
    def scrollTo(self, index: QtCore.QModelIndex, hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def visualRect(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def setGridStyle(self, style: QtCore.Qt.PenStyle) -> None: ...
    def gridStyle(self) -> QtCore.Qt.PenStyle: ...
    def setShowGrid(self, show: bool) -> None: ...
    def showGrid(self) -> bool: ...
    def setColumnHidden(self, column: int, hide: bool) -> None: ...
    def isColumnHidden(self, column: int) -> bool: ...
    def setRowHidden(self, row: int, hide: bool) -> None: ...
    def isRowHidden(self, row: int) -> bool: ...
    def columnAt(self, x: int) -> int: ...
    def columnWidth(self, column: int) -> int: ...
    def setColumnWidth(self, column: int, width: int) -> None: ...
    def columnViewportPosition(self, column: int) -> int: ...
    def rowAt(self, y: int) -> int: ...
    def rowHeight(self, row: int) -> int: ...
    def setRowHeight(self, row: int, height: int) -> None: ...
    def rowViewportPosition(self, row: int) -> int: ...
    def setVerticalHeader(self, header: typing.Optional[QHeaderView]) -> None: ...
    def setHorizontalHeader(self, header: typing.Optional[QHeaderView]) -> None: ...
    def verticalHeader(self) -> typing.Optional[QHeaderView]: ...
    def horizontalHeader(self) -> typing.Optional[QHeaderView]: ...
    def setSelectionModel(self, selectionModel: typing.Optional[QtCore.QItemSelectionModel]) -> None: ...
    def setRootIndex(self, index: QtCore.QModelIndex) -> None: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...


class QTableWidgetSelectionRange(PyQt5.sipsimplewrapper):

    @typing.overload
    def __init__(self) -> None: ...
    @typing.overload
    def __init__(self, top: int, left: int, bottom: int, right: int) -> None: ...
    @typing.overload
    def __init__(self, other: 'QTableWidgetSelectionRange') -> None: ...

    def columnCount(self) -> int: ...
    def rowCount(self) -> int: ...
    def rightColumn(self) -> int: ...
    def leftColumn(self) -> int: ...
    def bottomRow(self) -> int: ...
    def topRow(self) -> int: ...


class QTableWidgetItem(PyQt5.sip.wrapper):

    class ItemType(int):
        Type = ... # type: QTableWidgetItem.ItemType
        UserType = ... # type: QTableWidgetItem.ItemType

    @typing.overload
    def __init__(self, type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, icon: QtGui.QIcon, text: typing.Optional[str], type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, other: 'QTableWidgetItem') -> None: ...

    def __ge__(self, other: 'QTableWidgetItem') -> bool: ...
    def isSelected(self) -> bool: ...
    def setSelected(self, aselect: bool) -> None: ...
    def column(self) -> int: ...
    def row(self) -> int: ...
    def setForeground(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
    def foreground(self) -> QtGui.QBrush: ...
    def setBackground(self, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]) -> None: ...
```

```
    def background(self) -> QtGui.QBrush: ...
    def setSizeHint(self, size: QtCore.QSize) -> None: ...
    def sizeHint(self) -> QtCore.QSize: ...
    def setFont(self, afont: QtGui.QFont) -> None: ...
    def setWhatsThis(self, awhatsThis: typing.Optional[str]) -> None: ...
    def setToolTip(self, atoolTip: typing.Optional[str]) -> None: ...
    def setStatusTip(self, astatusTip: typing.Optional[str]) -> None: ...
    def setIcon(self, aicon: QtGui.QIcon) -> None: ...
    def setText(self, atext: typing.Optional[str]) -> None: ...
    def setFlags(self, aflags: typing.Union[QtCore.Qt.ItemFlags, QtCore.Qt.ItemFlag]) -> None: ...
    def type(self) -> int: ...
    def write(self, out: QtCore.QDataStream) -> None: ...
    def read(self, in_: QtCore.QDataStream) -> None: ...
    def __lt__(self, other: 'QTableWidgetItem') -> bool: ...
    def setData(self, role: int, value: typing.Any) -> None: ...
    def data(self, role: int) -> typing.Any: ...
    def setCheckState(self, state: QtCore.Qt.CheckState) -> None: ...
    def checkState(self) -> QtCore.Qt.CheckState: ...
    def setTextAlignment(self, alignment: int) -> None: ...
    def textAlignment(self) -> int: ...
    def font(self) -> QtGui.QFont: ...
    def whatsThis(self) -> str: ...
    def toolTip(self) -> str: ...
    def statusTip(self) -> str: ...
    def icon(self) -> QtGui.QIcon: ...
    def text(self) -> str: ...
    def flags(self) -> QtCore.Qt.ItemFlags: ...
    def tableWidget(self) -> typing.Optional['QTableWidget']: ...
    def clone(self) -> typing.Optional['QTableWidgetItem']: ...


class QTableWidget(QTableView):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, rows: int, columns: int, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isPersistentEditorOpen(self, item: typing.Optional[QTableWidgetItem]) -> bool: ...
    def dropEvent(self, event: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def itemFromIndex(self, index: QtCore.QModelIndex) -> typing.Optional[QTableWidgetItem]: ...
    def indexFromItem(self, item: typing.Optional[QTableWidgetItem]) -> QtCore.QModelIndex: ...
    def items(self, data: typing.Optional[QtCore.QMimeData]) -> typing.List[QTableWidgetItem]: ...
    def supportedDropActions(self) -> QtCore.Qt.DropActions: ...
    def dropMimeData(self, row: int, column: int, data: typing.Optional[QtCore.QMimeData], action: QtCore.Qt.DropAction) ->
bool: ...
    def mimeData(self, items: typing.Iterable[QTableWidgetItem]) -> typing.Optional[QtCore.QMimeData]: ...
    def mimeTypes(self) -> typing.List[str]: ...
    currentCellChanged: typing.ClassVar[QtCore.pyqtSignal]
    cellChanged: typing.ClassVar[QtCore.pyqtSignal]
    cellEntered: typing.ClassVar[QtCore.pyqtSignal]
    cellActivated: typing.ClassVar[QtCore.pyqtSignal]
    cellDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    cellClicked: typing.ClassVar[QtCore.pyqtSignal]
    cellPressed: typing.ClassVar[QtCore.pyqtSignal]
    itemSelectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentItemChanged: typing.ClassVar[QtCore.pyqtSignal]
    itemChanged: typing.ClassVar[QtCore.pyqtSignal]
    itemEntered: typing.ClassVar[QtCore.pyqtSignal]
    itemActivated: typing.ClassVar[QtCore.pyqtSignal]
    itemDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    itemClicked: typing.ClassVar[QtCore.pyqtSignal]
    itemPressed: typing.ClassVar[QtCore.pyqtSignal]
    def clearContents(self) -> None: ...
    def clear(self) -> None: ...
    def removeColumn(self, column: int) -> None: ...
    def removeRow(self, row: int) -> None: ...
    def insertColumn(self, column: int) -> None: ...
    def insertRow(self, row: int) -> None: ...
    def scrollToItem(self, item: typing.Optional[QTableWidgetItem], hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def setItemPrototype(self, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def itemPrototype(self) -> typing.Optional[QTableWidgetItem]: ...
    def visualItemRect(self, item: typing.Optional[QTableWidgetItem]) -> QtCore.QRect: ...
    @typing.overload
    def itemAt(self, p: QtCore.QPoint) -> typing.Optional[QTableWidgetItem]: ...
    @typing.overload
    def itemAt(self, ax: int, ay: int) -> typing.Optional[QTableWidgetItem]: ...
    def visualColumn(self, logicalColumn: int) -> int: ...
    def visualRow(self, logicalRow: int) -> int: ...
    def findItems(self, text: typing.Optional[str], flags: typing.Union[QtCore.Qt.MatchFlags, QtCore.Qt.MatchFlag]) ->
typing.List[QTableWidgetItem]: ...
    def selectedItems(self) -> typing.List[QTableWidgetItem]: ...
    def selectedRanges(self) -> typing.List[QTableWidgetSelectionRange]: ...
    def setRangeSelected(self, range: QTableWidgetSelectionRange, select: bool) -> None: ...
    def removeCellWidget(self, arow: int, acolumn: int) -> None: ...
    def setCellWidget(self, row: int, column: int, widget: typing.Optional[QWidget]) -> None: ...
    def cellWidget(self, row: int, column: int) -> typing.Optional[QWidget]: ...
    def closePersistentEditor(self, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def openPersistentEditor(self, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def editItem(self, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def isSortingEnabled(self) -> bool: ...
    def setSortingEnabled(self, enable: bool) -> None: ...
    def sortItems(self, column: int, order: QtCore.Qt.SortOrder = ...) -> None: ...
    @typing.overload
    def setCurrentCell(self, row: int, column: int) -> None: ...
    @typing.overload
    def setCurrentCell(self, row: int, column: int, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QTableWidgetItem]) -> None: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QTableWidgetItem], command:
typing.Union[QtCore.QItemSelectionModel.SelectionFlags, QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def currentItem(self) -> typing.Optional[QTableWidgetItem]: ...
    def currentColumn(self) -> int: ...
    def currentRow(self) -> int: ...
    def setHorizontalHeaderLabels(self, labels: typing.Iterable[typing.Optional[str]]) -> None: ...
    def setVerticalHeaderLabels(self, labels: typing.Iterable[typing.Optional[str]]) -> None: ...
    def takeHorizontalHeaderItem(self, column: int) -> typing.Optional[QTableWidgetItem]: ...
    def setHorizontalHeaderItem(self, column: int, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def horizontalHeaderItem(self, column: int) -> typing.Optional[QTableWidgetItem]: ...
    def takeVerticalHeaderItem(self, row: int) -> typing.Optional[QTableWidgetItem]: ...
    def setVerticalHeaderItem(self, row: int, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def verticalHeaderItem(self, row: int) -> typing.Optional[QTableWidgetItem]: ...
    def takeItem(self, row: int, column: int) -> typing.Optional[QTableWidgetItem]: ...
    def setItem(self, row: int, column: int, item: typing.Optional[QTableWidgetItem]) -> None: ...
    def item(self, row: int, column: int) -> typing.Optional[QTableWidgetItem]: ...
    def column(self, item: typing.Optional[QTableWidgetItem]) -> int: ...
    def row(self, item: typing.Optional[QTableWidgetItem]) -> int: ...
    def columnCount(self) -> int: ...
    def setColumnCount(self, columns: int) -> None: ...
    def rowCount(self) -> int: ...
    def setRowCount(self, rows: int) -> None: ...


