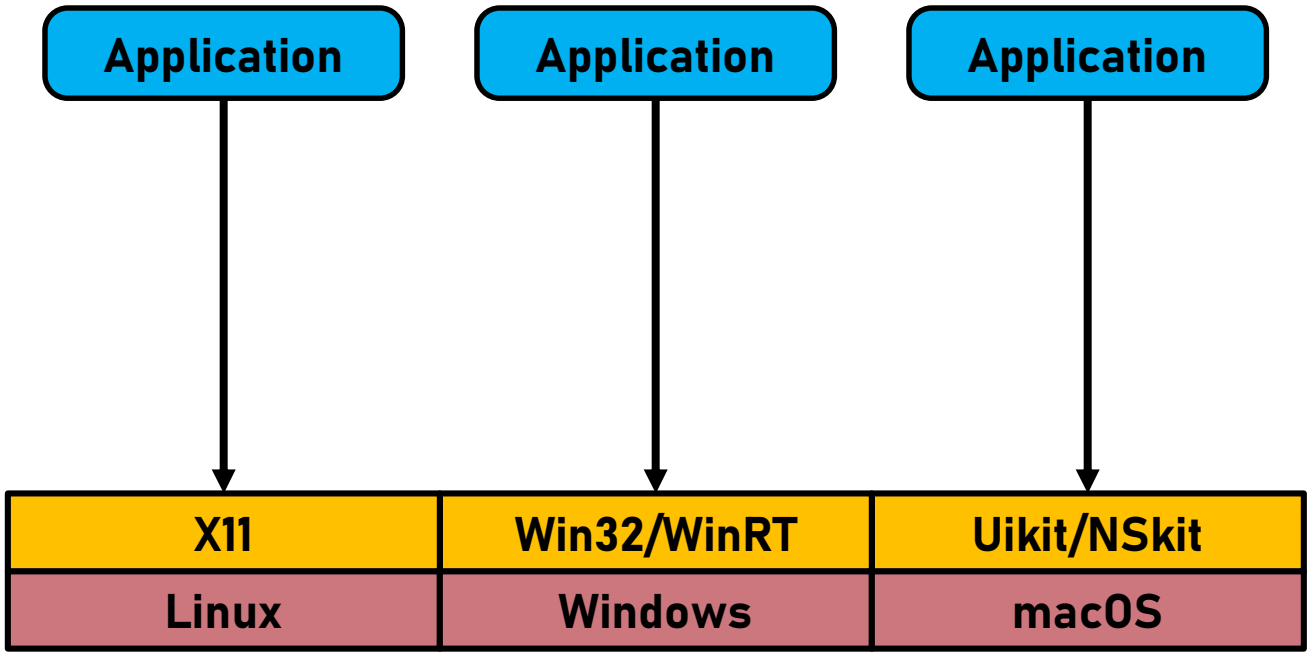
LUND UNIVERSITY

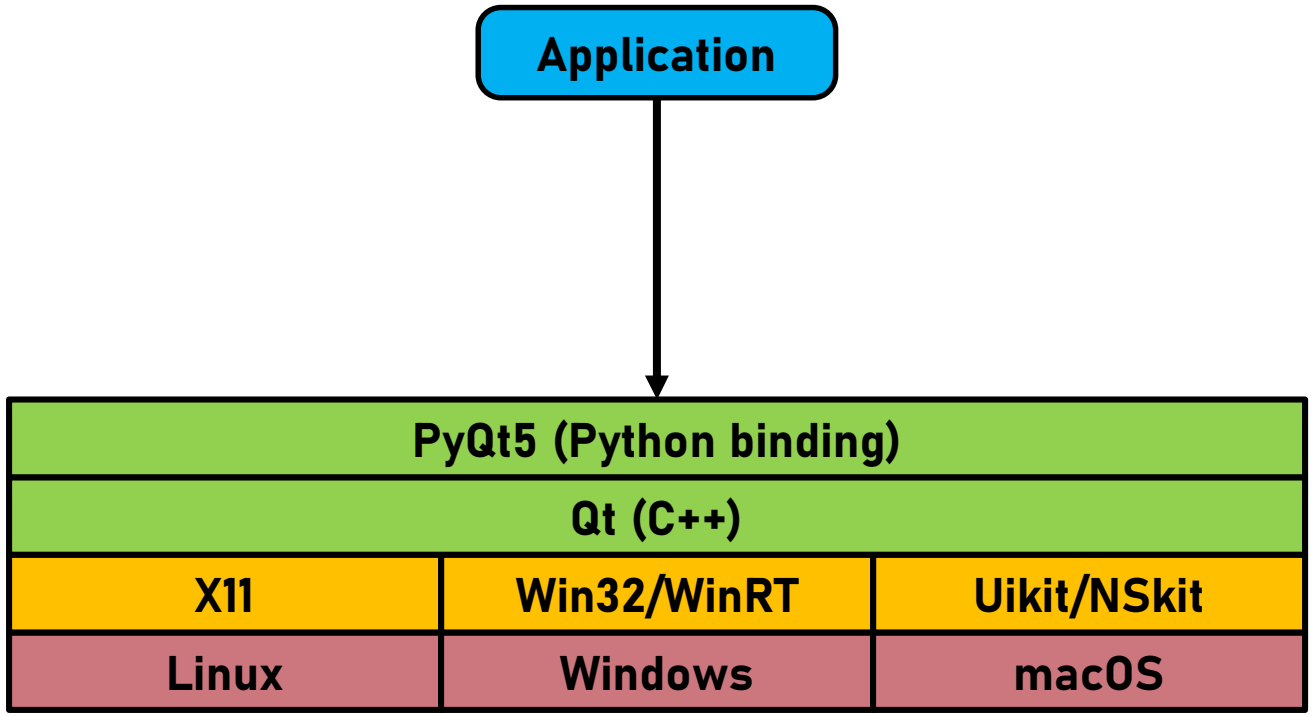
PyQt5

GRAPHICAL USER INTERFACES FOR PYTHON

User interfaces on different platforms



User interfaces on different platforms



Qt

- Platform independent user interface library
- Abstracts all user interface concepts in single library
  - No need to call native functions for any platform
- User interfaces adapts to current platform
  - Applications behave like native applications
- User interfaces can be styled for custom looks
- Events handled using a signals and slots mechanism

LUND UNIVERSITY

Qt and Python

- Qt is a C++ library
- Using Qt from Python requires a binding
  - Binary modules connecting Python with Qt
- Several bindings for Qt exists:
  - PyQt5/PyQt4 (GPLv3)
  - Qt for Python (PySide)
  - PythonQt
- PyQt5 is used in this course

LUND UNIVERSITY

A solution to the many Qt-libraries - **qtpy**

- Tries to figure out which Python-bindings are installed
- Create a neutral namespace for Qt
- Example replace
  - PyQt5.QtWidgets
  - Qtpy.QtWidgets

LUND UNIVERSITY

Event-based programming

- Centered around an event loop
- The event loop waits for messages from the operating system
  - Keyboard, mouse and system messages
- Dispatches the message to code that handles the event.
- Loop only exists when last window has been closed.

LUND UNIVERSITY

Event loop pseudo code

```
running = True

while running:
    event = check_for_event()

    if event.type == BUTTON_CLICK:
        handle_button_click(event)

    # ... More checks here ...

    if event.type == APP_QUIT
        running  = False
```

Qt main program

```
import sys

from qtpy.QtWidgets import *

if __name__ == "__main__":

    # Create application object

    app = QApplication(sys.argv)

    # Create user interface objects

    widget = QWidget()
    widget.show()

    # Enter application loop

    sys.exit(app.exec_())
```

ex1.py

A Qt application

- QWidget is the base class for all user interface objects in Qt
- Can contain other user interface objects
- Will create a window if it has no parent
- Derive a custom application window from Qwidget

LUND UNIVERSITY

A Main window class

```
from qtpy.QtWidgets import *

class MyWindow(QWidget):
     def __init__(self):
          """MyWindow constructor"""

          super().__init__()

          # Create user interface constrols

          self.init_gui()

     def init_gui(self):
          """Initialise user interface"""

          self.setGeometry(300, 300, 600, 600)
          self.setWindowTitle("MyWindow")

          # Show window

          self.show()
```

ex2.py

Main program

```
if __name__ == "__main__":

    app = QApplication(sys.argv)

    # Create our MyWindow object

    window = MyWindow()

    # Enter event loop.

    sys.exit(app.exec_())
```

