PYQT'S MODEL/VIEW FRAMEWORK -- A QUICK OVERVIEW

Chen Chun-Chia

Qt / PyQt / PySide

A Simple PyQt App

```
import sys
from PyQt4 import QtGui, QtCore


app = QtGui.QApplication(sys.argv)
```

Your GUI Widgets

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

A Simple PyQt App

OR

Not Recommend

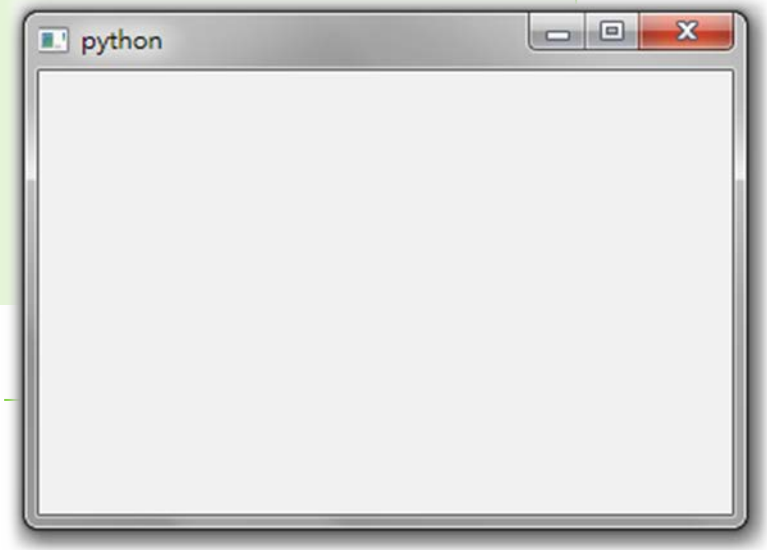
```
import sys
from PyQt4.QtGui import *
from PyQt4.QtCore import *

app = QApplication(sys.argv)
```

Your GUI Widgets

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

A Simple PyQt App – Example

```
import sys
from PyQt4.QtGui import *
from PyQt4.QtCore import *

app = QApplication(sys.argv)


win = QWidget()

win.show()


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



Customize Widget

or any other widget you want

```
class AWidget(QWidget):
    def __init__(self, parent=None):
```

Customize Widget ~ super.__init__

```
class AWidget(QWidget):
    def __init__(self, parent=None):
        super(AWidget, self).__init__(parent)
```

remember to call superclass.__init__()

Customize Widget ~ more Fields / Methods

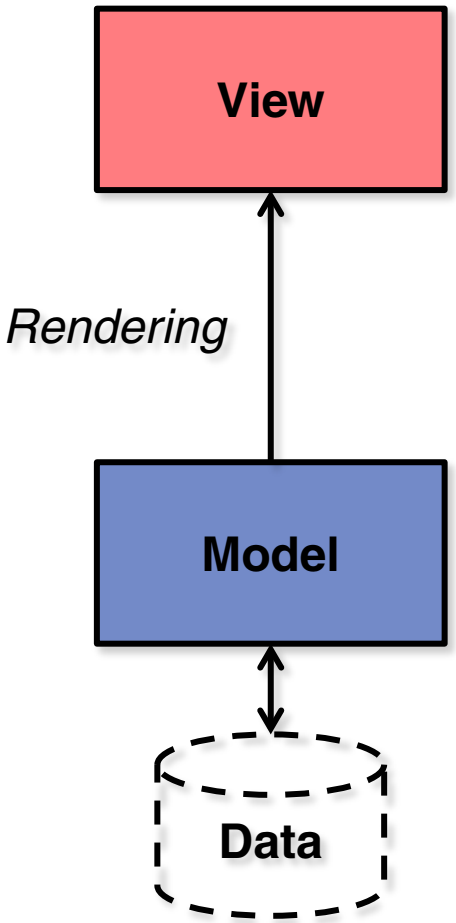
```
class AWidget(QWidget):
    def __init__(self, parent=None):
        super(AWidget, self).__init__(parent)
        self.a_field = ...

    def a_method(self, ...):
        ...
```

Customize Widget ~ Show

```
class AWidget(QWidget):
    def __init__(self, parent=None):
        super(AWidget, self).__init__(parent)
        self.a_field = ...