class QTabWidget(QWidget):

    class TabShape(int):
        Rounded = ... # type: QTabWidget.TabShape
        Triangular = ... # type: QTabWidget.TabShape

    class TabPosition(int):
        North = ... # type: QTabWidget.TabPosition
        South = ... # type: QTabWidget.TabPosition
        West = ... # type: QTabWidget.TabPosition
        East = ... # type: QTabWidget.TabPosition

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
def setTabVisible(self, index: int, visible: bool) -> None: ...
def isTabVisible(self, index: int) -> bool: ...
def setTabBarAutoHide(self, enabled: bool) -> None: ...
def tabBarAutoHide(self) -> bool: ...
tabBarDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
tabBarClicked: typing.ClassVar[QtCore.pyqtSignal]
def hasHeightForWidth(self) -> bool: ...
def heightForWidth(self, width: int) -> int: ...
tabCloseRequested: typing.ClassVar[QtCore.pyqtSignal]
def setDocumentMode(self, set: bool) -> None: ...
def documentMode(self) -> bool: ...
def setMovable(self, movable: bool) -> None: ...
def isMovable(self) -> bool: ...
def setTabsClosable(self, closeable: bool) -> None: ...
def tabsClosable(self) -> bool: ...
def setUsesScrollButtons(self, useButtons: bool) -> None: ...
def usesScrollButtons(self) -> bool: ...
def setIconSize(self, size: QtCore.QSize) -> None: ...
def iconSize(self) -> QtCore.QSize: ...
def setElideMode(self, a0: QtCore.Qt.TextElideMode) -> None: ...
def elideMode(self) -> QtCore.Qt.TextElideMode: ...
def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
def tabBar(self) -> typing.Optional[QTabBar]: ...
def setTabBar(self, a0: typing.Optional[QTabBar]) -> None: ...
def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
def keyPressEvent(self, a0: typing.Optional[QtGui.QKeyEvent]) -> None: ...
def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
def tabRemoved(self, index: int) -> None: ...
def tabInserted(self, index: int) -> None: ...
def initStyleOption(self, option: typing.Optional[QStyleOptionTabWidgetFrame]) -> None: ...
currentChanged: typing.ClassVar[QtCore.pyqtSignal]
def setCurrentWidget(self, widget: typing.Optional[QWidget]) -> None: ...
def setCurrentIndex(self, index: int) -> None: ...
def cornerWidget(self, corner: QtCore.Qt.Corner = ...) -> typing.Optional[QWidget]: ...
def setCornerWidget(self, widget: typing.Optional[QWidget], corner: QtCore.Qt.Corner = ...) -> None: ...
def minimumSizeHint(self) -> QtCore.QSize: ...
def sizeHint(self) -> QtCore.QSize: ...
def setTabShape(self, s: 'QTabWidget.TabShape') -> None: ...
def tabShape(self) -> 'QTabWidget.TabShape': ...
def setTabPosition(self, a0: 'QTabWidget.TabPosition') -> None: ...
def tabPosition(self) -> 'QTabWidget.TabPosition': ...
def __len__(self) -> int: ...
def count(self) -> int: ...
def indexOf(self, widget: typing.Optional[QWidget]) -> int: ...
def widget(self, index: int) -> typing.Optional[QWidget]: ...
def currentWidget(self) -> typing.Optional[QWidget]: ...
def currentIndex(self) -> int: ...
def tabWhatsThis(self, index: int) -> str: ...
def setTabWhatsThis(self, index: int, text: typing.Optional[str]) -> None: ...
def tabToolTip(self, index: int) -> str: ...
def setTabToolTip(self, index: int, tip: typing.Optional[str]) -> None: ...
def setTabIcon(self, index: int, icon: QtGui.QIcon) -> None: ...
def tabIcon(self, index: int) -> QtGui.QIcon: ...
def setTabText(self, index: int, a1: typing.Optional[str]) -> None: ...
def tabText(self, index: int) -> str: ...
def setTabEnabled(self, index: int, a1: bool) -> None: ...
def isTabEnabled(self, index: int) -> bool: ...
def removeTab(self, index: int) -> None: ...
@typing.overload
def insertTab(self, index: int, widget: typing.Optional[QWidget], a2: typing.Optional[str]) -> int: ...
@typing.overload
def insertTab(self, index: int, widget: typing.Optional[QWidget], icon: QtGui.QIcon, label: typing.Optional[str]) -> int: ...
@typing.overload
def addTab(self, widget: typing.Optional[QWidget], a1: typing.Optional[str]) -> int: ...
@typing.overload
def addTab(self, widget: typing.Optional[QWidget], icon: QtGui.QIcon, label: typing.Optional[str]) -> int: ...
def clear(self) -> None: ...
```

```
class QTextEdit(QAbstractScrollArea):

    class AutoFormattingFlag(int):
        AutoNone = ... # type: QTextEdit.AutoFormattingFlag
        AutoBulletList = ... # type: QTextEdit.AutoFormattingFlag
        AutoAll = ... # type: QTextEdit.AutoFormattingFlag

    class LineWrapMode(int):
        NoWrap = ... # type: QTextEdit.LineWrapMode
        WidgetWidth = ... # type: QTextEdit.LineWrapMode
        FixedPixelWidth = ... # type: QTextEdit.LineWrapMode
        FixedColumnWidth = ... # type: QTextEdit.LineWrapMode

    class ExtraSelection(PyQt5.sipsimplewrapper):

        cursor = ... # type: QtGui.QTextCursor
        format = ... # type: QtGui.QTextCharFormat

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, a0: 'QTextEdit.ExtraSelection') -> None: ...

    class AutoFormatting(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) ->
'QTextEdit.AutoFormatting': ...
        def __xor__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) ->
'QTextEdit.AutoFormatting': ...
        def __ior__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) ->
'QTextEdit.AutoFormatting': ...
        def __or__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) ->
'QTextEdit.AutoFormatting': ...
        def __iand__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) ->
'QTextEdit.AutoFormatting': ...
        def __and__(self, f: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) ->
'QTextEdit.AutoFormatting': ...
        def __invert__(self) -> 'QTextEdit.AutoFormatting': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...