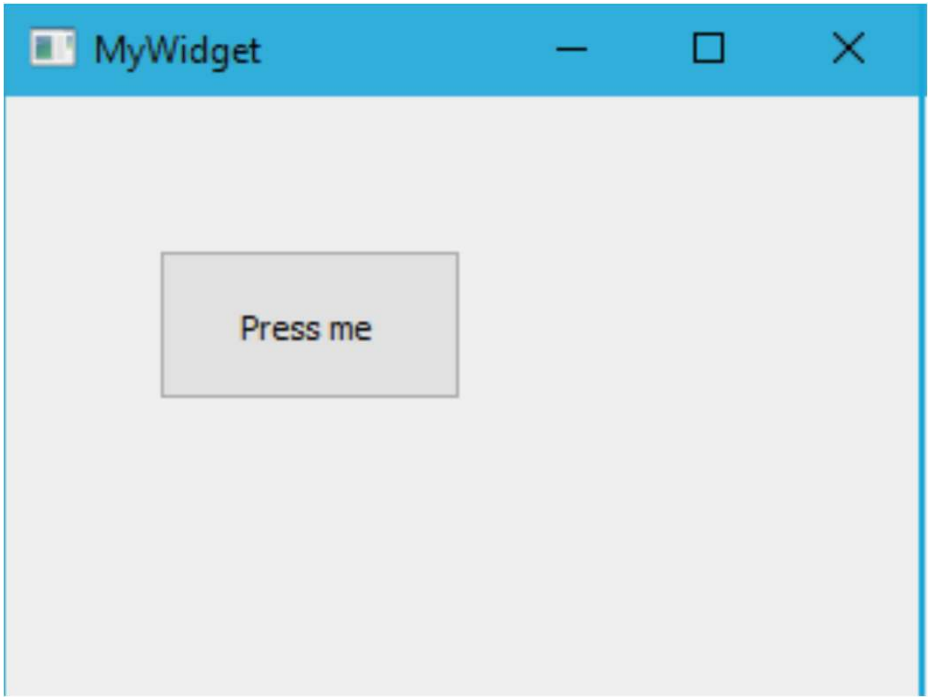
ex2.py

User interface controls

ex3.py

```
from qtpy.QtWidgets import *

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

     def init_gui(self):
          """Initialise user interface"""

          self.setGeometry(300, 300, 600, 600)
          self.setWindowTitle("MyWindow")

          # User interface controls

          self.button = QPushButton("Press me", self)
          self.button.setToolTip("I am a button. Please press me")
          self.button.resize(100,50)
          self.button.move(50,50)

          # Show window

          self.show()
```

User interface controls



Connecting events/signals

- All Qt-controls have a number of events/signals that can be connected to class methods
- Available events/signals are available as properties of the class
  - my_button.clicked (button click)
  - my_button.pressed (button is held down)
  - my_listbox.currentRowChanged (row in list box has changed)
- Connection to code is done using the connect method of the signal
  - my_button.clicked.connect(self.on_my_button_clicked)

LUND UNIVERSITY

Connecting events/signals

- Information on which signals are available to different controls can be found the in the Qt reference library
  - https://doc.qt.io/qt-5/qtwidgets-module.html
- General information on Qt and its library can be found here:
  - https://doc.qt.io/qt-5/qtwidgets-index.html#
- These references are for C++, but the Python methods are very similar.

`ex4.py`

LUND UNIVERSITY

Common control properties

- QWidget base class implements many shared control properties
  - If control is visible (.setVisible()/.isVisible())
  - If control is active (.setEnabled()/.isEnabled())
  - If control has focus (.setFocus())
  - Fonts
  - Standard text (.setText()/.text())

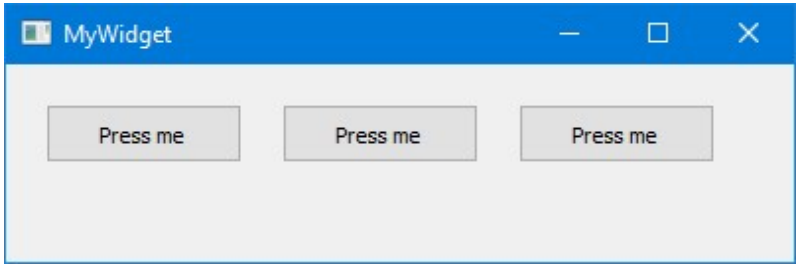
LUND UNIVERSITY

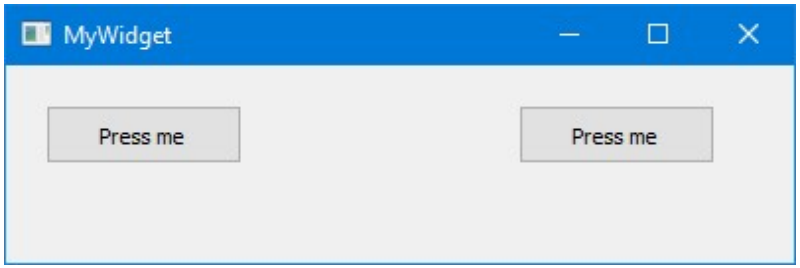
Common properties

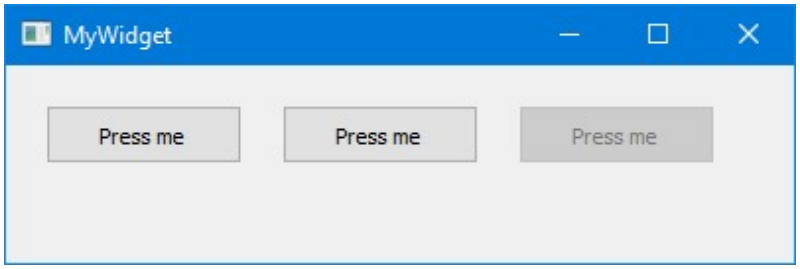
control1.py

```
class MyWindow(QWidget):

        ...

        def init_gui(self):
                """Initiera gränssnitt"""

                self.button1 = QPushButton("Press me", self)
                self.button1.resize(100,30)
                self.button1.move(20,20)

                self.button2 = QPushButton("Press me", self)
                self.button2.resize(100,30)
                self.button2.move(20+120,20)

                self.button3 = QPushButton("Press me", self)
                self.button3.resize(100,30)
                self.button3.move(20+120*2,20)

                self.button1.clicked.connect(self.on_button1_clicked)
                self.button2.clicked.connect(self.on_button2_clicked)

        def on_button1_clicked(self):
                """Händelsemetod för signalen clicked för button1"""
                if self.button2.isVisible():
                        self.button2.setVisible(False)
                else:
                        self.button2.setVisible(True)


        def on_button2_clicked(self):
                """Händelsemetod för signalen clicked för button2"""
                if self.button3.isEnabled():
                        self.button3.setEnabled(False)
                else:
                        self.button3.setEnabled(True)
```

Common properties

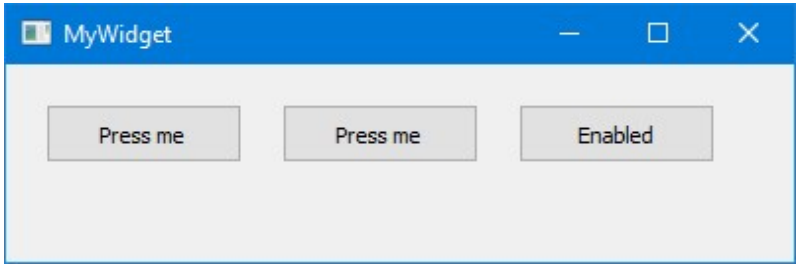


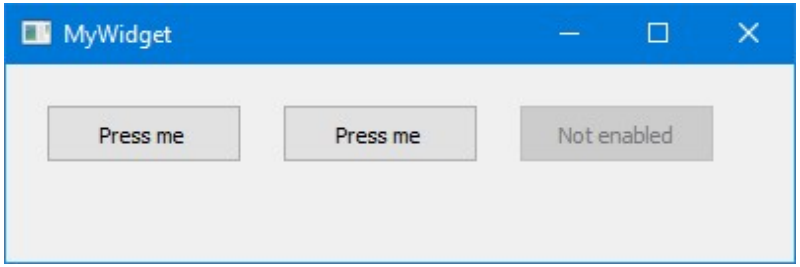




Common properties

```
def on_button2_clicked(self):
    """Händelsemetod för signalen clicked"""
    if self.button3.isEnabled():
        self.button3.setEnabled(False)
        self.button3.setText("Not enabled")
    else:
        self.button3.setEnabled(True)
        self.button3.setText("Enabled")
```





Connecting events

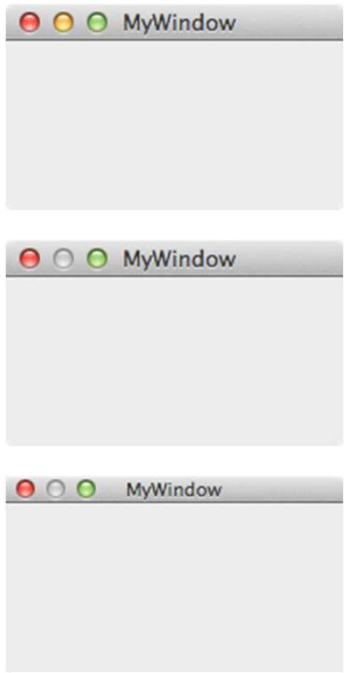
ex4.py

```
from qtpy.QtWidgets import *

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

     def init_gui(self):
          """Initialise user interface"""

          ...

          self.button = QPushButton("Press me", self)
          self.button.setToolTip("I am a button. Please press me")
          self.button.resize(100,50)
          self.button.move(50,50)

          # Connect events to method

          self.button.clicked.connect(self.on_button_clicked)

          ...


     def on_button_clicked(self):
          """Event method for self.button"""
          print("Hello")
```

Window styles

```
import sys

from qtpy.QtWidgets import *
from qtpy.QtCore import *

class MyWindow(QMainWindow):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__(None, Qt.Window)
        #super().__init__(None, Qt.Window | Qt.Dialog)
        #super().__init__(None, Qt.Window | Qt.Tool)
        self.resize(200,100)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

if __name__ == '__main__':

    app = QApplication(sys.argv)

    window = MyWindow()
    window.show()

    sys.exit(app.exec_())
```