    def a_method(self, ...):
        ...
```

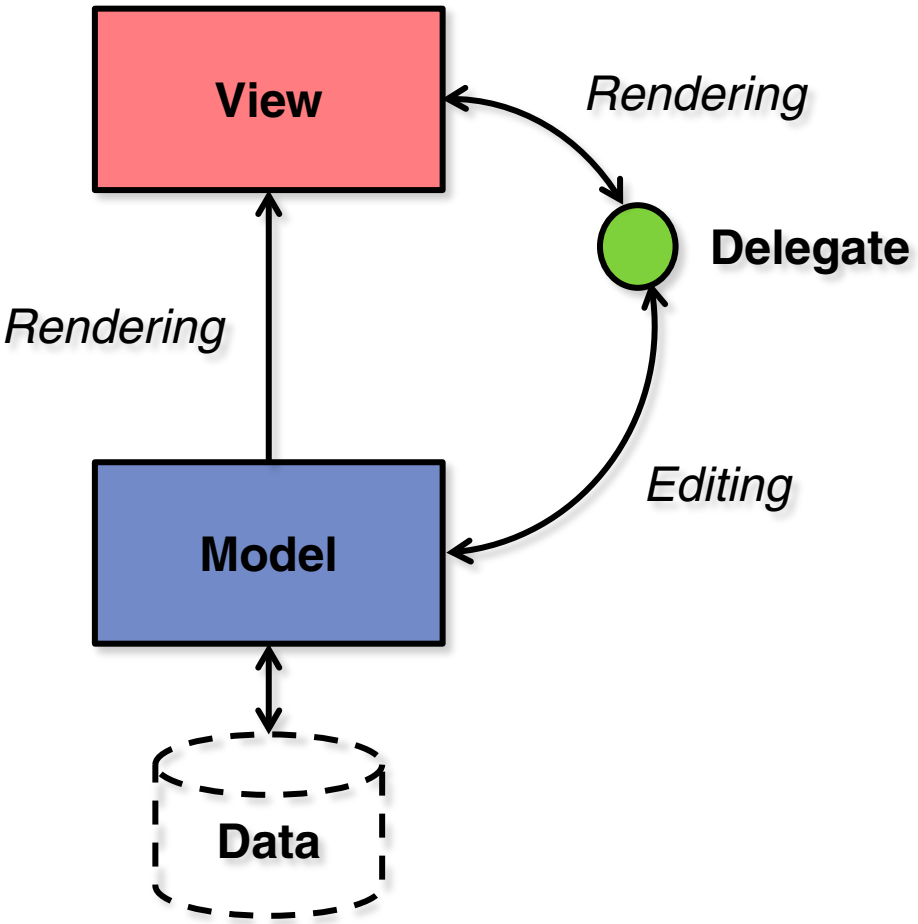
```
app = QApplication(sys.argv)

win = AWidget()
win.show()