    def setMarkdown(self, markdown: typing.Optional[str]) -> None: ...
    def toMarkdown(self, features: typing.Union[QtGui.QTextDocument.MarkdownFeatures,
QtGui.QTextDocument.MarkdownFeature] = ...) -> str: ...
    def setTabStopDistance(self, distance: float) -> None: ...
    def tabStopDistance(self) -> float: ...
    def placeholderText(self) -> str: ...
    def setPlaceholderText(self, placeholderText: typing.Optional[str]) -> None: ...
    def setTextBackgroundColor(self, c: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    def textBackgroundColor(self) -> QtGui.QColor: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    @typing.overload
    def inputMethodQuery(self, property: QtCore.Qt.InputMethodQuery) -> typing.Any: ...
    @typing.overload
    def inputMethodQuery(self, query: QtCore.Qt.InputMethodQuery, argument: typing.Any) -> typing.Any: ...
    def inputMethodEvent(self, a0: typing.Optional[QtGui.QInputMethodEvent]) -> None: ...
    def insertFromMimeData(self, source: typing.Optional[QtCore.QMimeData]) -> None: ...
    def canInsertFromMimeData(self, source: typing.Optional[QtCore.QMimeData]) -> bool: ...
    def createMimeDataFromSelection(self) -> typing.Optional[QtCore.QMimeData]: ...
    def wheelEvent(self, e: typing.Optional[QtGui.QWheelEvent]) -> None: ...
    def changeEvent(self, e: typing.Optional[QtCore.QEvent]) -> None: ...
    def showEvent(self, a0: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def focusOutEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def focusInEvent(self, e: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def dropEvent(self, e: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def dragMoveEvent(self, e: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def dragLeaveEvent(self, e: typing.Optional[QtGui.QDragLeaveEvent]) -> None: ...
    def dragEnterEvent(self, e: typing.Optional[QtGui.QDragEnterEvent]) -> None: ...
    def contextMenuEvent(self, e: typing.Optional[QtGui.QContextMenuEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def mouseDoubleClickEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseReleaseEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, a0: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def keyReleaseEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def keyPressEvent(self, e: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def timerEvent(self, e: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    cursorPositionChanged: typing.ClassVar[QtCore.pyqtSignal]
    selectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    copyAvailable: typing.ClassVar[QtCore.pyqtSignal]
    currentCharFormatChanged: typing.ClassVar[QtCore.pyqtSignal]
    redoAvailable: typing.ClassVar[QtCore.pyqtSignal]
    undoAvailable: typing.ClassVar[QtCore.pyqtSignal]
    textChanged: typing.ClassVar[QtCore.pyqtSignal]
    def zoomOut(self, range: int = ...) -> None: ...
    def zoomIn(self, range: int = ...) -> None: ...
    def undo(self) -> None: ...
    def redo(self) -> None: ...
    def scrollToAnchor(self, name: typing.Optional[str]) -> None: ...
    def insertHtml(self, text: typing.Optional[str]) -> None: ...
    def insertPlainText(self, text: typing.Optional[str]) -> None: ...
    def selectAll(self) -> None: ...
    def clear(self) -> None: ...
    def paste(self) -> None: ...
    def copy(self) -> None: ...
    def cut(self) -> None: ...
    def setHtml(self, text: typing.Optional[str]) -> None: ...
    def setPlainText(self, text: typing.Optional[str]) -> None: ...
    def setAlignment(self, a: typing.Union[QtCore.Qt.Alignment, QtCore.Qt.AlignmentFlag]) -> None: ...
    def setCurrentFont(self, f: QtGui.QFont) -> None: ...
    def setTextColor(self, c: typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor]) -> None: ...
    def setText(self, text: typing.Optional[str]) -> None: ...
    def setFontItalic(self, b: bool) -> None: ...
    def setFontUnderline(self, b: bool) -> None: ...
    def setFontWeight(self, w: int) -> None: ...
    def setFontFamily(self, fontFamily: typing.Optional[str]) -> None: ...
    def setFontPointSize(self, s: float) -> None: ...
    def print(self, printer: typing.Optional[QtGui.QPagedPaintDevice]) -> None: ...
    def print_(self, printer: typing.Optional[QtGui.QPagedPaintDevice]) -> None: ...
    def moveCursor(self, operation: QtGui.QTextCursor.MoveOperation, mode: QtGui.QTextCursor.MoveMode = ...) -> None:
...
    def canPaste(self) -> bool: ...
    def extraSelections(self) -> typing.List['QTextEdit.ExtraSelection']: ...
    def setExtraSelections(self, selections: typing.Iterable['QTextEdit.ExtraSelection']) -> None: ...
    def cursorWidth(self) -> int: ...
    def setCursorWidth(self, width: int) -> None: ...
    def textInteractionFlags(self) -> QtCore.Qt.TextInteractionFlags: ...
    def setTextInteractionFlags(self, flags: typing.Union[QtCore.Qt.TextInteractionFlags, QtCore.Qt.TextInteractionFlag]) ->
None: ...
    def setAcceptRichText(self, accept: bool) -> None: ...
    def acceptRichText(self) -> bool: ...
    def setTabStopWidth(self, width: int) -> None: ...
    def tabStopWidth(self) -> int: ...
    def setOverwriteMode(self, overwrite: bool) -> None: ...
    def overwriteMode(self) -> bool: ...
    def anchorAt(self, pos: QtCore.QPoint) -> str: ...
    @typing.overload
    def cursorRect(self, cursor: QtGui.QTextCursor) -> QtCore.QRect: ...
    @typing.overload
    def cursorRect(self) -> QtCore.QRect: ...
    def cursorForPosition(self, pos: QtCore.QPoint) -> QtGui.QTextCursor: ...
    @typing.overload
    def createStandardContextMenu(self) -> typing.Optional[QMenu]: ...
    @typing.overload
    def createStandardContextMenu(self, position: QtCore.QPoint) -> typing.Optional[QMenu]: ...
    def loadResource(self, type: int, name: QtCore.QUrl) -> typing.Any: ...
    def ensureCursorVisible(self) -> None: ...
    def append(self, text: typing.Optional[str]) -> None: ...
    def toHtml(self) -> str: ...
    def toPlainText(self) -> str: ...
    @typing.overload
    def find(self, exp: typing.Optional[str], options: typing.Union[QtGui.QTextDocument.FindFlags,
QtGui.QTextDocument.FindFlag] = ...) -> bool: ...
    @typing.overload
    def find(self, exp: QtCore.QRegExp, options: typing.Union[QtGui.QTextDocument.FindFlags,
QtGui.QTextDocument.FindFlag] = ...) -> bool: ...
    @typing.overload
    def find(self, exp: QtCore.QRegularExpression, options: typing.Union[QtGui.QTextDocument.FindFlags,
QtGui.QTextDocument.FindFlag] = ...) -> bool: ...
    def setWordWrapMode(self, policy: QtGui.QTextOption.WrapMode) -> None: ...
    def wordWrapMode(self) -> QtGui.QTextOption.WrapMode: ...
    def setLineWrapColumnOrWidth(self, w: int) -> None: ...
    def lineWrapColumnOrWidth(self) -> int: ...
    def setLineWrapMode(self, mode: 'QTextEdit.LineWrapMode') -> None: ...
    def lineWrapMode(self) -> 'QTextEdit.LineWrapMode': ...
    def setUndoRedoEnabled(self, enable: bool) -> None: ...
    def isUndoRedoEnabled(self) -> bool: ...
    def documentTitle(self) -> str: ...
    def setDocumentTitle(self, title: typing.Optional[str]) -> None: ...
    def setTabChangesFocus(self, b: bool) -> None: ...
    def tabChangesFocus(self) -> bool: ...
    def setAutoFormatting(self, features: typing.Union['QTextEdit.AutoFormatting', 'QTextEdit.AutoFormattingFlag']) -> None:
...
    def autoFormatting(self) -> 'QTextEdit.AutoFormatting': ...
    def currentCharFormat(self) -> QtGui.QTextCharFormat: ...
    def setCurrentCharFormat(self, format: QtGui.QTextCharFormat) -> None: ...
    def mergeCurrentCharFormat(self, modifier: QtGui.QTextCharFormat) -> None: ...
    def alignment(self) -> QtCore.Qt.Alignment: ...
    def currentFont(self) -> QtGui.QFont: ...
    def textColor(self) -> QtGui.QColor: ...
    def fontItalic(self) -> bool: ...
    def fontUnderline(self) -> bool: ...
    def fontWeight(self) -> int: ...
    def fontFamily(self) -> str: ...
    def fontPointSize(self) -> float: ...
    def setReadOnly(self, ro: bool) -> None: ...
    def isReadOnly(self) -> bool: ...
    def textCursor(self) -> QtGui.QTextCursor: ...
    def setTextCursor(self, cursor: QtGui.QTextCursor) -> None: ...
    def document(self) -> typing.Optional[QtGui.QTextDocument]: ...
    def setDocument(self, document: typing.Optional[QtGui.QTextDocument]) -> None: ...


class QTextBrowser(QTextEdit):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def doSetSource(self, name: QtCore.QUrl, type: QtGui.QTextDocument.ResourceType = ...) -> None: ...
    def sourceType(self) -> QtGui.QTextDocument.ResourceType: ...
    historyChanged: typing.ClassVar[QtCore.pyqtSignal]
    def forwardHistoryCount(self) -> int: ...
    def backwardHistoryCount(self) -> int: ...
    def historyUrl(self, a0: int) -> QtCore.QUrl: ...
    def historyTitle(self, a0: int) -> str: ...
    def setOpenLinks(self, open: bool) -> None: ...
    def openLinks(self) -> bool: ...
    def setOpenExternalLinks(self, open: bool) -> None: ...
    def openExternalLinks(self) -> bool: ...
    def clearHistory(self) -> None: ...
    def isForwardAvailable(self) -> bool: ...
    def isBackwardAvailable(self) -> bool: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def focusNextPrevChild(self, next: bool) -> bool: ...
    def focusOutEvent(self, ev: typing.Optional[QtGui.QFocusEvent]) -> None: ...
    def mouseReleaseEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, ev: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def keyPressEvent(self, ev: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    anchorClicked: typing.ClassVar[QtCore.pyqtSignal]
    highlighted: typing.ClassVar[QtCore.pyqtSignal]
    sourceChanged: typing.ClassVar[QtCore.pyqtSignal]
    forwardAvailable: typing.ClassVar[QtCore.pyqtSignal]
    backwardAvailable: typing.ClassVar[QtCore.pyqtSignal]
    def reload(self) -> None: ...
    def home(self) -> None: ...
    def forward(self) -> None: ...
    def backward(self) -> None: ...
    @typing.overload
    def setSource(self, name: QtCore.QUrl) -> None: ...
    @typing.overload
    def setSource(self, name: QtCore.QUrl, type: QtGui.QTextDocument.ResourceType) -> None: ...
    def loadResource(self, type: int, name: QtCore.QUrl) -> typing.Any: ...
    def setSearchPaths(self, paths: typing.Iterable[typing.Optional[str]]) -> None: ...
    def searchPaths(self) -> typing.List[str]: ...
    def source(self) -> QtCore.QUrl: ...


class QToolBar(QWidget):