Different window styles



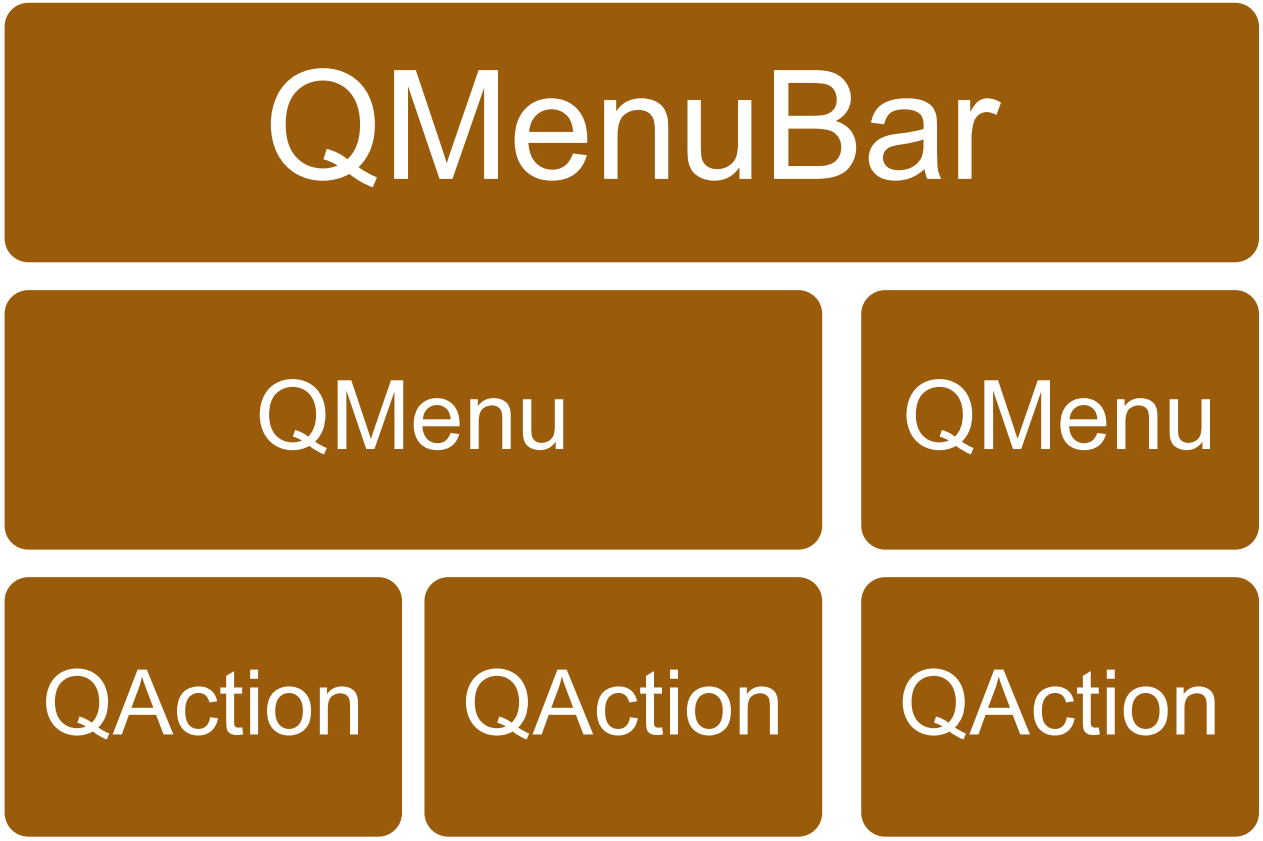
`window_style1.py`

Maximised windows

```
import sys

from qtpy.QtWidgets import *
from qtpy.QtCore import *

class MyWindow(QMainWindow):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.resize(200,100)
        self.move(50,50)
        self.setWindowTitle("MyWindow")
        self.setWindowState(Qt.WindowMaximized)
        #self.setWindowState(Qt.WindowFullScreen)

if __name__ == '__main__':

    app = QApplication(sys.argv)

    window = MyWindow()
    window.show()

    sys.exit(app.exec_())
```

window_maximised.py

Menues and toolbars



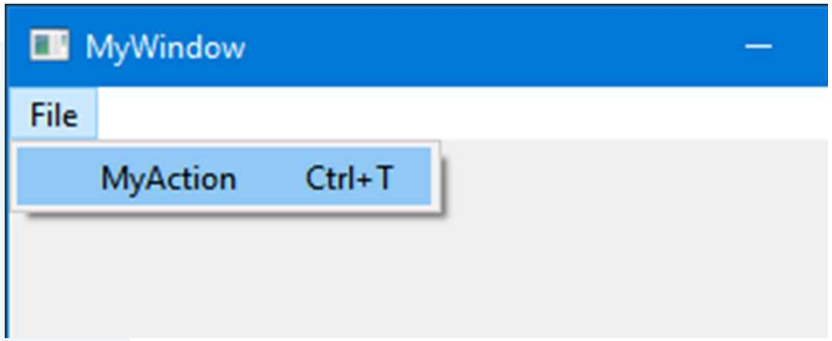
Menues

- Menus are connected to QAction-objects
- QActions is a generic connection in the user interface
  - Shared with menus and toolbars
  - Icons and shortcuts handles in one location
- Events connected directly to actions instead of controls.

LUND UNIVERSITY

Menyer

```python
import sys

from qtpy.QtWidgets import *

class MyWindow(QMainWindow):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.resize(200,200)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        # Define action

        self.my_action = QAction("MyAction", self)
        self.my_action.setShortcut("Ctrl+T")
        self.my_action.triggered.connect(self.on_my_action)

        # Connect action to menu

        self.fileMenu = self.menuBar().addMenu("File")
        self.fileMenu.addAction(self.my_action)

    def on_my_action(self):
        """Method for handling MyAction"""
        QMessageBox.information(self, "Meddelande", "Ouch!")
```



Create action

Create menu

Connect action to menu

menu1.py

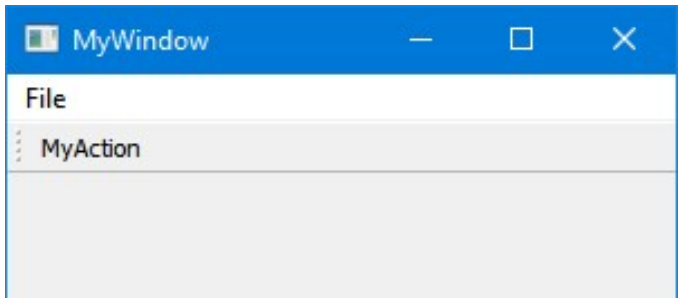
Toolbars

toolbar1.py
toolbar2.py

```
class MyWindow(QMainWindow):

    ...

    def init_gui(self):

        ...

        # Create a toolbar

        self.toolbar = self.addToolBar("MyToolbar")
        self.toolbar.addAction(self.my_action)
```

Add toolbar

Connect action to toolbar



Layout management

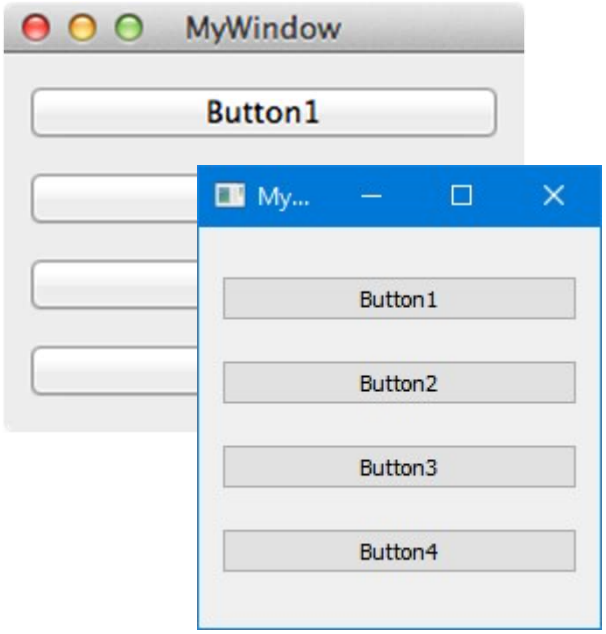
- Two options
  - Absolute positioning0
  - ”Sizers”
- Absolute positioning can be problematic if window resizes or other changes.
- Sizers automatically places controls according to rules.