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

Model / View Architecture

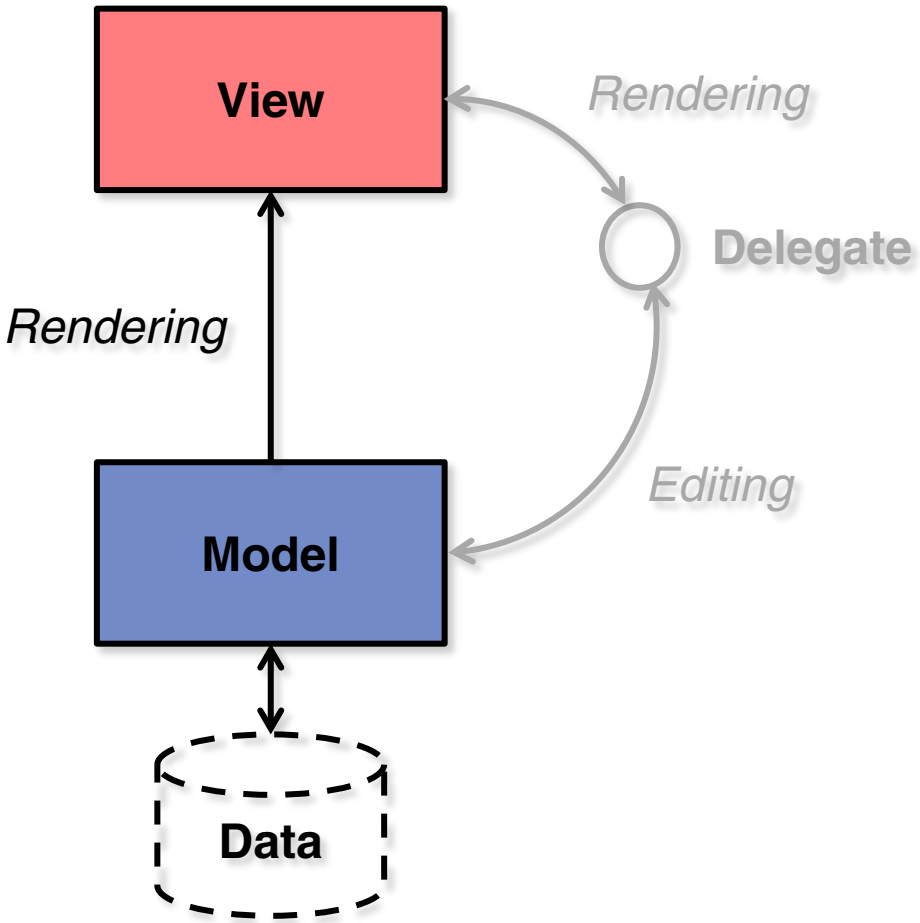


Model / View Architecture

with Delegate

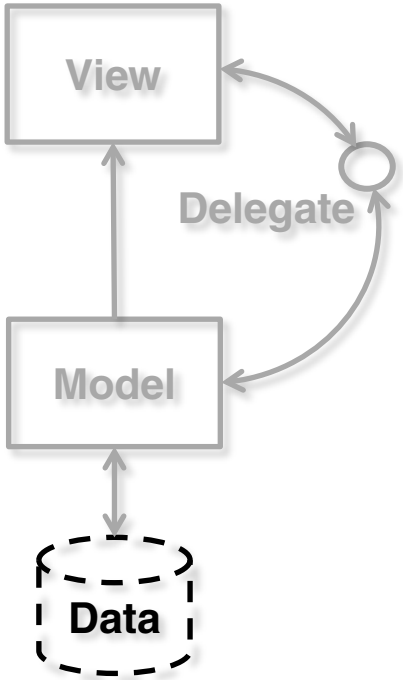


Display Data



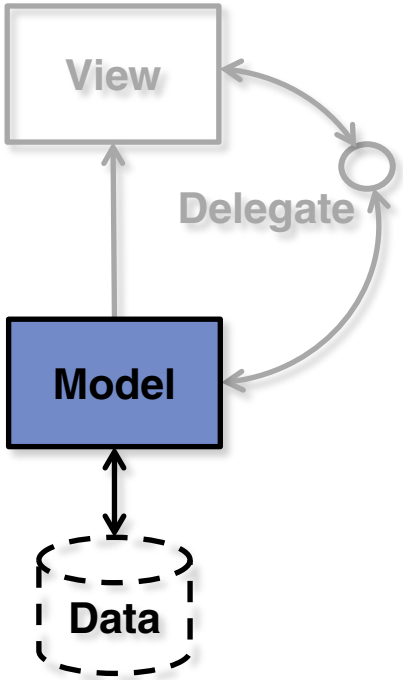
Example ~ A List

```
data = [70, 90, 20, 50]
```



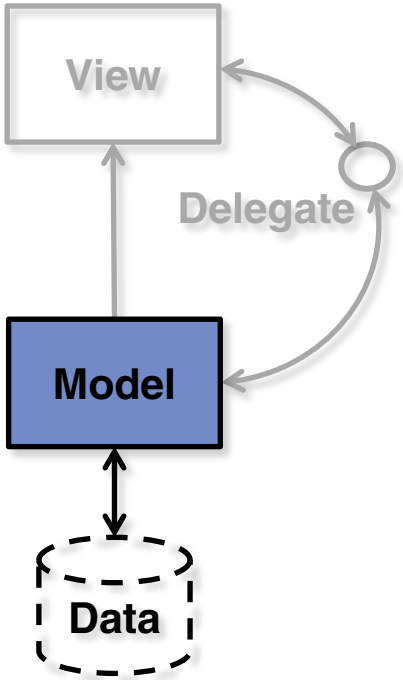
Build List Model

```
class MyListModel(QAbstractListModel):
    def __init__(self, parent=None):



    def rowCount(self, parent=QModelIndex()):


    def data(self, index, role=Qt.DisplayRole):
```



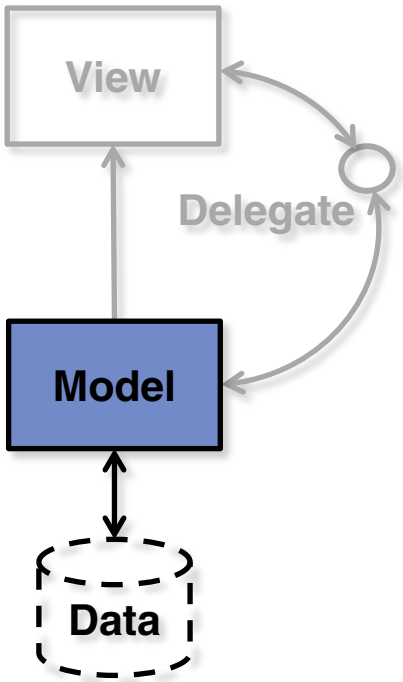
Build List Model

```
class MyListModel(QAbstractListModel):
    def __init__(self, parent=None):
        super(MyListModel, self).__init__(parent)
        self._data = [70, 90, 20, 50]

    def rowCount(self, parent=QModelIndex()):


    def data(self, index, role=Qt.DisplayRole):
```