    @typing.overload
    def __init__(self, title: typing.Optional[str], parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isFloating(self) -> bool: ...
    def setFloatable(self, floatable: bool) -> None: ...
    def isFloatable(self) -> bool: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def paintEvent(self, event: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def changeEvent(self, event: typing.Optional[QtCore.QEvent]) -> None: ...
    def actionEvent(self, event: typing.Optional[QtGui.QActionEvent]) -> None: ...
    def initStyleOption(self, option: typing.Optional[QStyleOptionToolBar]) -> None: ...
    visibilityChanged: typing.ClassVar[QtCore.pyqtSignal]
    topLevelChanged: typing.ClassVar[QtCore.pyqtSignal]
    toolButtonStyleChanged: typing.ClassVar[QtCore.pyqtSignal]
    iconSizeChanged: typing.ClassVar[QtCore.pyqtSignal]
    orientationChanged: typing.ClassVar[QtCore.pyqtSignal]
    allowedAreasChanged: typing.ClassVar[QtCore.pyqtSignal]
    movableChanged: typing.ClassVar[QtCore.pyqtSignal]
    actionTriggered: typing.ClassVar[QtCore.pyqtSignal]
    def setToolButtonStyle(self, toolButtonStyle: QtCore.Qt.ToolButtonStyle) -> None: ...
    def setIconSize(self, iconSize: QtCore.QSize) -> None: ...
    def widgetForAction(self, action: typing.Optional[QAction]) -> typing.Optional[QWidget]: ...
    def toolButtonStyle(self) -> QtCore.Qt.ToolButtonStyle: ...
    def iconSize(self) -> QtCore.QSize: ...
    def toggleViewAction(self) -> typing.Optional[QAction]: ...
    @typing.overload
    def actionAt(self, p: QtCore.QPoint) -> typing.Optional[QAction]: ...
    @typing.overload
    def actionAt(self, ax: int, ay: int) -> typing.Optional[QAction]: ...
    def actionGeometry(self, action: typing.Optional[QAction]) -> QtCore.QRect: ...
    def insertWidget(self, before: typing.Optional[QAction], widget: typing.Optional[QWidget]) -> typing.Optional[QAction]: ...
    def addWidget(self, widget: typing.Optional[QWidget]) -> typing.Optional[QAction]: ...
    def insertSeparator(self, before: typing.Optional[QAction]) -> typing.Optional[QAction]: ...
    def addSeparator(self) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, action: typing.Optional[QAction]) -> None: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, icon: QtGui.QIcon, text: typing.Optional[str]) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, text: typing.Optional[str], slot: PYQT_SLOT) -> typing.Optional[QAction]: ...
    @typing.overload
    def addAction(self, icon: QtGui.QIcon, text: typing.Optional[str], slot: PYQT_SLOT) -> typing.Optional[QAction]: ...
    def clear(self) -> None: ...
    def orientation(self) -> QtCore.Qt.Orientation: ...
    def setOrientation(self, orientation: QtCore.Qt.Orientation) -> None: ...
    def isAreaAllowed(self, area: QtCore.Qt.ToolBarArea) -> bool: ...
    def allowedAreas(self) -> QtCore.Qt.ToolBarAreas: ...
    def setAllowedAreas(self, areas: typing.Union[QtCore.Qt.ToolBarAreas, QtCore.Qt.ToolBarArea]) -> None: ...
    def isMovable(self) -> bool: ...
    def setMovable(self, movable: bool) -> None: ...


class QToolBox(QFrame):

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def showEvent(self, e: typing.Optional[QtGui.QShowEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def itemRemoved(self, index: int) -> None: ...
    def itemInserted(self, index: int) -> None: ...
    currentChanged: typing.ClassVar[QtCore.pyqtSignal]
    def setCurrentWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def setCurrentIndex(self, index: int) -> None: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def indexOf(self, widget: typing.Optional[QWidget]) -> int: ...
    def widget(self, index: int) -> typing.Optional[QWidget]: ...
    def currentWidget(self) -> typing.Optional[QWidget]: ...
    def currentIndex(self) -> int: ...
    def itemToolTip(self, index: int) -> str: ...
    def setItemToolTip(self, index: int, toolTip: typing.Optional[str]) -> None: ...
    def itemIcon(self, index: int) -> QtGui.QIcon: ...
    def setItemIcon(self, index: int, icon: QtGui.QIcon) -> None: ...
    def itemText(self, index: int) -> str: ...
    def setItemText(self, index: int, text: typing.Optional[str]) -> None: ...
    def isItemEnabled(self, index: int) -> bool: ...
    def setItemEnabled(self, index: int, enabled: bool) -> None: ...
    def removeItem(self, index: int) -> None: ...
    @typing.overload
    def insertItem(self, index: int, item: typing.Optional[QWidget], text: typing.Optional[str]) -> int: ...
    @typing.overload
    def insertItem(self, index: int, widget: typing.Optional[QWidget], icon: QtGui.QIcon, text: typing.Optional[str]) -> int: ...
    @typing.overload
    def addItem(self, item: typing.Optional[QWidget], text: typing.Optional[str]) -> int: ...
    @typing.overload
    def addItem(self, item: typing.Optional[QWidget], iconSet: QtGui.QIcon, text: typing.Optional[str]) -> int: ...


class QToolButton(QAbstractButton):

    class ToolButtonPopupMode(int):
        DelayedPopup = ... # type: QToolButton.ToolButtonPopupMode
        MenuButtonPopup = ... # type: QToolButton.ToolButtonPopupMode
        InstantPopup = ... # type: QToolButton.ToolButtonPopupMode

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    def hitButton(self, pos: QtCore.QPoint) -> bool: ...
    def nextCheckState(self) -> None: ...
    def mouseReleaseEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def changeEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def timerEvent(self, a0: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def leaveEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def enterEvent(self, a0: typing.Optional[QtCore.QEvent]) -> None: ...
    def actionEvent(self, a0: typing.Optional[QtGui.QActionEvent]) -> None: ...
    def paintEvent(self, a0: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def mousePressEvent(self, a0: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def initStyleOption(self, option: typing.Optional[QStyleOptionToolButton]) -> None: ...
    triggered: typing.ClassVar[QtCore.pyqtSignal]
    def setDefaultAction(self, a0: typing.Optional[QAction]) -> None: ...
    def setToolButtonStyle(self, style: QtCore.Qt.ToolButtonStyle) -> None: ...
    def showMenu(self) -> None: ...
    def autoRaise(self) -> bool: ...
    def setAutoRaise(self, enable: bool) -> None: ...
    def defaultAction(self) -> typing.Optional[QAction]: ...
    def popupMode(self) -> 'QToolButton.ToolButtonPopupMode': ...
    def setPopupMode(self, mode: 'QToolButton.ToolButtonPopupMode') -> None: ...
    def menu(self) -> typing.Optional[QMenu]: ...
    def setMenu(self, menu: typing.Optional[QMenu]) -> None: ...
    def setArrowType(self, type: QtCore.Qt.ArrowType) -> None: ...
    def arrowType(self) -> QtCore.Qt.ArrowType: ...
    def toolButtonStyle(self) -> QtCore.Qt.ToolButtonStyle: ...
    def minimumSizeHint(self) -> QtCore.QSize: ...
    def sizeHint(self) -> QtCore.QSize: ...


class QToolTip(PyQt5.sipsimplewrapper):

    def __init__(self, a0: 'QToolTip') -> None: ...

    @staticmethod
    def text() -> str: ...
    @staticmethod
    def isVisible() -> bool: ...
    @staticmethod
    def setFont(a0: QtGui.QFont) -> None: ...
    @staticmethod
    def font() -> QtGui.QFont: ...
    @staticmethod
    def setPalette(a0: QtGui.QPalette) -> None: ...
    @staticmethod
    def hideText() -> None: ...
    @staticmethod
    def palette() -> QtGui.QPalette: ...
    @typing.overload
    @staticmethod
    def showText(pos: QtCore.QPoint, text: typing.Optional[str], widget: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    @staticmethod
    def showText(pos: QtCore.QPoint, text: typing.Optional[str], w: typing.Optional[QWidget], rect: QtCore.QRect) -> None:
...
    @typing.overload
    @staticmethod
    def showText(pos: QtCore.QPoint, text: typing.Optional[str], w: typing.Optional[QWidget], rect: QtCore.QRect,
msecShowTime: int) -> None: ...