LUND UNIVERSITY

QVBoxLayout

vboxsizer.py

```
import sys

from qtpy.QtWidgets import *

class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.resize(200,200)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        self.button1 = QPushButton("Button1")
        self.button2 = QPushButton("Button2")
        self.button3 = QPushButton("Button3")
        self.button4 = QPushButton("Button4")

        self.vbox = QVBoxLayout(self)
        self.vbox.addWidget(self.button1)
        self.vbox.addWidget(self.button2)
        self.vbox.addWidget(self.button3)
        self.vbox.addWidget(self.button4)

        self.setLayout(self.vbox)
```



Create layout

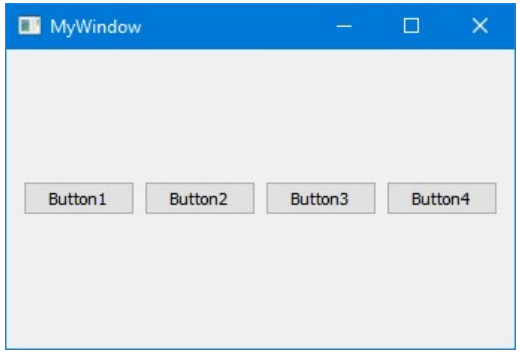
Add controls to layout

Set window layout

QHBoxLayout

hboxsizer.py

```
import sys

from qtpy.QtWidgets import *

class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.init_gui()

    def init_gui(self):
        """Inititera gränssnitt"""

        self.resize(200,200)
        self.move(50,50)
        self.setWindowTitle('MyWindow')

        self.button1 = QPushButton('Button1', self)
        self.button2 = QPushButton('Button2', self)
        self.button3 = QPushButton('Button3', self)
        self.button4 = QPushButton('Button4', self)

        self.hbox = QHBoxLayout(self)
        self.hbox.addWidget(self.button1)
        self.hbox.addWidget(self.button2)
        self.hbox.addWidget(self.button3)
        self.hbox.addWidget(self.button4)

        self.setLayout(self.hbox)
```

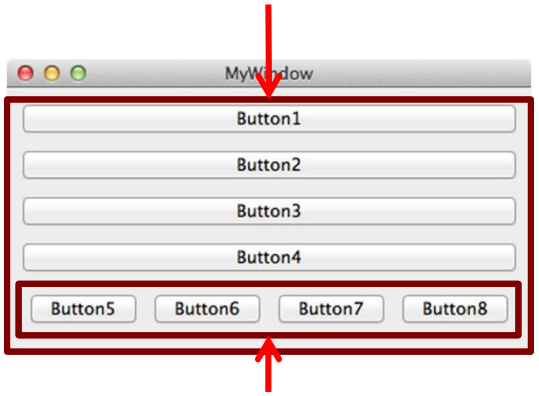


Mix and match

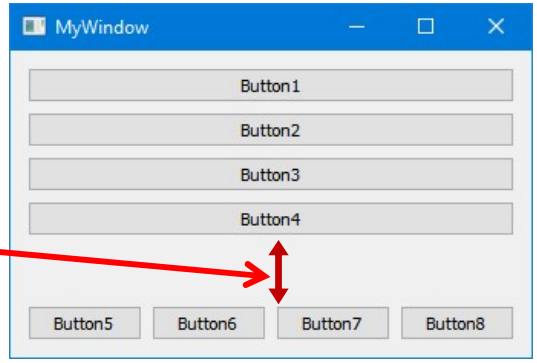
hvboxsizer.py

```python
import sys

from qtpy.QtWidgets import *

class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.init_gui()

    def init_gui(self):

        self.resize(200,200)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        self.button1 = QPushButton('Button1')
        self.button2 = QPushButton('Button2')
        self.button3 = QPushButton('Button3')
        self.button4 = QPushButton('Button4')

        self.button5 = QPushButton('Button5')
        self.button6 = QPushButton('Button6')
        self.button7 = QPushButton('Button7')
        self.button8 = QPushButton('Button8')

        self.vbox = QVBoxLayout(self)
        self.vbox.addWidget(self.button1)
        self.vbox.addWidget(self.button2)
        self.vbox.addWidget(self.button3)
        self.vbox.addWidget(self.button4)

        self.hbox = QHBoxLayout(self)
        self.hbox.addWidget(self.button5)
        self.hbox.addWidget(self.button6)
        self.hbox.addWidget(self.button7)
        self.hbox.addWidget(self.button8)

        self.vbox.addStretch(1)
        self.vbox.addLayout(self.hbox)

        self.setLayout(self.vbox)
```

self.vbox



self.hbox



Add an expanding "spring"

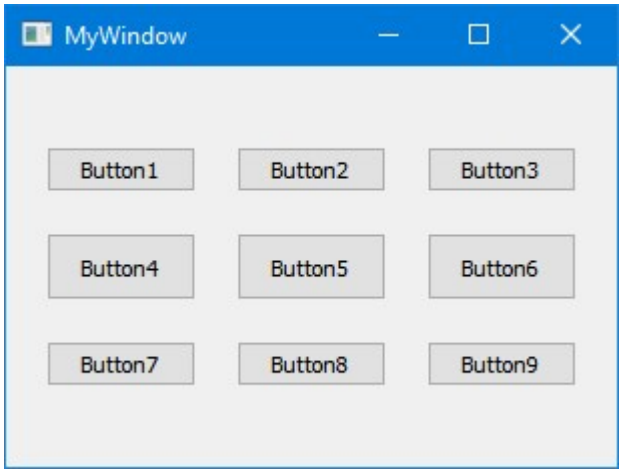
Add layout to self.vbox

Set window layout to self.vbox

QGridLayout

gridlayout1.py

```python
import sys

from qtpy.QtWidgets import *

class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """MyWidget constructor"""
        super().__init__()

        self.init_gui()

    def init_gui(self):
        """Initiera gränssnitt"""

        self.resize(200,200)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        self.button1 = QPushButton("Button1")
        self.button2 = QPushButton("Button2")
        self.button3 = QPushButton("Button3")
        self.button4 = QPushButton("Button4")
        self.button5 = QPushButton("Button5")
        self.button6 = QPushButton("Button6")
        self.button7 = QPushButton("Button7")
        self.button8 = QPushButton("Button8")
        self.button9 = QPushButton("Button9")

        self.button1.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button2.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button3.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button4.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button5.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button6.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button7.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button8.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        self.button9.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.grid = QGridLayout(self)
        self.grid.addWidget(self.button1, 0, 0)
        self.grid.addWidget(self.button2, 0, 1)
        self.grid.addWidget(self.button3, 0, 2)
        self.grid.addWidget(self.button4, 1, 0)
        self.grid.addWidget(self.button5, 1, 1)
        self.grid.addWidget(self.button6, 1, 2)
        self.grid.addWidget(self.button7, 2, 0)
        self.grid.addWidget(self.button8, 2, 1)
        self.grid.addWidget(self.button9, 2, 2)

        self.grid.setContentsMargins(20, 40, 20, 40)

        self.grid.setHorizontalSpacing(20)
        self.grid.setVerticalSpacing(20)

        self.grid.setColumnStretch(0, 1)
        self.grid.setColumnStretch(1, 4)
        self.grid.setColumnStretch(2, 1)

        self.grid.setRowStretch(0, 1)
        self.grid.setRowStretch(1, 4)
        self.grid.setRowStretch(2, 1)

        self.setLayout(self.grid)
```



Controls button size behavior

Controls are place by assigning row and column

Springs/stretch can be added to specify desired scaling behavior

Standard dialogs

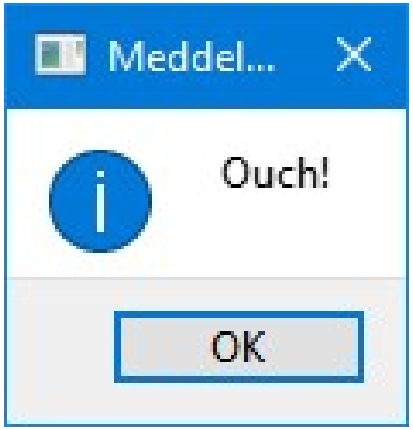
- Provide the same dialogs that are available on each platform
  - Message dialogs
  - File dialogs
  - Color dialogs

LUND UNIVERSITY

Dialogrutor

dialogs1.py

```
def on_dialog(self):
        """Method for handling MyAction"""
        QMessageBox.information(self, "Meddelande", "Ouch!")
```



Dialogrutor

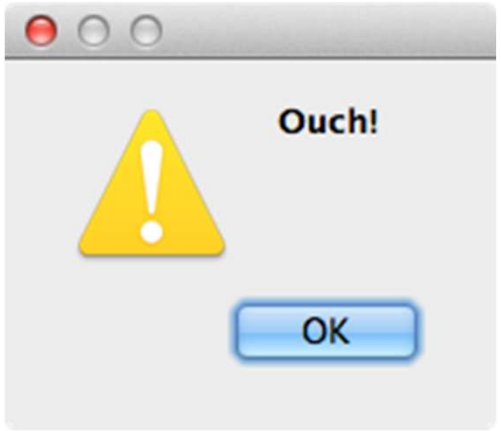
dialogs2.py

```
def on_dialog(self):
        """Method for handling MyAction"""
        QMessageBox.critical(self, "Meddelande", "Ouch!")
```

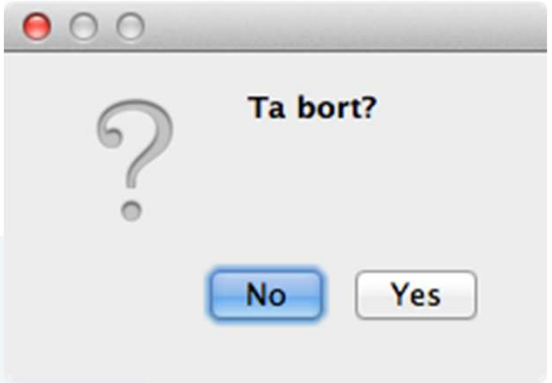
Dialogrutor

dialogs3.py

```
def on_dialog(self):
        """Method for handling MyAction"""
        QMessageBox.warning(self, "Meddelande", "Ouch!")
```