Build List Model ~ rowCount

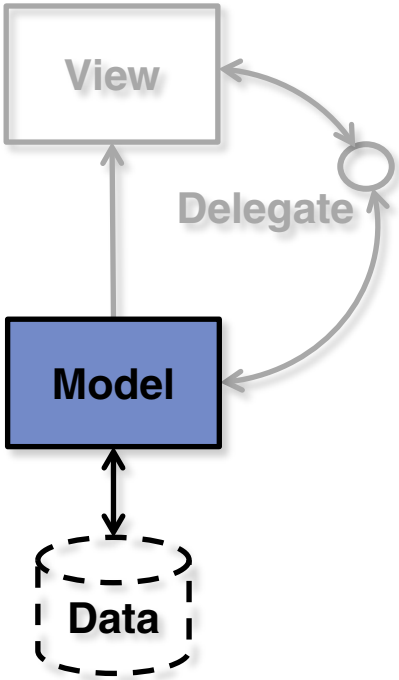
```
class MyListModel(QAbstractListModel):
    def __init__(self, parent=None):
        super(MyListModel, self).__init__(parent)
        self._data = [70, 90, 20, 50]

    def rowCount(self, parent=QModelIndex()):
        return len(self._data)  [int]

    def data(self, index, role=Qt.DisplayRole):
```



Build List Model ~ data

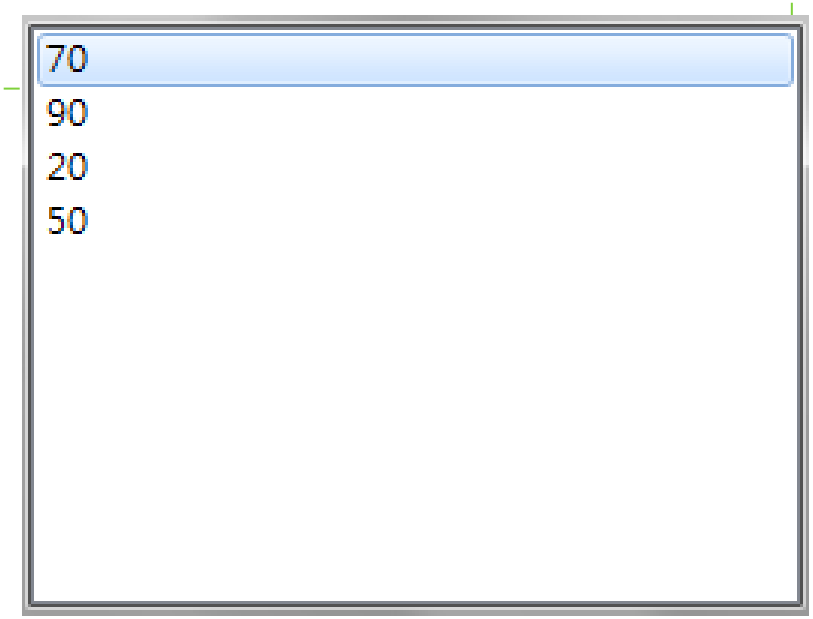
```
class MyListModel(QAbstractListModel):
    def __init__(self, parent=None):
        super(MyListModel, self).__init__(parent)
        self._data = [70, 90, 20, 50]

    def rowCount(self, parent=QModelIndex()):
        return len(self._data)

    def data(self, index, role=Qt.DisplayRole):
        if not index.isValid() or \
           not 0 <= index.row() < self.rowCount():
            return QVariant()

        row = index.row()
        if role == Qt.DisplayRole:
            return str(self._data[row])

        return QVariant()
```

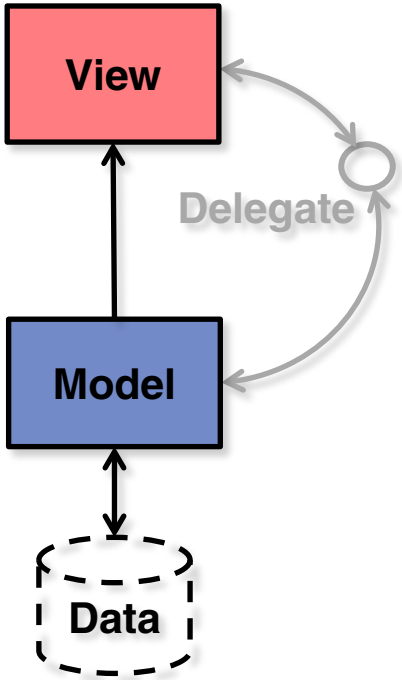
[str / QString]



Show List View

```
app = QApplication(sys.argv)

model = MyListModel()

view = QListView()
view.setModel(model)
view.show()

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





Display Data with Roles