class QTreeView(QAbstractItemView):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def expandRecursively(self, index: QtCore.QModelIndex, depth: int = ...) -> None: ...
    def resetIndentation(self) -> None: ...
    def viewportSizeHint(self) -> QtCore.QSize: ...
    def treePosition(self) -> int: ...
    def setTreePosition(self, logicalIndex: int) -> None: ...
    def setHeaderHidden(self, hide: bool) -> None: ...
    def isHeaderHidden(self) -> bool: ...
    def setExpandsOnDoubleClick(self, enable: bool) -> None: ...
    def expandsOnDoubleClick(self) -> bool: ...
    def currentChanged(self, current: QtCore.QModelIndex, previous: QtCore.QModelIndex) -> None: ...
    def selectionChanged(self, selected: QtCore.QItemSelection, deselected: QtCore.QItemSelection) -> None: ...
    def rowHeight(self, index: QtCore.QModelIndex) -> int: ...
    def viewportEvent(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def dragMoveEvent(self, event: typing.Optional[QtGui.QDragMoveEvent]) -> None: ...
    def expandToDepth(self, depth: int) -> None: ...
    def wordWrap(self) -> bool: ...
    def setWordWrap(self, on: bool) -> None: ...
    def setFirstColumnSpanned(self, row: int, parent: QtCore.QModelIndex, span: bool) -> None: ...
    def isFirstColumnSpanned(self, row: int, parent: QtCore.QModelIndex) -> bool: ...
    def setAutoExpandDelay(self, delay: int) -> None: ...
    def autoExpandDelay(self) -> int: ...
    def sortByColumn(self, column: int, order: QtCore.Qt.SortOrder) -> None: ...
    def allColumnsShowFocus(self) -> bool: ...
    def setAllColumnsShowFocus(self, enable: bool) -> None: ...
    def isAnimated(self) -> bool: ...
    def setAnimated(self, enable: bool) -> None: ...
    def isSortingEnabled(self) -> bool: ...
    def setSortingEnabled(self, enable: bool) -> None: ...
    def setColumnWidth(self, column: int, width: int) -> None: ...
    def isIndexHidden(self, index: QtCore.QModelIndex) -> bool: ...
    def horizontalScrollbarAction(self, action: int) -> None: ...
    def indexRowSizeHint(self, index: QtCore.QModelIndex) -> int: ...
    def sizeHintForColumn(self, column: int) -> int: ...
    def updateGeometries(self) -> None: ...
    def keyPressEvent(self, event: typing.Optional[QtGui.QKeyEvent]) -> None: ...
    def mouseDoubleClickEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mouseMoveEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def mousePressEvent(self, e: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def drawTree(self, painter: typing.Optional[QtGui.QPainter], region: QtGui.QRegion) -> None: ...
    def drawBranches(self, painter: typing.Optional[QtGui.QPainter], rect: QtCore.QRect, index: QtCore.QModelIndex) ->
None: ...
    def drawRow(self, painter: typing.Optional[QtGui.QPainter], options: QStyleOptionViewItem, index: QtCore.QModelIndex) -
> None: ...
    def mouseReleaseEvent(self, event: typing.Optional[QtGui.QMouseEvent]) -> None: ...
    def timerEvent(self, event: typing.Optional[QtCore.QTimerEvent]) -> None: ...
    def paintEvent(self, e: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def selectedIndexes(self) -> typing.List[QtCore.QModelIndex]: ...
    def visualRegionForSelection(self, selection: QtCore.QItemSelection) -> QtGui.QRegion: ...
    def setSelection(self, rect: QtCore.QRect, command: typing.Union[QtCore.QItemSelectionModel.SelectionFlags,
QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def verticalOffset(self) -> int: ...
    def horizontalOffset(self) -> int: ...
    def moveCursor(self, cursorAction: QAbstractItemView.CursorAction, modifiers: typing.Union[QtCore.Qt.KeyboardModifiers,
QtCore.Qt.KeyboardModifier]) -> QtCore.QModelIndex: ...
    def rowsAboutToBeRemoved(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def rowsInserted(self, parent: QtCore.QModelIndex, start: int, end: int) -> None: ...
    def scrollContentsBy(self, dx: int, dy: int) -> None: ...
    def rowsRemoved(self, parent: QtCore.QModelIndex, first: int, last: int) -> None: ...
    def reexpand(self) -> None: ...
    def columnMoved(self) -> None: ...
    def columnCountChanged(self, oldCount: int, newCount: int) -> None: ...
    def columnResized(self, column: int, oldSize: int, newSize: int) -> None: ...
    def selectAll(self) -> None: ...
    def resizeColumnToContents(self, column: int) -> None: ...
    def collapseAll(self) -> None: ...
    def collapse(self, index: QtCore.QModelIndex) -> None: ...
    def expandAll(self) -> None: ...
    def expand(self, index: QtCore.QModelIndex) -> None: ...
    def showColumn(self, column: int) -> None: ...
    def hideColumn(self, column: int) -> None: ...
    def dataChanged(self, topLeft: QtCore.QModelIndex, bottomRight: QtCore.QModelIndex, roles: typing.Iterable[int] = ...) -
> None: ...
    collapsed: typing.ClassVar[QtCore.pyqtSignal]
    expanded: typing.ClassVar[QtCore.pyqtSignal]
    def reset(self) -> None: ...
    def indexBelow(self, index: QtCore.QModelIndex) -> QtCore.QModelIndex: ...
    def indexAbove(self, index: QtCore.QModelIndex) -> QtCore.QModelIndex: ...
    def indexAt(self, p: QtCore.QPoint) -> QtCore.QModelIndex: ...
    def scrollTo(self, index: QtCore.QModelIndex, hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def visualRect(self, index: QtCore.QModelIndex) -> QtCore.QRect: ...
    def keyboardSearch(self, search: typing.Optional[str]) -> None: ...
    def setExpanded(self, index: QtCore.QModelIndex, expand: bool) -> None: ...
    def isExpanded(self, index: QtCore.QModelIndex) -> bool: ...
    def setRowHidden(self, row: int, parent: QtCore.QModelIndex, hide: bool) -> None: ...
    def isRowHidden(self, row: int, parent: QtCore.QModelIndex) -> bool: ...
    def setColumnHidden(self, column: int, hide: bool) -> None: ...
    def isColumnHidden(self, column: int) -> bool: ...
    def columnAt(self, x: int) -> int: ...
    def columnWidth(self, column: int) -> int: ...
    def columnViewportPosition(self, column: int) -> int: ...
    def setItemsExpandable(self, enable: bool) -> None: ...
    def itemsExpandable(self) -> bool: ...
    def setUniformRowHeights(self, uniform: bool) -> None: ...
    def uniformRowHeights(self) -> bool: ...
    def setRootIsDecorated(self, show: bool) -> None: ...
    def rootIsDecorated(self) -> bool: ...
    def setIndentation(self, i: int) -> None: ...
    def indentation(self) -> int: ...
    def setHeader(self, header: typing.Optional[QHeaderView]) -> None: ...
    def header(self) -> typing.Optional[QHeaderView]: ...
    def setSelectionModel(self, selectionModel: typing.Optional[QtCore.QItemSelectionModel]) -> None: ...
    def setRootIndex(self, index: QtCore.QModelIndex) -> None: ...
    def setModel(self, model: typing.Optional[QtCore.QAbstractItemModel]) -> None: ...


class QTreeWidgetItem(PyQt5.sip.wrapper):

    class ChildIndicatorPolicy(int):
        ShowIndicator = ... # type: QTreeWidgetItem.ChildIndicatorPolicy
        DontShowIndicator = ... # type: QTreeWidgetItem.ChildIndicatorPolicy
        DontShowIndicatorWhenChildless = ... # type: QTreeWidgetItem.ChildIndicatorPolicy

    class ItemType(int):
        Type = ... # type: QTreeWidgetItem.ItemType
        UserType = ... # type: QTreeWidgetItem.ItemType

    @typing.overload
    def __init__(self, type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, strings: typing.Iterable[typing.Optional[str]], type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional['QTreeWidget'], type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional['QTreeWidget'], strings: typing.Iterable[typing.Optional[str]], type: int = ...) ->
None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional['QTreeWidget'], preceding: typing.Optional['QTreeWidgetItem'], type: int = ...) -
> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional['QTreeWidgetItem'], type: int = ...) -> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional['QTreeWidgetItem'], strings: typing.Iterable[typing.Optional[str]], type: int = ...)
-> None: ...
    @typing.overload
    def __init__(self, parent: typing.Optional['QTreeWidgetItem'], preceding: typing.Optional['QTreeWidgetItem'], type: int =
...) -> None: ...
    @typing.overload
    def __init__(self, other: 'QTreeWidgetItem') -> None: ...

    def __ge__(self, other: 'QTreeWidgetItem') -> bool: ...
    def emitDataChanged(self) -> None: ...
    def isDisabled(self) -> bool: ...
    def setDisabled(self, disabled: bool) -> None: ...
    def isFirstColumnSpanned(self) -> bool: ...
    def setFirstColumnSpanned(self, aspan: bool) -> None: ...
    def removeChild(self, child: typing.Optional['QTreeWidgetItem']) -> None: ...
    def childIndicatorPolicy(self) -> 'QTreeWidgetItem.ChildIndicatorPolicy': ...
    def setChildIndicatorPolicy(self, policy: 'QTreeWidgetItem.ChildIndicatorPolicy') -> None: ...
    def isExpanded(self) -> bool: ...
    def setExpanded(self, aexpand: bool) -> None: ...
    def isHidden(self) -> bool: ...
    def setHidden(self, ahide: bool) -> None: ...
    def isSelected(self) -> bool: ...
    def setSelected(self, aselect: bool) -> None: ...
    def sortChildren(self, column: int, order: QtCore.Qt.SortOrder) -> None: ...
    def setForeground(self, column: int, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], QtGui.QGradient]) -> None: ...
    def foreground(self, column: int) -> QtGui.QBrush: ...
    def setBackground(self, column: int, brush: typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], QtGui.QGradient]) -> None: ...
    def background(self, column: int) -> QtGui.QBrush: ...
    def takeChildren(self) -> typing.List['QTreeWidgetItem']: ...
    def insertChildren(self, index: int, children: typing.Iterable['QTreeWidgetItem']) -> None: ...
    def addChildren(self, children: typing.Iterable['QTreeWidgetItem']) -> None: ...
    def setSizeHint(self, column: int, size: QtCore.QSize) -> None: ...
    def sizeHint(self, column: int) -> QtCore.QSize: ...
    def indexOfChild(self, achild: typing.Optional['QTreeWidgetItem']) -> int: ...
    def setFont(self, column: int, afont: QtGui.QFont) -> None: ...
    def setWhatsThis(self, column: int, awhatsThis: typing.Optional[str]) -> None: ...
    def setToolTip(self, column: int, atoolTip: typing.Optional[str]) -> None: ...
    def setStatusTip(self, column: int, astatusTip: typing.Optional[str]) -> None: ...
    def setIcon(self, column: int, aicon: QtGui.QIcon) -> None: ...
    def setText(self, column: int, atext: typing.Optional[str]) -> None: ...
    def setFlags(self, aflags: typing.Union[QtCore.Qt.ItemFlags, QtCore.Qt.ItemFlag]) -> None: ...
    def type(self) -> int: ...
    def takeChild(self, index: int) -> typing.Optional['QTreeWidgetItem']: ...
    def insertChild(self, index: int, child: typing.Optional['QTreeWidgetItem']) -> None: ...
    def addChild(self, child: typing.Optional['QTreeWidgetItem']) -> None: ...
    def columnCount(self) -> int: ...
    def childCount(self) -> int: ...
    def child(self, index: int) -> typing.Optional['QTreeWidgetItem']: ...
    def parent(self) -> typing.Optional['QTreeWidgetItem']: ...
    def write(self, out: QtCore.QDataStream) -> None: ...
    def read(self, in_: QtCore.QDataStream) -> None: ...
    def __lt__(self, other: 'QTreeWidgetItem') -> bool: ...
    def setData(self, column: int, role: int, value: typing.Any) -> None: ...
    def data(self, column: int, role: int) -> typing.Any: ...
    def setCheckState(self, column: int, state: QtCore.Qt.CheckState) -> None: ...
    def checkState(self, column: int) -> QtCore.Qt.CheckState: ...
    def setTextAlignment(self, column: int, alignment: int) -> None: ...
    def textAlignment(self, column: int) -> int: ...
    def font(self, column: int) -> QtGui.QFont: ...
    def whatsThis(self, column: int) -> str: ...
    def toolTip(self, column: int) -> str: ...
    def statusTip(self, column: int) -> str: ...
    def icon(self, column: int) -> QtGui.QIcon: ...
    def text(self, column: int) -> str: ...
    def flags(self) -> QtCore.Qt.ItemFlags: ...
    def treeWidget(self) -> typing.Optional['QTreeWidget']: ...
    def clone(self) -> typing.Optional['QTreeWidgetItem']: ...