Dialogrutor




```
def on_dialog(self):
        """Method for handling MyAction"""
        result = QMessageBox.question(
            self, "Meddelande", "Ta bort?",
            QMessageBox.Yes | QMessageBox.No,
            QMessageBox.No
        )

        if result == QMessageBox.Yes:
            QMessageBox.information(self, "Val", "Du valde Yes")
        else:
            QMessageBox.information(self, "Val", "Du valde No")
```

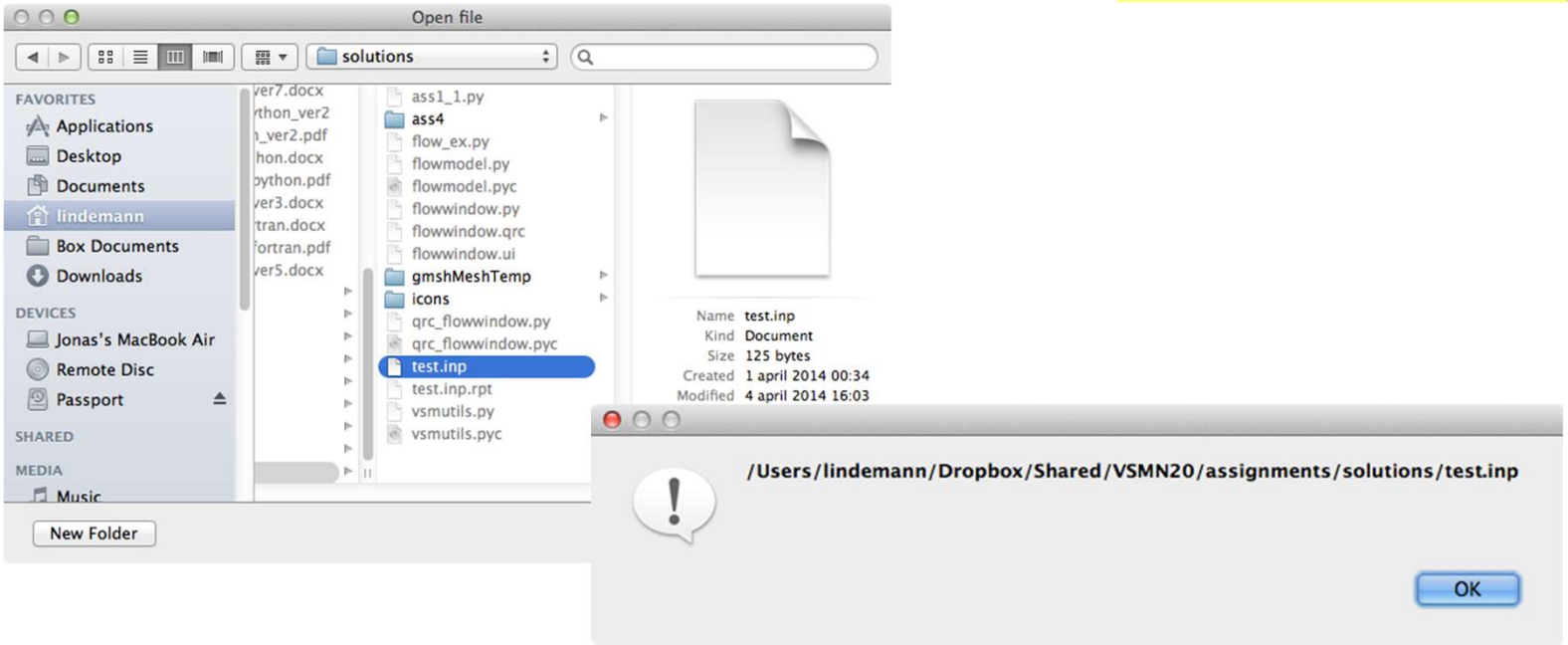
dialogs4.py

Dialogrutor

```
def on_dialog(self):
        """Method for handling MyAction"""
        filename, _ = QFileDialog.getOpenFileName(
            self, 'Open file', '', 'Flow input files (*.inp)')

        if filename != "":
            QMessageBox.information(self, "Val", filename)
```

Filters which extensions are visible

dialogs5.py

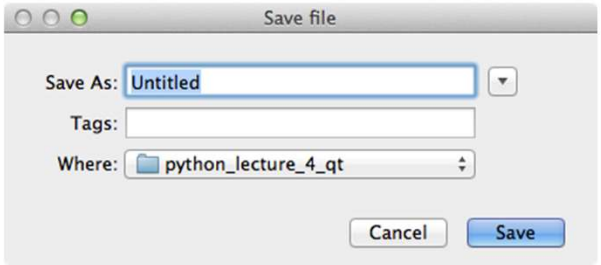


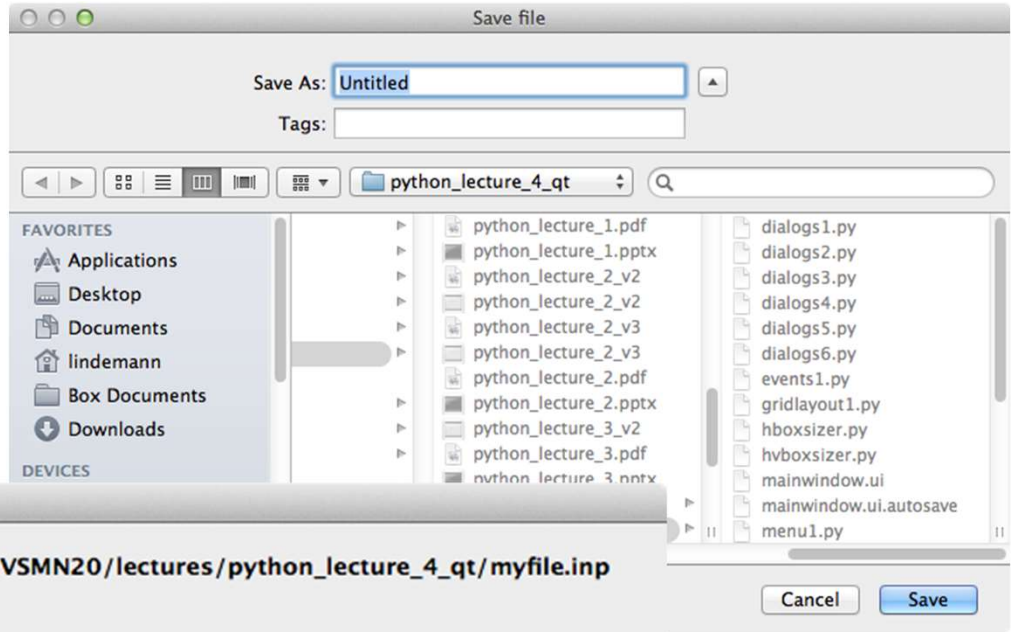
Dialogrutor

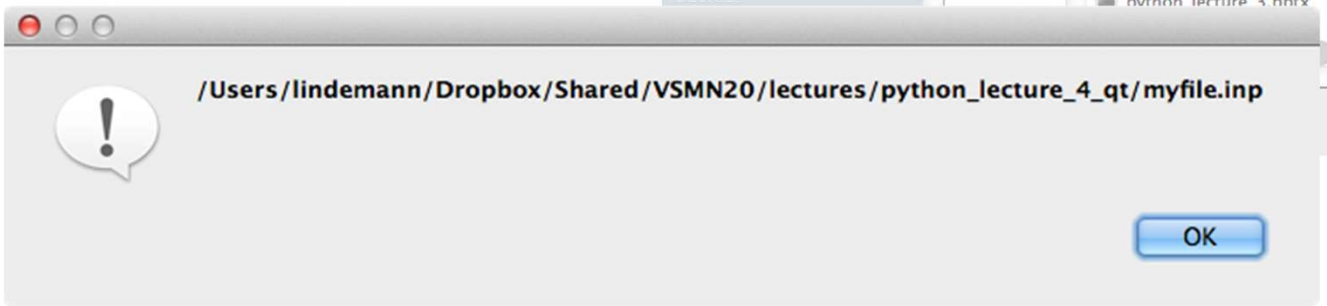
dialogs6.py

```
def on_dialog(self):
        """Method for handling MyAction"""
        filename, _ = QFileDialog.getSaveFileName(
            self, 'Save file', '', 'Flow input files (*.inp)')

        if filename != "":
            QMessageBox.information(self, "Val", filename)
```

Standard extension for new files







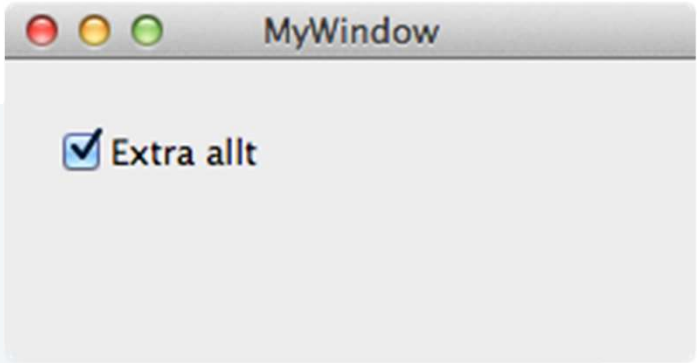
Controls

QCheckBox / QRadioButton

- Buttons with state
  - Compare on/off buttons
  - Enabled/Disabled
- QCheckBox
  - setCheckState() / checkState() – Sets and returns current state
- QRadioButton
  - setChecked() / isChecked() – Sets and returns current state