```
class MyListModel(QAbstractListModel):
    def __init__(self, parent=None):
        super(MyListModel, self).__init__(parent)
        self._data = [70, 90, 20, 50]

    def rowCount(self, parent=QModelIndex()):
        return len(self._data)

    def data(self, index, role=Qt.DisplayRole):
        if not index.isValid() or \
           not 0 <= index.row() < self.rowCount():
            return QVariant()

        row = index.row()
        if role == Qt.DisplayRole:
            return str(self._data[row])

        return QVariant()
```

Qt.ItemDataRole

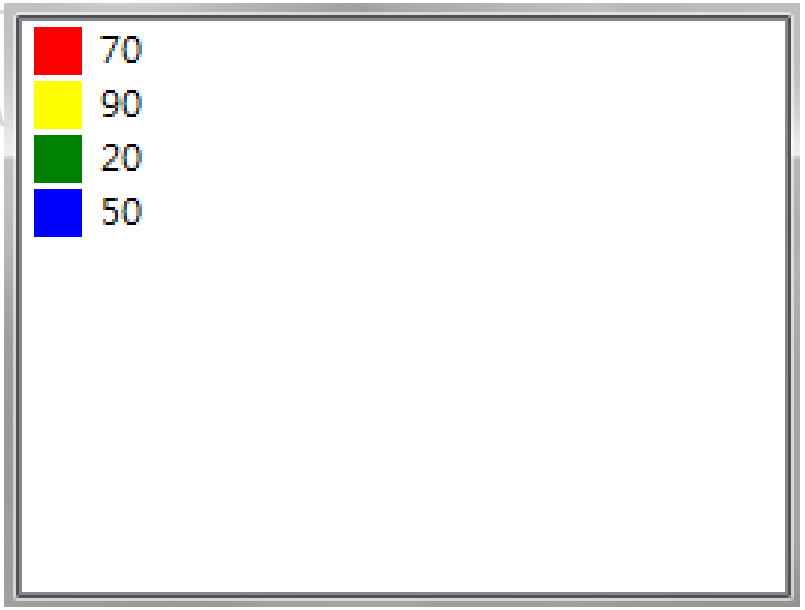
- General purpose roles
  - Qt.DisplayRole Text of the item (QString)
  - Qt.DecorationRole Icon of the item (QColor, QIcon or QPixmap)
  - Qt.EditRole Editing data for editor (QString)
  - Qt.ToolTipRole Tooltip of the item (QString)
  - Qt.StatusTipRole Text in the status bar (QString)
  - Qt.WhatsThisRole Text in "What's This?" mode (QString)
  - Qt.SizeHintRole Size hint in view (QSize)

Qt.ItemDataRole (More…)

- Roles describing appearance and meta data:
  - Qt.FontRole Font of the item (QFont)
  - Qt.TextAlignmentRole Text alignment of the item (Qt.AlignmentFlag)
  - Qt.BackgroundRole Background of the item (QBrush)
  - Qt.ForegroundRole Foreground of the item (QBrush)
  - Qt.CheckStateRole Checked state of the item (Qt.CheckState)
- Accessibility roles:
  - Qt.AccessibleTextRole Text for accessibility (QString)
  - Qt.AccessibleDescriptionRole Description for accessibility (QString)
- User roles:
  - Qt.UserRole 1st role for specific purposes

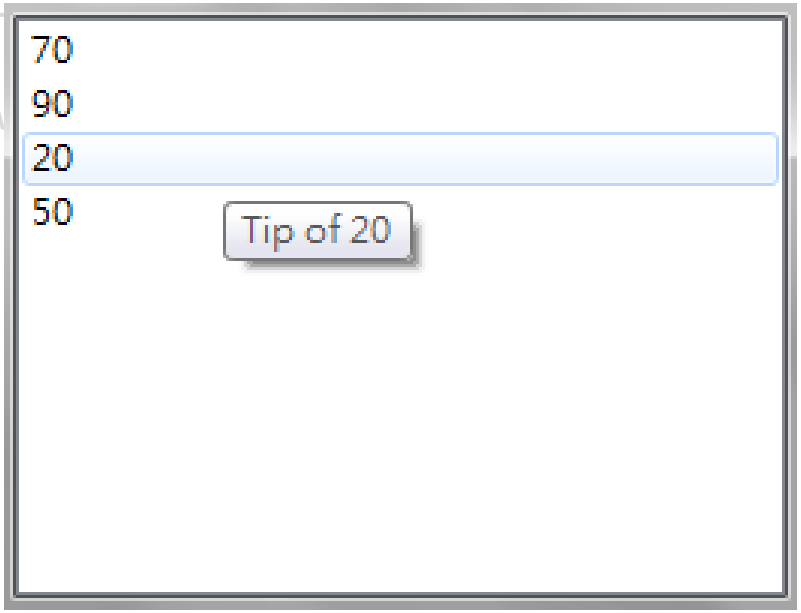
Qt.DecorationRole

- General purpose roles
  - Qt.DisplayRole Text of the item (QString)
  - Qt.DecorationRole Icon of the item (QColor, QIcon or QPixmap)
  - Qt.EditRole Editing data for editor (QString)