class QTreeWidget(QTreeView):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def isPersistentEditorOpen(self, item: typing.Optional[QTreeWidgetItem], column: int = ...) -> bool: ...
    def setSelectionModel(self, selectionModel: typing.Optional[QtCore.QItemSelectionModel]) -> None: ...
    def removeItemWidget(self, item: typing.Optional[QTreeWidgetItem], column: int) -> None: ...
    def itemBelow(self, item: typing.Optional[QTreeWidgetItem]) -> typing.Optional[QTreeWidgetItem]: ...
    def itemAbove(self, item: typing.Optional[QTreeWidgetItem]) -> typing.Optional[QTreeWidgetItem]: ...
    def setFirstItemColumnSpanned(self, item: typing.Optional[QTreeWidgetItem], span: bool) -> None: ...
    def isFirstItemColumnSpanned(self, item: typing.Optional[QTreeWidgetItem]) -> bool: ...
    def setHeaderLabel(self, alabel: typing.Optional[str]) -> None: ...
    def invisibleRootItem(self) -> typing.Optional[QTreeWidgetItem]: ...
    def dropEvent(self, event: typing.Optional[QtGui.QDropEvent]) -> None: ...
    def event(self, e: typing.Optional[QtCore.QEvent]) -> bool: ...
    def itemFromIndex(self, index: QtCore.QModelIndex) -> typing.Optional[QTreeWidgetItem]: ...
    def indexFromItem(self, item: typing.Optional[QTreeWidgetItem], column: int = ...) -> QtCore.QModelIndex: ...
    def supportedDropActions(self) -> QtCore.Qt.DropActions: ...
    def dropMimeData(self, parent: typing.Optional[QTreeWidgetItem], index: int, data: typing.Optional[QtCore.QMimeData], 
action: QtCore.Qt.DropAction) -> bool: ...
    def mimeData(self, items: typing.Iterable[QTreeWidgetItem]) -> typing.Optional[QtCore.QMimeData]: ...
    def mimeTypes(self) -> typing.List[str]: ...
    itemSelectionChanged: typing.ClassVar[QtCore.pyqtSignal]
    currentItemChanged: typing.ClassVar[QtCore.pyqtSignal]
    itemCollapsed: typing.ClassVar[QtCore.pyqtSignal]
    itemExpanded: typing.ClassVar[QtCore.pyqtSignal]
    itemChanged: typing.ClassVar[QtCore.pyqtSignal]
    itemEntered: typing.ClassVar[QtCore.pyqtSignal]
    itemActivated: typing.ClassVar[QtCore.pyqtSignal]
    itemDoubleClicked: typing.ClassVar[QtCore.pyqtSignal]
    itemClicked: typing.ClassVar[QtCore.pyqtSignal]
    itemPressed: typing.ClassVar[QtCore.pyqtSignal]
    def clear(self) -> None: ...
    def collapseItem(self, item: typing.Optional[QTreeWidgetItem]) -> None: ...
    def expandItem(self, item: typing.Optional[QTreeWidgetItem]) -> None: ...
    def scrollToItem(self, item: typing.Optional[QTreeWidgetItem], hint: QAbstractItemView.ScrollHint = ...) -> None: ...
    def findItems(self, text: typing.Optional[str], flags: typing.Union[QtCore.Qt.MatchFlags, QtCore.Qt.MatchFlag], column: int 
= ...) -> typing.List[QTreeWidgetItem]: ...
    def selectedItems(self) -> typing.List[QTreeWidgetItem]: ...
    def setItemWidget(self, item: typing.Optional[QTreeWidgetItem], column: int, widget: typing.Optional[QWidget]) -> None: 
...
    def itemWidget(self, item: typing.Optional[QTreeWidgetItem], column: int) -> typing.Optional[QWidget]: ...
    def closePersistentEditor(self, item: typing.Optional[QTreeWidgetItem], column: int = ...) -> None: ...
    def openPersistentEditor(self, item: typing.Optional[QTreeWidgetItem], column: int = ...) -> None: ...
    def editItem(self, item: typing.Optional[QTreeWidgetItem], column: int = ...) -> None: ...
    def sortItems(self, column: int, order: QtCore.Qt.SortOrder) -> None: ...
    def sortColumn(self) -> int: ...
    def visualItemRect(self, item: typing.Optional[QTreeWidgetItem]) -> QtCore.QRect: ...
    @typing.overload
    def itemAt(self, p: QtCore.QPoint) -> typing.Optional[QTreeWidgetItem]: ...
    @typing.overload
    def itemAt(self, ax: int, ay: int) -> typing.Optional[QTreeWidgetItem]: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QTreeWidgetItem]) -> None: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QTreeWidgetItem], column: int) -> None: ...
    @typing.overload
    def setCurrentItem(self, item: typing.Optional[QTreeWidgetItem], column: int, command: 
typing.Union[QtCore.QItemSelectionModel.SelectionFlags, QtCore.QItemSelectionModel.SelectionFlag]) -> None: ...
    def currentColumn(self) -> int: ...
    def currentItem(self) -> typing.Optional[QTreeWidgetItem]: ...
    def setHeaderLabels(self, labels: typing.Iterable[typing.Optional[str]]) -> None: ...
    def setHeaderItem(self, item: typing.Optional[QTreeWidgetItem]) -> None: ...
    def headerItem(self) -> typing.Optional[QTreeWidgetItem]: ...
    def addTopLevelItems(self, items: typing.Iterable[QTreeWidgetItem]) -> None: ...
    def insertTopLevelItems(self, index: int, items: typing.Iterable[QTreeWidgetItem]) -> None: ...
    def indexOfTopLevelItem(self, item: typing.Optional[QTreeWidgetItem]) -> int: ...
    def takeTopLevelItem(self, index: int) -> typing.Optional[QTreeWidgetItem]: ...
    def addTopLevelItem(self, item: typing.Optional[QTreeWidgetItem]) -> None: ...
    def insertTopLevelItem(self, index: int, item: typing.Optional[QTreeWidgetItem]) -> None: ...
    def topLevelItemCount(self) -> int: ...
    def topLevelItem(self, index: int) -> typing.Optional[QTreeWidgetItem]: ...
    def setColumnCount(self, columns: int) -> None: ...
    def columnCount(self) -> int: ...


class QTreeWidgetItemIterator(PyQt5.sipsimplewrapper):

    class IteratorFlag(int):
        All = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Hidden = ... # type: QTreeWidgetItemIterator.IteratorFlag
        NotHidden = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Selected = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Unselected = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Selectable = ... # type: QTreeWidgetItemIterator.IteratorFlag
        NotSelectable = ... # type: QTreeWidgetItemIterator.IteratorFlag
        DragEnabled = ... # type: QTreeWidgetItemIterator.IteratorFlag
        DragDisabled = ... # type: QTreeWidgetItemIterator.IteratorFlag
        DropEnabled = ... # type: QTreeWidgetItemIterator.IteratorFlag
        DropDisabled = ... # type: QTreeWidgetItemIterator.IteratorFlag
        HasChildren = ... # type: QTreeWidgetItemIterator.IteratorFlag
        NoChildren = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Checked = ... # type: QTreeWidgetItemIterator.IteratorFlag
        NotChecked = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Enabled = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Disabled = ... # type: QTreeWidgetItemIterator.IteratorFlag
        Editable = ... # type: QTreeWidgetItemIterator.IteratorFlag
        NotEditable = ... # type: QTreeWidgetItemIterator.IteratorFlag
        UserFlag = ... # type: QTreeWidgetItemIterator.IteratorFlag

    class IteratorFlags(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
'QTreeWidgetItemIterator.IteratorFlags': ...
        def __xor__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
'QTreeWidgetItemIterator.IteratorFlags': ...
        def __ior__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
'QTreeWidgetItemIterator.IteratorFlags': ...
        def __or__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
'QTreeWidgetItemIterator.IteratorFlags': ...
        def __iand__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
'QTreeWidgetItemIterator.IteratorFlags': ...
        def __and__(self, f: typing.Union['QTreeWidgetItemIterator.IteratorFlags', 'QTreeWidgetItemIterator.IteratorFlag']) -> 
'QTreeWidgetItemIterator.IteratorFlags': ...
        def __invert__(self) -> 'QTreeWidgetItemIterator.IteratorFlags': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    @typing.overload
    def __init__(self, it: 'QTreeWidgetItemIterator') -> None: ...
    @typing.overload
    def __init__(self, widget: typing.Optional[QTreeWidget], flags: 'QTreeWidgetItemIterator.IteratorFlags' = ...) -> None: ...
    @typing.overload
    def __init__(self, item: typing.Optional[QTreeWidgetItem], flags: 'QTreeWidgetItemIterator.IteratorFlags' = ...) -> None: 
...