LUND UNIVERSITY

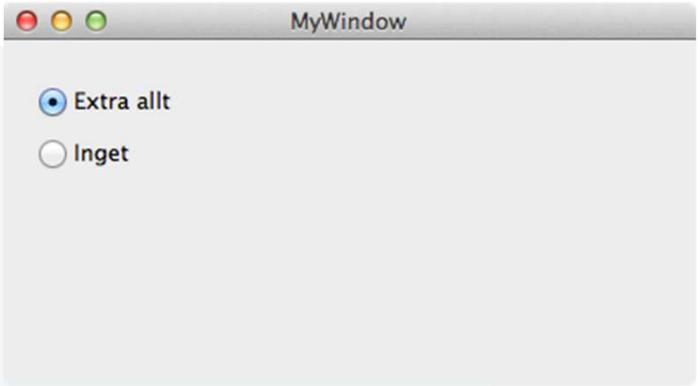
QCheckBox




```python
class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.resize(400,100)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        self.check_box = QCheckBox("Extra allt", self)
        self.check_box.move(20,20)
        self.check_box.setChecked(True)
        self.check_box.stateChanged.connect(self.on_state_change)


    def on_state_change(self):
        """Respond to button click"""
        if self.check_box.checkState():
            QMessageBox.information(self, "Meddelande", "Extra allt")
        else:
            QMessageBox.information(self, "Meddelande", "Inget")
```

checkbox.py

QRadioButton




```
class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        self.resize(400,100)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        self.radio_button1 = QRadioButton("Extra allt", self)
        self.radio_button1.move(20,20)
        self.radio_button2 = QRadioButton("Mer...", self)
        self.radio_button2.move(20,50)
        self.radio_button1.clicked.connect(self.on_radio_button_clicked)
        self.radio_button2.clicked.connect(self.on_radio_button_clicked)

        self.radio_button1.setChecked(True)

    def on_radio_button_clicked(self):
        """Svara på radiorute val"""
        if self.radio_button1.isChecked():
            QMessageBox.information(self, "Meddelande", "Radio 1 markerad")
        else:
            QMessageBox.information(self, "Meddelande", "Radio 2 markerad")
```

radiobox.py

QComboBox

- Multiple selection control / Text box
  - Select from a limited set of options
  - Text box with predefined options
- Compact control – takes up little room
- Selection in the list can be connected to currentItemChanged
  - currentIndex() return currently selected item

LUND UNIVERSITY

QComboBox

```python
class MyWindow(QWidget):
    """Huvudklass för vårt fönster"""

    def __init__(self):
        """Klass constructor"""
        super().__init__()

        # Konfigurera fönster

        self.resize(400, 200)
        self.move(50, 50)
        self.setWindowTitle("MyWindow")

        # Skapa combobox-kontroll

        self.combo_box = QComboBox(self)
        self.combo_box.move(20, 20)

        # Lägg till alternativ

        self.combo_box.addItem("Alternativ 1")
        self.combo_box.addItem("Alternativ 2")
        self.combo_box.addItem("Alternativ 3")
        self.combo_box.addItem("Alternativ 4")

        # Ange standardval

        self.combo_box.setCurrentIndex(3)

        # Koppla händelsemetod till signal

        self.combo_box.currentIndexChanged.connect(self.on_current_index_changed)

    def on_current_index_changed(self, index):
        """Hantera signalen currentIndexChanged"""

        QMessageBox.information(self, "Meddelande", "Du valde: " + str(index))
        QMessageBox.information(self, "Meddelande", "Texten var: " + self.combo_box.currentText())
```

combobox.py

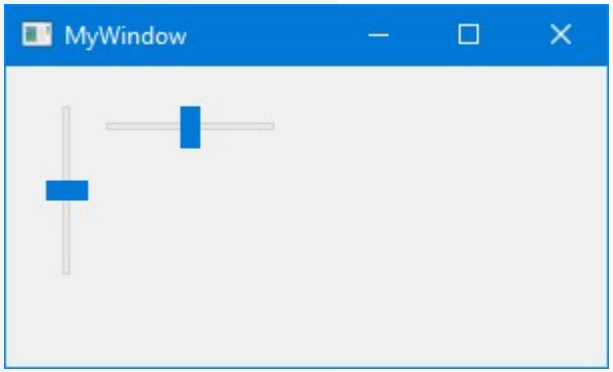
QSlider

- Slider for enabling fast input of numerical values
- Options availble for range and fixed positions on a scale

LUND UNIVERSITY

QSlider

```python
class MyWindow(QWidget):
    """Huvudklass för vårt fönster"""

    def __init__(self):
        """Klass konstructor"""
        super().__init__()

        # Konfigurera fönster

        self.resize(300, 150)
        self.move(50, 50)
        self.setWindowTitle("MyWindow")

        # Skapa kontroller

        self.vert_slider = QSlider(Qt.Vertical, self)
        self.vert_slider.move(20, 20)
        self.vert_slider.setMaximum(100)
        self.vert_slider.setMinimum(0)
        self.vert_slider.setValue(50)

        self.horiz_slider = QSlider(Qt.Horizontal, self)
        self.horiz_slider.move(50, 20)
        self.horiz_slider.setMaximum(100)
        self.horiz_slider.setMinimum(0)
        self.horiz_slider.setValue(50)

        # Koppla signaler

        self.vert_slider.valueChanged.connect(self.on_value_changed)
        self.horiz_slider.valueChanged.connect(self.on_value_changed)

    def on_value_changed(self, value):
        """Hantera signalen valueChanged"""
        print("vertical value   =", self.vert_slider.value())
        print("horizontal value =", self.horiz_slider.value())
```



slider.py

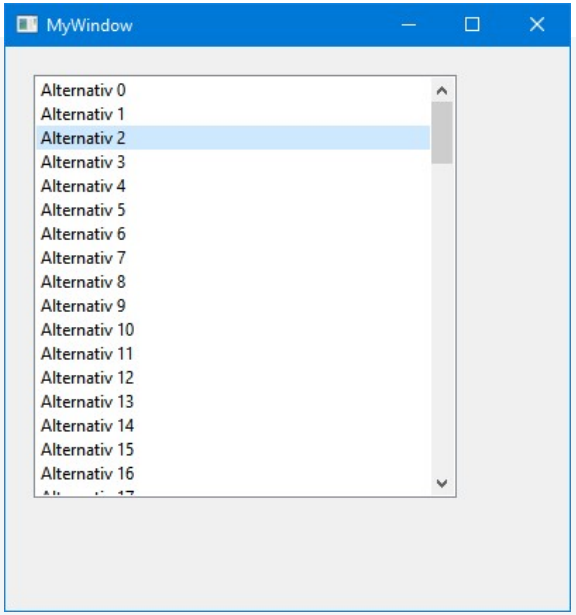
QListBox

- Shows a box with a number of selectable options
- Used when selecting from a large number of options.
  - Scrollbars are added when needed
- Supports multiple selection

LUND UNIVERSITY

QListWidget

```python
class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class konstruktor"""
        super().__init__()

        # Sätt fönsteregenskaper

        self.resize(400,200)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        # Skapa listkontroll

        self.list_box = QListWidget(self)
        self.list_box.move(20,20)
        self.list_box.resize(300,300)

        # Lägg till alternativ i listan

        for i in range(100):
            self.list_box.addItem("Alternativ %d" % i)

        # Sätt standardalternativet till rad 2

        self.list_box.setCurrentRow(2)

        # Koppla en händelsemetod till signal

        self.list_box.currentRowChanged.connect(self.on_current_row_changed)

    def on_current_row_changed(self, curr):
        """Hantera signalen currentRowChanged"""
        QMessageBox.information(self, "Meddelande", "Du valde: " + str(curr))
        QMessageBox.information(self, "Meddelande", "Raden innehöll: " + self.list_box.currentItem().text())
```



listbox.py

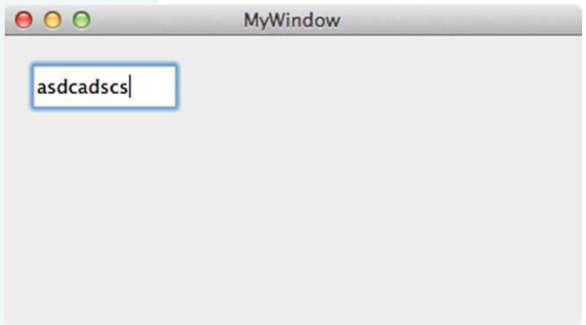
QLineEdit

- Shows a text box where a user can enter values
- Values are set with .setText() and retrieved using .text()

LUND UNIVERSITY

QLineEdit

```python
class MyWindow(QWidget):
    """Main Window class for our application"""

    def __init__(self):
        """Class constructor"""
        super().__init__()

        # Konfigurera fönster

        self.resize(400,200)
        self.move(50,50)
        self.setWindowTitle("MyWindow")

        # Skapa knapp

        self.button = QPushButton("Tryck", self)
        self.button.move(50,50)
        self.button.resize(100,50)
        self.button.clicked.connect(self.on_button_clicked)

        # Skapa textkontroll

        self.lineEdit = QLineEdit(self)
        self.lineEdit.move(20,20)
        self.lineEdit.setText("Text")

    def on_button_clicked(self):
        """Händelsemetod för signalen clicked"""
        QMessageBox.information(self, "Text", self.lineEdit.text())
```



lineedit.py

Connecting model and user interface

- Important not to mix user interface code with simulation models.
- Handle update of user interface and model in a consistent way.
- Create separate functions in the user interface code
  - updateControls(...)
  - updateModel(…)
- Functions can be reused
  - Reading model from disk
  - Before calculation
  - Creating new model.