  - Qt.ToolTipRole Tooltip of the item (QString)
  - Qt.StatusTipRole Text in the status bar (QString)
  - Qt.WhatsThisRole Text in "What's T
  - Qt.SizeHintRole Size hint in view



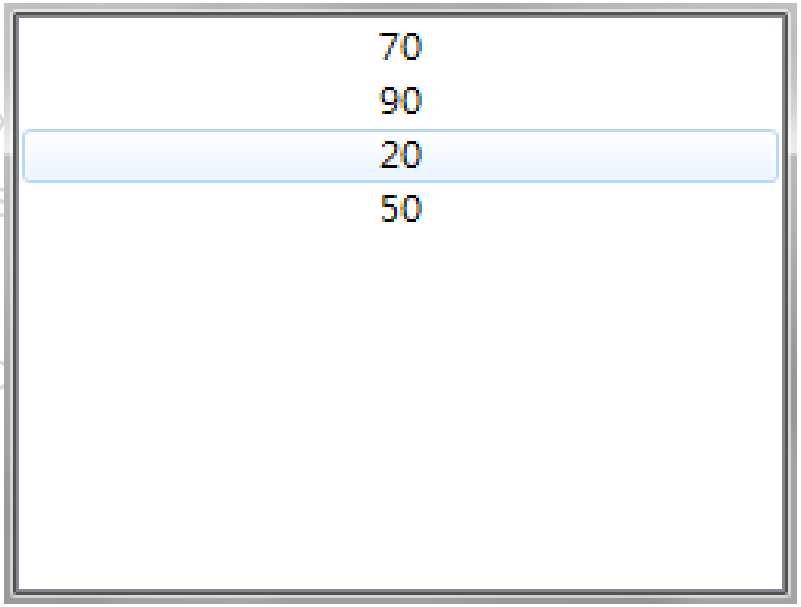
Qt.ToolTipRole

- General purpose roles
  - Qt.DisplayRole Text of the item (QString)
  - Qt.DecorationRole Icon of the item (QColor, QIcon or QPixmap)
  - Qt.EditRole Editing data for editor (QString)
  - Qt.ToolTipRole Tooltip of the item (QString)
  - Qt.StatusTipRole Text in the status bar (QString)
  - Qt.WhatsThisRole Text in "What's T
  - Qt.SizeHintRole Size hint in view



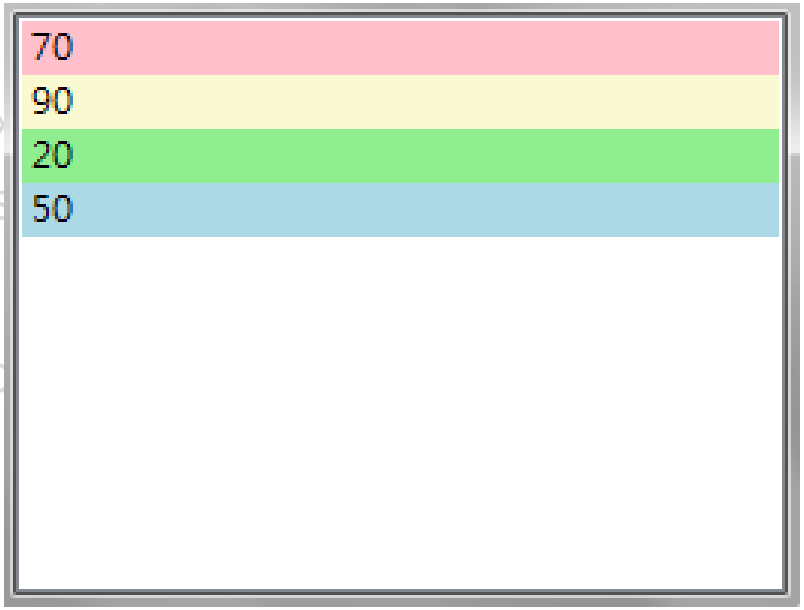
Qt.TextAlignmentRole

- Roles describing appearance and meta data:
  - Qt.FontRole Font of the item (QFont)
  - Qt.TextAlignmentRole Text alignment of the item (Qt.AlignmentFlag)
  - Qt.BackgroundRole Background of the item (QBrush)
  - Qt.ForegroundRole Foreground of the item (QBrush)
  - Qt.CheckStateRole Checked state of the item (Qt.CheckState)
- Accessibility roles:
  - Qt.AccessibleTextRole Tex
  - Qt.AccessibleDescriptionRole Des
- User roles:
  - Qt.UserRole 1st role for spec



Qt.BackgroundRole

- Roles describing appearance and meta data:
  - Qt.FontRole Font of the item (QFont)
  - Qt.TextAlignmentRole Text alignment of the item (Qt.AlignmentFlag)
  - Qt.BackgroundRole Background of the item (QBrush)
  - Qt.ForegroundRole Foreground of the item (QBrush)
  - Qt.CheckStateRole Checked state of the item (Qt.CheckState)
- Accessibility roles:
  - Qt.AccessibleTextRole Tex
  - Qt.AccessibleDescriptionRole Des
- User roles:
  - Qt.UserRole 1st role for spec

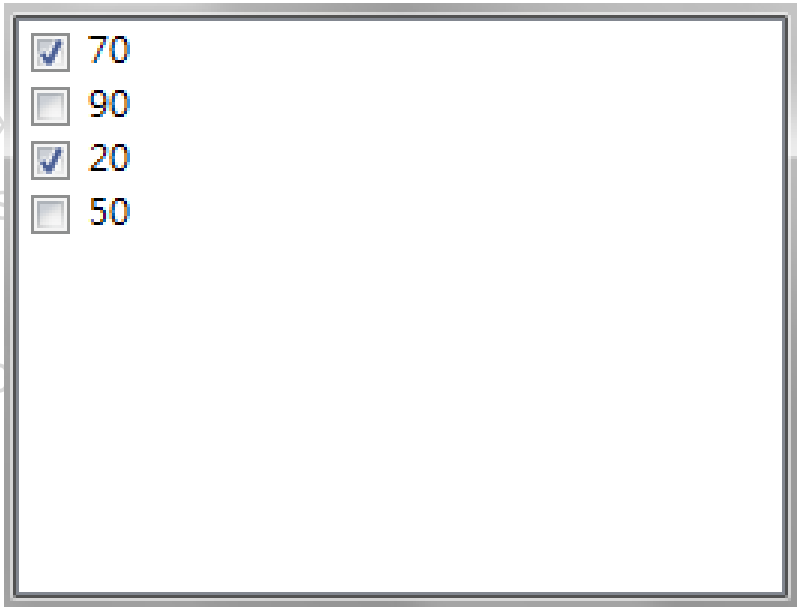


Qt.ForegroundRole

- Roles describing appearance and meta data:
  - Qt.FontRole Font of the item (QFont)