    def __isub__(self, n: int) -> 'QTreeWidgetItemIterator': ...
    def __iadd__(self, n: int) -> 'QTreeWidgetItemIterator': ...
    def value(self) -> typing.Optional[QTreeWidgetItem]: ...


class QUndoGroup(QtCore.QObject):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    undoTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    redoTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    indexChanged: typing.ClassVar[QtCore.pyqtSignal]
    cleanChanged: typing.ClassVar[QtCore.pyqtSignal]
    canUndoChanged: typing.ClassVar[QtCore.pyqtSignal]
    canRedoChanged: typing.ClassVar[QtCore.pyqtSignal]
    activeStackChanged: typing.ClassVar[QtCore.pyqtSignal]
    def undo(self) -> None: ...
    def setActiveStack(self, stack: typing.Optional['QUndoStack']) -> None: ...
    def redo(self) -> None: ...
    def isClean(self) -> bool: ...
    def redoText(self) -> str: ...
    def undoText(self) -> str: ...
    def canRedo(self) -> bool: ...
    def canUndo(self) -> bool: ...
    def createUndoAction(self, parent: typing.Optional[QtCore.QObject], prefix: typing.Optional[str] = ...) ->
typing.Optional[QAction]: ...
    def createRedoAction(self, parent: typing.Optional[QtCore.QObject], prefix: typing.Optional[str] = ...) ->
typing.Optional[QAction]: ...
    def activeStack(self) -> typing.Optional['QUndoStack']: ...
    def stacks(self) -> typing.List['QUndoStack']: ...
    def removeStack(self, stack: typing.Optional['QUndoStack']) -> None: ...
    def addStack(self, stack: typing.Optional['QUndoStack']) -> None: ...


class QUndoCommand(PyQt5.sip.wrapper):

    @typing.overload
    def __init__(self, parent: typing.Optional['QUndoCommand'] = ...) -> None: ...
    @typing.overload
    def __init__(self, text: typing.Optional[str], parent: typing.Optional['QUndoCommand'] = ...) -> None: ...

    def setObsolete(self, obsolete: bool) -> None: ...
    def isObsolete(self) -> bool: ...
    def actionText(self) -> str: ...
    def child(self, index: int) -> typing.Optional['QUndoCommand']: ...
    def childCount(self) -> int: ...
    def undo(self) -> None: ...
    def text(self) -> str: ...
    def setText(self, text: typing.Optional[str]) -> None: ...
    def redo(self) -> None: ...
    def mergeWith(self, other: typing.Optional['QUndoCommand']) -> bool: ...
    def id(self) -> int: ...


class QUndoStack(QtCore.QObject):

    def __init__(self, parent: typing.Optional[QtCore.QObject] = ...) -> None: ...

    def command(self, index: int) -> typing.Optional[QUndoCommand]: ...
    def undoLimit(self) -> int: ...
    def setUndoLimit(self, limit: int) -> None: ...
    undoTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    redoTextChanged: typing.ClassVar[QtCore.pyqtSignal]
    indexChanged: typing.ClassVar[QtCore.pyqtSignal]
    cleanChanged: typing.ClassVar[QtCore.pyqtSignal]
    canUndoChanged: typing.ClassVar[QtCore.pyqtSignal]
    canRedoChanged: typing.ClassVar[QtCore.pyqtSignal]
    def resetClean(self) -> None: ...
    def undo(self) -> None: ...
    def setIndex(self, idx: int) -> None: ...
    def setClean(self) -> None: ...
    def setActive(self, active: bool = ...) -> None: ...
    def redo(self) -> None: ...
    def endMacro(self) -> None: ...
    def beginMacro(self, text: typing.Optional[str]) -> None: ...
    def cleanIndex(self) -> int: ...
    def isClean(self) -> bool: ...
    def isActive(self) -> bool: ...
    def createRedoAction(self, parent: typing.Optional[QtCore.QObject], prefix: typing.Optional[str] = ...) ->
typing.Optional[QAction]: ...
    def createUndoAction(self, parent: typing.Optional[QtCore.QObject], prefix: typing.Optional[str] = ...) ->
typing.Optional[QAction]: ...
    def text(self, idx: int) -> str: ...
    def index(self) -> int: ...
    def __len__(self) -> int: ...
    def count(self) -> int: ...
    def redoText(self) -> str: ...
    def undoText(self) -> str: ...
    def canRedo(self) -> bool: ...
    def canUndo(self) -> bool: ...
    def push(self, cmd: typing.Optional[QUndoCommand]) -> None: ...
    def clear(self) -> None: ...


class QUndoView(QListView):

    @typing.overload
    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, stack: typing.Optional[QUndoStack], parent: typing.Optional[QWidget] = ...) -> None: ...
    @typing.overload
    def __init__(self, group: typing.Optional[QUndoGroup], parent: typing.Optional[QWidget] = ...) -> None: ...

    def setGroup(self, group: typing.Optional[QUndoGroup]) -> None: ...
    def setStack(self, stack: typing.Optional[QUndoStack]) -> None: ...
    def cleanIcon(self) -> QtGui.QIcon: ...
    def setCleanIcon(self, icon: QtGui.QIcon) -> None: ...
    def emptyLabel(self) -> str: ...
    def setEmptyLabel(self, label: typing.Optional[str]) -> None: ...
    def group(self) -> typing.Optional[QUndoGroup]: ...
    def stack(self) -> typing.Optional[QUndoStack]: ...


class QWhatsThis(PyQt5.sipsimplewrapper):

    def __init__(self, a0: 'QWhatsThis') -> None: ...

    @staticmethod
    def createAction(parent: typing.Optional[QtCore.QObject] = ...) -> typing.Optional[QAction]: ...
    @staticmethod
    def hideText() -> None: ...
    @staticmethod
    def showText(pos: QtCore.QPoint, text: typing.Optional[str], widget: typing.Optional[QWidget] = ...) -> None: ...
    @staticmethod
    def leaveWhatsThisMode() -> None: ...
    @staticmethod
    def inWhatsThisMode() -> bool: ...
    @staticmethod
    def enterWhatsThisMode() -> None: ...


class QWidgetAction(QAction):

    def __init__(self, parent: typing.Optional[QtCore.QObject]) -> None: ...

    def createdWidgets(self) -> typing.List[QWidget]: ...
    def deleteWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def createWidget(self, parent: typing.Optional[QWidget]) -> typing.Optional[QWidget]: ...
    def eventFilter(self, a0: typing.Optional[QtCore.QObject], a1: typing.Optional[QtCore.QEvent]) -> bool: ...
    def event(self, a0: typing.Optional[QtCore.QEvent]) -> bool: ...
    def releaseWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def requestWidget(self, parent: typing.Optional[QWidget]) -> typing.Optional[QWidget]: ...
    def defaultWidget(self) -> typing.Optional[QWidget]: ...
    def setDefaultWidget(self, w: typing.Optional[QWidget]) -> None: ...


class QWizard(QDialog):

    class WizardOption(int):
        IndependentPages = ... # type: QWizard.WizardOption
        IgnoreSubTitles = ... # type: QWizard.WizardOption
        ExtendedWatermarkPixmap = ... # type: QWizard.WizardOption
        NoDefaultButton = ... # type: QWizard.WizardOption
        NoBackButtonOnStartPage = ... # type: QWizard.WizardOption
        NoBackButtonOnLastPage = ... # type: QWizard.WizardOption
        DisabledBackButtonOnLastPage = ... # type: QWizard.WizardOption
        HaveNextButtonOnLastPage = ... # type: QWizard.WizardOption
        HaveFinishButtonOnEarlyPages = ... # type: QWizard.WizardOption
        NoCancelButton = ... # type: QWizard.WizardOption
        CancelButtonOnLeft = ... # type: QWizard.WizardOption
        HaveHelpButton = ... # type: QWizard.WizardOption
        HelpButtonOnRight = ... # type: QWizard.WizardOption
        HaveCustomButton1 = ... # type: QWizard.WizardOption
        HaveCustomButton2 = ... # type: QWizard.WizardOption
        HaveCustomButton3 = ... # type: QWizard.WizardOption
        NoCancelButtonOnLastPage = ... # type: QWizard.WizardOption

    class WizardStyle(int):
        ClassicStyle = ... # type: QWizard.WizardStyle
        ModernStyle = ... # type: QWizard.WizardStyle
        MacStyle = ... # type: QWizard.WizardStyle
        AeroStyle = ... # type: QWizard.WizardStyle

    class WizardPixmap(int):
        WatermarkPixmap = ... # type: QWizard.WizardPixmap
        LogoPixmap = ... # type: QWizard.WizardPixmap
        BannerPixmap = ... # type: QWizard.WizardPixmap
        BackgroundPixmap = ... # type: QWizard.WizardPixmap

    class WizardButton(int):
        BackButton = ... # type: QWizard.WizardButton
        NextButton = ... # type: QWizard.WizardButton
        CommitButton = ... # type: QWizard.WizardButton
        FinishButton = ... # type: QWizard.WizardButton
        CancelButton = ... # type: QWizard.WizardButton
        HelpButton = ... # type: QWizard.WizardButton
        CustomButton1 = ... # type: QWizard.WizardButton
        CustomButton2 = ... # type: QWizard.WizardButton
        CustomButton3 = ... # type: QWizard.WizardButton
        Stretch = ... # type: QWizard.WizardButton

    class WizardOptions(PyQt5.sipsimplewrapper):

        @typing.overload
        def __init__(self) -> None: ...
        @typing.overload
        def __init__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> None: ...