LUND UNIVERSITY

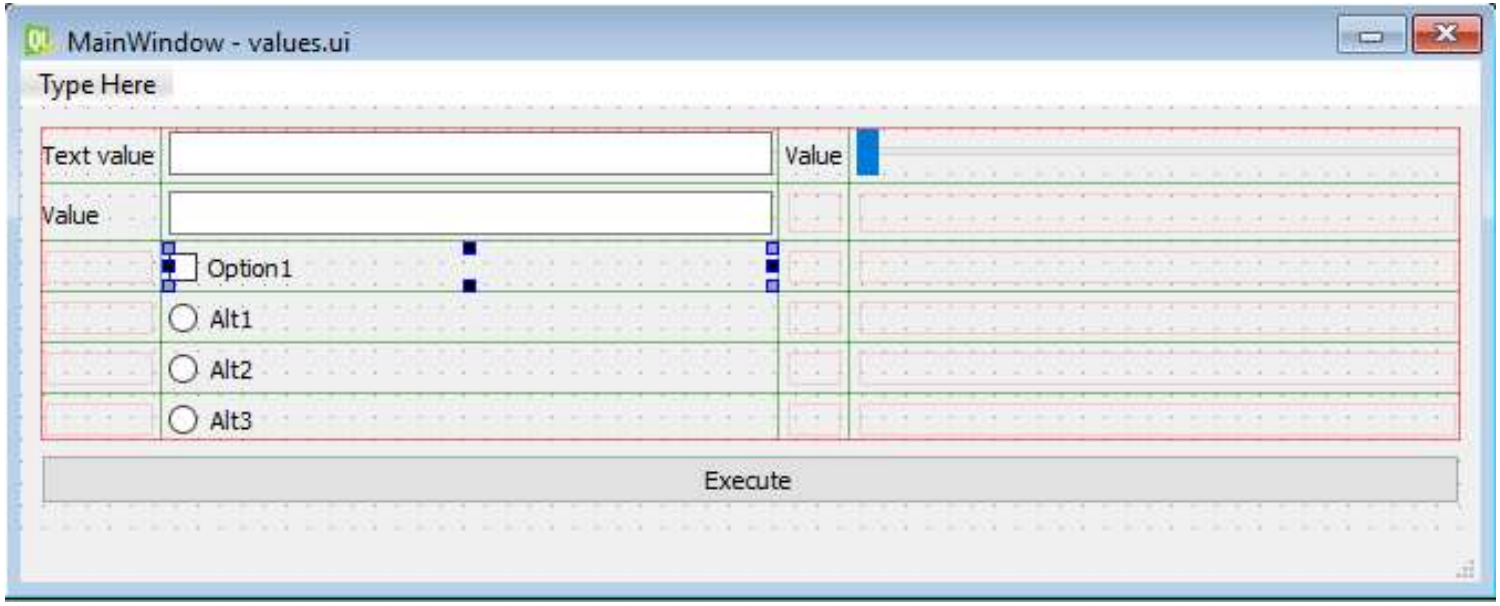
Example - Model

```
class Model(object):
    def __init__(self):
        self.textValue = "Noname"
        self.value1 = 42.0
        self.option1 = True
        self.alternative = 1
        self.value2 = 84

    def __str__(self):
        s = StringIO()
        sys.stdout = s

        print("Text value =", self.textValue)
        print("Value1 =", self.value1)
        print("Option =", self.option1)
        print("Alternative =", self.alternative)
        print("Value2 = ", self.value2)

        sys.stdout = sys.__stdout__

        return s.getvalue()
```

Model values

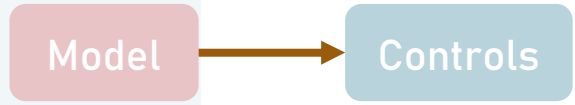
Printing support

Example – User interface



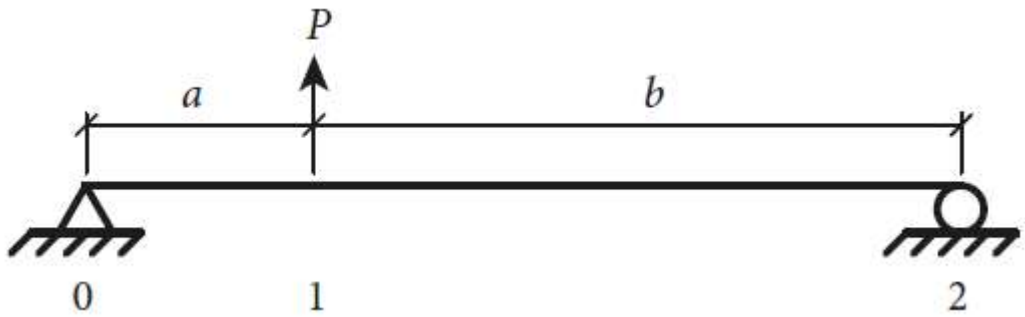
Exempel – User interface

```python
def updateControls(self):
        """Update controls from model values"""
        self.ui.textEdit.setText(self.model.textValue)
        self.ui.valueEdit.setText(str(self.model.value1))
        self.ui.optionCheck.setChecked(self.model.option1)
        self.ui.valueSlider.setValue(self.model.value2)

        if self.model.alternative == 1:
            self.ui.alt1Radio.setChecked(True)
        elif self.model.alternative == 2:
            self.ui.alt2Radio.setChecked(True)
        elif self.model.alternative == 3:
            self.ui.alt3Radio.setChecked(True)

    def updateModel(self):
        """Update model from controls"""
        self.model.textValue = self.ui.textEdit.text()
        self.model.value1 = float(self.ui.valueEdit.text())
        self.model.option1 = self.ui.optionCheck.isChecked()
        self.model.value2 = self.ui.valueSlider.value()

        if self.ui.alt1Radio.isChecked():
            self.model.alternative = 1
        elif self.ui.alt2Radio.isChecked():
            self.model.alternative = 2
        elif self.ui.alt3Radio.isChecked():
            self.model.alternative = 3
```



Simply supported beam

Beam theory



$$V_{0-1}(x) = \frac{Pb}{L} \qquad V_{1-2}(x) = -\frac{Pa}{L}$$

$$_1(x) = -\frac{Pbx}{L} \qquad M_{1-2}(x) = -\frac{Pa(L-x)}{L}$$

$$v_{1-2}(x) = \frac{Pa}{6EI}(-a^2 + (2L + \frac{a^2}{L})x - 3x^2 + \frac{x^3}{L})$$

$$v_{0-1}(x) = \frac{PbL}{6EI}((1 - \frac{b^2}{L^2})x - \frac{x^3}{L^2})$$

Example – Beam model usage

```
beam = BeamSimplySupported()

beam.a = 1.0                # Distance from P to left
beam.b = 2.0                # Distance from right to P
beam.P = 1000               # Force P
beam.E = 2.1e9              # Elastic modulus
beam.I = 0.1*0.1**4/12.0    # Moment of inertia

# Initiate loop variables

x = 0.0
dx = 0.1

# Print table headers

print('{:>10}  {:>10}  {:>10}  {:>10}'.format("x (m)", "v (m)", "V (N)", "M (Nm)"))

# Loop over x and print beam section values

while x < beam.L + dx:
    print('{:10.5}, {:10.5}, {:10.5}, {:10.5}'.format(x, beam.v(x), beam.V(x), beam.M(x)))
    x += dx
```

Example – Beam model class

Class constructor:

```
class BeamSimplySupported:
    """Class for computing deflection and section forces
    for a simply supported beam"""

    def __init__(self):
        """BeamSimplySupported constructor"""

        # Initiera standardvärden

        self.__a = 1.0
        self.__b = 2.0
        self.__L = self.__a + self.__b
        self.__P = 1000
        self.__E = 2.1e9
        self.__I = 0.1*0.1**4/12.0
```

Example – Beam model class

Safe float conversion method.

```
class BeamSimplySupported:

    ...

    def to_float(self, new_value, old_value):
        """Assigning properties safely"""

        try:
            v = float(new_value)
        except ValueError:
            return old_value

        return v
```

Example – Beam model class

Get/set methods with safe conversion

```
class BeamSimplySupported:

    ...

    # --- Get/Set methods for properties

    def get_a(self):
        return self.__a

    def set_a(self, v):
        self.__a = self.to_float(v, self.__a)

    def get_b(self):
        return self.__b

    def set_b(self, v):
        self.__b = self.to_float(v, self.__b)

    def get_P(self):
        return self.__P

    def set_P(self, v):
        self.__P = self.to_float(v, self.__P)

    def get_L(self):
        return self.__a + self.__b

    def get_E(self):
        return self.__E

    def set_E(self, v):
        self.__E = self.to_float(v, self.__E)

    def get_I(self):
        return self.__I

    def set_I(self, v):
        self.__I = self.to_float(v, self.__I)
```

Example – Beam model class

Defining class properties

```
class BeamSimplySupported:

    ...

    a = property(get_a, set_a)
    b = property(get_b, set_b)
    L = property(get_L)  # <-- Read only
    P = property(get_P, set_P)
    E = property(get_E, set_E)
    I = property(get_I, set_I)
```