  - Qt.TextAlignmentRole Text alignment of the item (Qt.AlignmentFlag)
  - Qt.BackgroundRole Background of the item (QBrush)
  - Qt.ForegroundRole Foreground of the item (QBrush)
  - Qt.CheckStateRole Checked state of the item (Qt.CheckState)
- Accessibility roles:
  - Qt.AccessibleTextRole Tex
  - Qt.AccessibleDescriptionRole Des
- User roles:
  - Qt.UserRole 1st role for spec

Qt.CheckStateRole

- Roles describing appearance and meta data:
  - Qt.FontRole Font of the item (QFont)
  - Qt.TextAlignmentRole Text alignment of the item (Qt.AlignmentFlag)
  - Qt.BackgroundRole Background of the item (QBrush)
  - Qt.ForegroundRole Foreground of the item (QBrush)
  - Qt.CheckStateRole Checked state of the item (Qt.CheckState)
- Accessibility roles:
  - Qt.AccessibleTextRole Tex
  - Qt.AccessibleDescriptionRole Des
- User roles:
  - Qt.UserRole 1st role for spec



Qt.UserRole

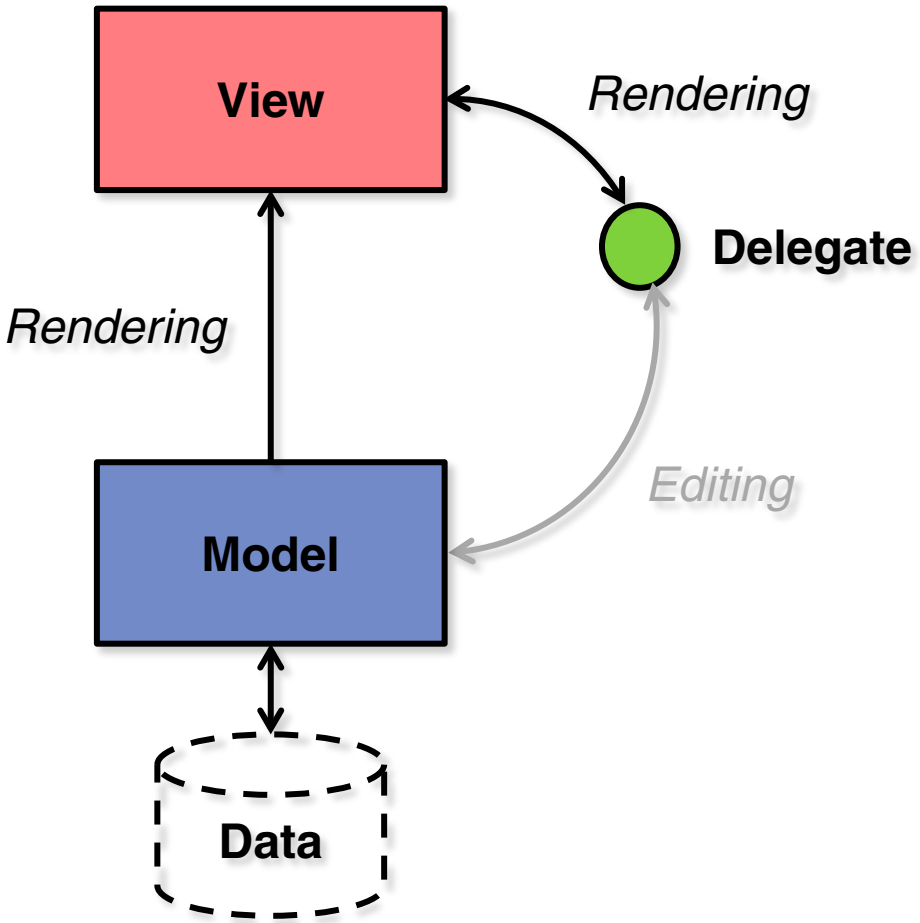
- Roles describing appearance and meta data:
  - Qt.FontRole Font of the item (QFont)
  - Qt.TextAlignmentRole Text alignment of the item (Qt.AlignmentFlag)
  - Qt.BackgroundRole Background of the item (QBrush)
  - Qt.ForegroundRole Foreground of the item (QBrush)
  - Qt.CheckStateRole Checked state of the item (Qt.CheckState)
- Accessibility roles:

For user roles, it is up to the developer to decide which types to use and ensure that components use the correct types when accessing and setting data.

- Qt.UserRole 1st role for specific purposes

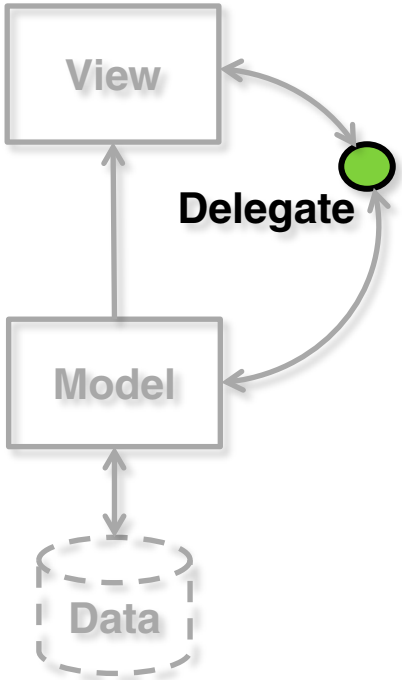
UserRole, UserRole+1, UserRole+2, ...

Display Data with Delegate



Build Item Delegate ~ paint

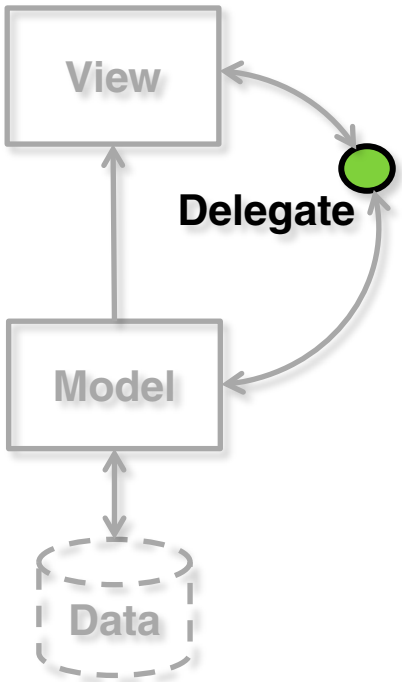
```
class MyDelegate(QStyledItemDelegate):
    def paint(self, painter, option, index):
```



Build Item Delegate ~ Get Model Data