        def __hash__(self) -> int: ...
        def __bool__(self) -> int: ...
        def __ne__(self, other: object): ...
        def __eq__(self, other: object): ...
        def __ixor__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> 'QWizard.WizardOptions': ...
        def __xor__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> 'QWizard.WizardOptions': ...
        def __ior__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> 'QWizard.WizardOptions': ...
        def __or__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> 'QWizard.WizardOptions': ...
        def __iand__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> 'QWizard.WizardOptions': ...
        def __and__(self, f: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> 'QWizard.WizardOptions': ...
        def __invert__(self) -> 'QWizard.WizardOptions': ...
        def __index__(self) -> int: ...
        def __int__(self) -> int: ...

    def __init__(self, parent: typing.Optional[QWidget] = ..., flags: typing.Union[QtCore.Qt.WindowFlags,
QtCore.Qt.WindowType] = ...) -> None: ...

    def visitedIds(self) -> typing.List[int]: ...
    pageRemoved: typing.ClassVar[QtCore.pyqtSignal]
    pageAdded: typing.ClassVar[QtCore.pyqtSignal]
    def sideWidget(self) -> typing.Optional[QWidget]: ...
    def setSideWidget(self, widget: typing.Optional[QWidget]) -> None: ...
    def pageIds(self) -> typing.List[int]: ...
    def removePage(self, id: int) -> None: ...
    def cleanupPage(self, id: int) -> None: ...
    def initializePage(self, id: int) -> None: ...
    def done(self, result: int) -> None: ...
    def paintEvent(self, event: typing.Optional[QtGui.QPaintEvent]) -> None: ...
    def resizeEvent(self, event: typing.Optional[QtGui.QResizeEvent]) -> None: ...
    def event(self, event: typing.Optional[QtCore.QEvent]) -> bool: ...
    def restart(self) -> None: ...
    def next(self) -> None: ...
    def back(self) -> None: ...
    customButtonClicked: typing.ClassVar[QtCore.pyqtSignal]
    helpRequested: typing.ClassVar[QtCore.pyqtSignal]
    currentIdChanged: typing.ClassVar[QtCore.pyqtSignal]
    def sizeHint(self) -> QtCore.QSize: ...
    def setVisible(self, visible: bool) -> None: ...
    def setDefaultProperty(self, className: typing.Optional[str], property: typing.Optional[str], changedSignal: PYQT_SIGNAL)
-> None: ...
    def pixmap(self, which: 'QWizard.WizardPixmap') -> QtGui.QPixmap: ...
    def setPixmap(self, which: 'QWizard.WizardPixmap', pixmap: QtGui.QPixmap) -> None: ...
    def subTitleFormat(self) -> QtCore.Qt.TextFormat: ...
    def setSubTitleFormat(self, format: QtCore.Qt.TextFormat) -> None: ...
    def titleFormat(self) -> QtCore.Qt.TextFormat: ...
    def setTitleFormat(self, format: QtCore.Qt.TextFormat) -> None: ...
    def button(self, which: 'QWizard.WizardButton') -> typing.Optional[QAbstractButton]: ...
    def setButton(self, which: 'QWizard.WizardButton', button: typing.Optional[QAbstractButton]) -> None: ...
    def setButtonLayout(self, layout: typing.Iterable['QWizard.WizardButton']) -> None: ...
    def buttonText(self, which: 'QWizard.WizardButton') -> str: ...
    def setButtonText(self, which: 'QWizard.WizardButton', text: typing.Optional[str]) -> None: ...
    def options(self) -> 'QWizard.WizardOptions': ...
    def setOptions(self, options: typing.Union['QWizard.WizardOptions', 'QWizard.WizardOption']) -> None: ...
    def testOption(self, option: 'QWizard.WizardOption') -> bool: ...
    def setOption(self, option: 'QWizard.WizardOption', on: bool = ...) -> None: ...
    def wizardStyle(self) -> 'QWizard.WizardStyle': ...
    def setWizardStyle(self, style: 'QWizard.WizardStyle') -> None: ...
    def field(self, name: typing.Optional[str]) -> typing.Any: ...
    def setField(self, name: typing.Optional[str], value: typing.Any) -> None: ...
    def nextId(self) -> int: ...
    def validateCurrentPage(self) -> bool: ...
    def currentId(self) -> int: ...
    def currentPage(self) -> typing.Optional['QWizardPage']: ...
    def startId(self) -> int: ...
    def setStartId(self, id: int) -> None: ...
    def visitedPages(self) -> typing.List[int]: ...
    def hasVisitedPage(self, id: int) -> bool: ...
    def page(self, id: int) -> typing.Optional['QWizardPage']: ...
    def setPage(self, id: int, page: typing.Optional['QWizardPage']) -> None: ...
    def addPage(self, page: typing.Optional['QWizardPage']) -> int: ...


class QWizardPage(QWidget):

    def __init__(self, parent: typing.Optional[QWidget] = ...) -> None: ...

    def wizard(self) -> typing.Optional[QWizard]: ...
    def registerField(self, name: typing.Optional[str], widget: typing.Optional[QWidget], property: typing.Optional[str] = ...,
changedSignal: PYQT_SIGNAL = ...) -> None: ...
    def field(self, name: typing.Optional[str]) -> typing.Any: ...
    def setField(self, name: typing.Optional[str], value: typing.Any) -> None: ...
    completeChanged: typing.ClassVar[QtCore.pyqtSignal]
    def nextId(self) -> int: ...
    def isComplete(self) -> bool: ...
    def validatePage(self) -> bool: ...
    def cleanupPage(self) -> None: ...
    def initializePage(self) -> None: ...
    def buttonText(self, which: QWizard.WizardButton) -> str: ...
    def setButtonText(self, which: QWizard.WizardButton, text: typing.Optional[str]) -> None: ...
    def isCommitPage(self) -> bool: ...
    def setCommitPage(self, commitPage: bool) -> None: ...
    def isFinalPage(self) -> bool: ...
    def setFinalPage(self, finalPage: bool) -> None: ...
    def pixmap(self, which: QWizard.WizardPixmap) -> QtGui.QPixmap: ...
    def setPixmap(self, which: QWizard.WizardPixmap, pixmap: QtGui.QPixmap) -> None: ...
    def subTitle(self) -> str: ...
    def setSubTitle(self, subTitle: typing.Optional[str]) -> None: ...
    def title(self) -> str: ...
    def setTitle(self, title: typing.Optional[str]) -> None: ...


QWIDGETSIZE_MAX = ... # type: int
qApp = ... # type: QApplication


def qDrawBorderPixmap(painter: typing.Optional[QtGui.QPainter], target: QtCore.QRect, margins: QtCore.QMargins, pixmap:
QtGui.QPixmap) -> None: ...
@typing.overload
def qDrawPlainRect(p: typing.Optional[QtGui.QPainter], x: int, y: int, w: int, h: int, a5: typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], lineWidth: int = ..., fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...) -> None: ...
@typing.overload
def qDrawPlainRect(p: typing.Optional[QtGui.QPainter], r: QtCore.QRect, a2: typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], lineWidth: int = ..., fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...) -> None: ...
@typing.overload
def qDrawWinPanel(p: typing.Optional[QtGui.QPainter], x: int, y: int, w: int, h: int, pal: QtGui.QPalette, sunken: bool = ...,
fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...)
-> None: ...
@typing.overload
def qDrawWinPanel(p: typing.Optional[QtGui.QPainter], r: QtCore.QRect, pal: QtGui.QPalette, sunken: bool = ..., fill:
typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...) ->
None: ...
@typing.overload
def qDrawWinButton(p: typing.Optional[QtGui.QPainter], x: int, y: int, w: int, h: int, pal: QtGui.QPalette, sunken: bool = ...,
fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...)
-> None: ...
@typing.overload
def qDrawWinButton(p: typing.Optional[QtGui.QPainter], r: QtCore.QRect, pal: QtGui.QPalette, sunken: bool = ..., fill:
typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...) ->
None: ...
@typing.overload
def qDrawShadePanel(p: typing.Optional[QtGui.QPainter], x: int, y: int, w: int, h: int, pal: QtGui.QPalette, sunken: bool = ...,
lineWidth: int = ..., fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]] = ...) -> None: ...
@typing.overload
def qDrawShadePanel(p: typing.Optional[QtGui.QPainter], r: QtCore.QRect, pal: QtGui.QPalette, sunken: bool = ..., lineWidth:
int = ..., fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor, QtCore.Qt.GlobalColor],
QtGui.QGradient]] = ...) -> None: ...
@typing.overload
def qDrawShadeRect(p: typing.Optional[QtGui.QPainter], x: int, y: int, w: int, h: int, pal: QtGui.QPalette, sunken: bool = ...,
lineWidth: int = ..., midLineWidth: int = ..., fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...) -> None: ...
@typing.overload
def qDrawShadeRect(p: typing.Optional[QtGui.QPainter], r: QtCore.QRect, pal: QtGui.QPalette, sunken: bool = ..., lineWidth:
int = ..., midLineWidth: int = ..., fill: typing.Optional[typing.Union[QtGui.QBrush, typing.Union[QtGui.QColor,
QtCore.Qt.GlobalColor], QtGui.QGradient]] = ...) -> None: ...
@typing.overload
def qDrawShadeLine(p: typing.Optional[QtGui.QPainter], x1: int, y1: int, x2: int, y2: int, pal: QtGui.QPalette, sunken: bool =
..., lineWidth: int = ..., midLineWidth: int = ...) -> None: ...
@typing.overload
def qDrawShadeLine(p: typing.Optional[QtGui.QPainter], p1: QtCore.QPoint, p2: QtCore.QPoint, pal: QtGui.QPalette, sunken:
bool = ..., lineWidth: int = ..., midLineWidth: int = ...) -> None: ...
```