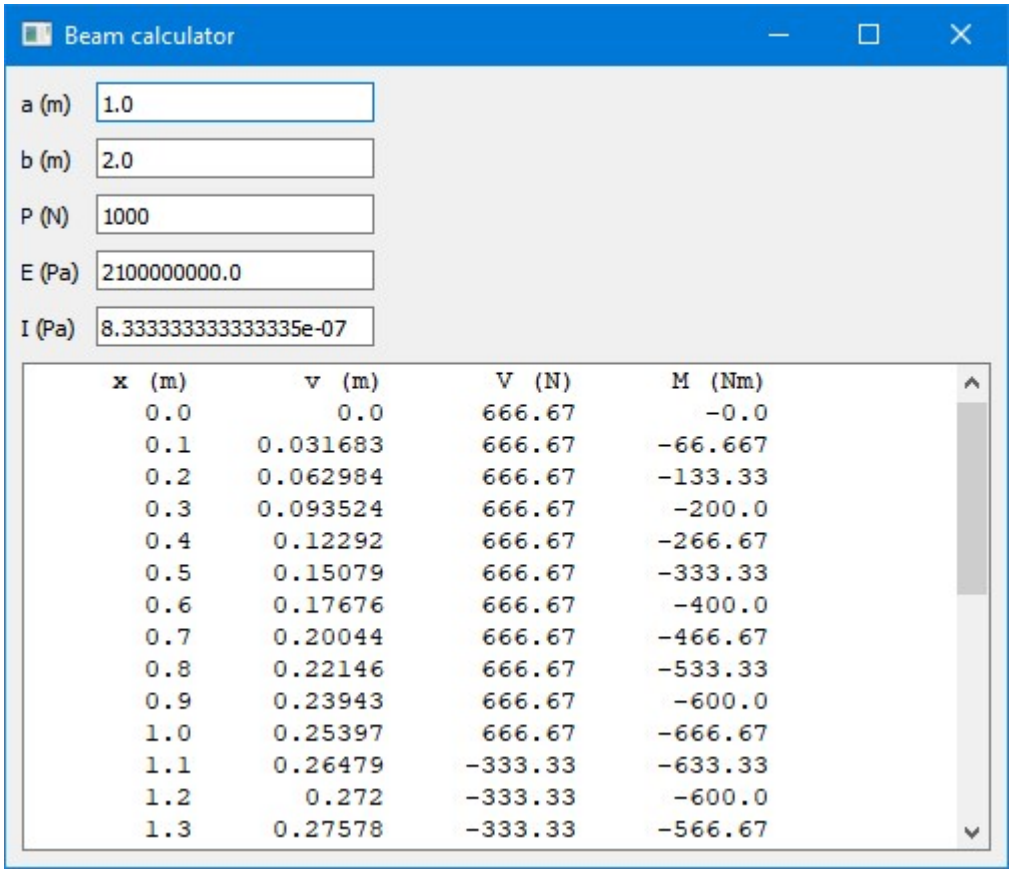
Example – Beam model class

Section force calculation methods

```
class BeamSimplySupported:

    ...

 def v(self, x):
        """Compute deflection of at x"""

        a = self.__a
        b = self.__b
        L = self.__L
        P = self.__P
        E = self.__E
        I = self.__I

        if x < a:
            return (P*b*L/(6*E*I))*((1-b**2/L**2)*x - x**3/L**2)
        else:
            return (P*a/(6*E*I))*(-a**2+(2*L+a**2/L)*x - 3*x**2+x**3/L)

    def V(self, x):
        """Section forces at x"""

        a = self.__a
        b = self.__b
        L = self.__L
        P = self.__P

        if x < a:
            return P*b/L
        else:
            return -P*a/L

    def M(self, x):
        """Moment at x"""

        a = self.__a
        b = self.__b
        L = self.__L
        P = self.__P

        if x < a:
            return -P*b*x/L
        else:
            return -P*a*(L-x)/L
```

Assign class attributes to shorter variable references. Easier to write formulas

User interface



x (m)	v (m)	V (N)	M (Nm)
0.0	0.0	666.67	-0.0
0.1	0.031683	666.67	-66.667
0.2	0.062984	666.67	-133.33
0.3	0.093524	666.67	-200.0
0.4	0.12292	666.67	-266.67
0.5	0.15079	666.67	-333.33
0.6	0.17676	666.67	-400.0
0.7	0.20044	666.67	-466.67
0.8	0.22146	666.67	-533.33
0.9	0.23943	666.67	-600.0
1.0	0.25397	666.67	-666.67
1.1	0.26479	-333.33	-633.33
1.2	0.272	-333.33	-600.0
1.3	0.27578	-333.33	-566.67

Example – Beam window class

BeamWindow constructor

```
import sys

from qtpy.QtWidgets import *
from qtpy.QtGui import *

import beam_model as bm

class BeamWindow(QWidget):
    """Main window class"""

    def __init__(self):
        """BeamWindow constructor"""
        super().__init__()

        # Create model instance

        self.beam = bm.BeamSimplySupported()

        # Initialise user interface

        self.init_gui()

        # Update controls with values from model instance

        self.update_controls()
        self.update_text_edit()
```

Import beam model class

Example – Beam window class

Create user interface controls

```
class BeamWindow(QWidget):

    ...

    def init_gui(self):
        """Initialise user interface"""

        self.resize(500, 400)
        self.move(50, 50)
        self.setWindowTitle("Beam calculator")

        # Create controls

        self.a_label = QLabel("a (m)")
        self.a_edit = QLineEdit()

        self.b_label = QLabel("b (m)")
        self.b_edit = QLineEdit()

        self.P_label = QLabel("P (N)")
        self.P_edit = QLineEdit()

        self.E_label = QLabel("E (Pa)")
        self.E_edit = QLineEdit()

        self.I_label = QLabel("I (Pa)")
        self.I_edit = QLineEdit()

        self.text_edit = QTextEdit("")
        self.text_edit.setFont(QFont("Courier", 6))
```

Example – Beam window class

Connect event editingFinished with event method

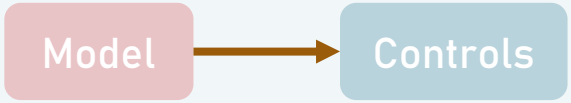
```
class BeamWindow(QWidget):

  ...

  def init_gui(self):
    """Initialise user interface"""

    ...

    # Connect events

    self.a_edit.editingFinished.connect(self.on_editing_finished)
    self.b_edit.editingFinished.connect(self.on_editing_finished)
    self.P_edit.editingFinished.connect(self.on_editing_finished)
    self.E_edit.editingFinished.connect(self.on_editing_finished)
    self.I_edit.editingFinished.connect(self.on_editing_finished)
```

Will be called when an edit is finished in the text box

Example – Beam window class

Methods for getting/setting control values

```
class BeamWindow(QWidget):

    ...

    def update_controls(self):
        """Update controls with model values"""

        self.a_edit.setText(str(self.beam.a))
        self.b_edit.setText(str(self.beam.b))
        self.P_edit.setText(str(self.beam.P))
        self.E_edit.setText(str(self.beam.E))
        self.I_edit.setText(str(self.beam.I))

    def update_text_edit(self):
        """Update text controls"""

        self.text_edit.clear()
        self.text_edit.append('{:>10}  {:>10}  {:>10}  {:>10}'.format("x (m)", "v (m)", "V (N)", "M (Nm)"))

        x = 0.0
        dx = 0.1

        while x < self.beam.L + dx:
            self.text_edit.append('{:10.5}  {:10.5}  {:10.5}  {:10.5}'.format(
                x, self.beam.v(x), self.beam.V(x), self.beam.M(x)))
            x += dx

        self.text_edit.moveCursor(QTextCursor.Start)

    def update_model(self):
        """Update model with values from controls"""

        self.beam.a = self.a_edit.text()
        self.beam.b = self.b_edit.text()
        self.beam.P = self.P_edit.text()
        self.beam.E = self.E_edit.text()
        self.beam.I = self.I_edit.text()
```



Example – Beam window class

Handle changes in text boxes

```
class BeamWindow(QWidget):

    ...

    def on_editing_finished(self):
        """Update """


        # Get current values from controls

        self.update_model()

        # Fill text box with beam section values.

        self.update_text_edit()

        # Make sure values in controls reflect the model .
        # If users enters invalid values they have to be removed.
        self.update_controls()
```

Needs to be called last as section values depends on updated model values

Qt Designer

Qt Designer

- Large user interfaces are complex
  - Difficult to implement directly in code
- Easier to layout interactively using a tool
- Qt supports a special XML format, which can describe the user interface – ui-files
- UI-files can be read by Qt and the user interface elements can be instantiated automatically as objects
- Qt Designer is a tool for creating user interfaces graphically and exporting them as ui-files

LUND UNIVERSITY

Example – Loading a ui-file

form.py

```
# -*- coding: utf-8 -*-

import sys

from qtpy.QtWidgets import *
from qtpy import uic

class MainWindow(QWidget):
    """Main window class for the Flow application"""

    def __init__(self):
        """Class constructor"""

        super().__init__()

        # Load and show our user interface

        self.init_gui()

    def init_gui(self):
        """Initialisera gränssnitt"""

        uic.loadUi("form4.ui", self)

        self.push_button.setText("Press me!")


if __name__ == '__main__':

    app = QApplication(sys.argv)

    window = MainWindow()
    window.show()

    sys.exit(app.exec_())
```

Loads ui-file and creates user interface objects

LUND
UNIVERSITY