```
class MyDelegate(QStyledItemDelegate):
    def paint(self, painter, option, index):
        item_var = index.data(Qt.DisplayRole)
        item_str = item_var.toPyObject()

        opts = QStyleOptionProgressBarV2()
        opts.rect = option.rect
        opts.minimum = 0
        opts.maximum = 100
        opts.text = item_str
        opts.textAlignment = Qt.AlignCenter
        opts.textVisible = True
        opts.progress = int(item_str)

        QApplication.style().drawControl(
                QStyle.CE_ProgressBar, opts, painter)
```

[QVariant]

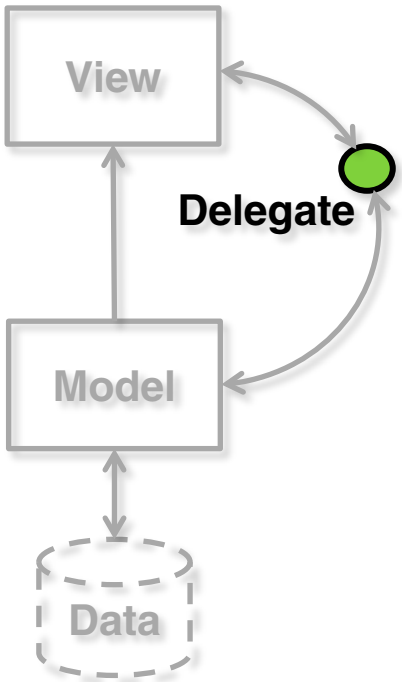


Build Item Delegate ~ Trans to Python Type

```
class MyDelegate(QStyledItemDelegate):
    def paint(self, painter, option, index):
        item_var = index.data(Qt.DisplayRole)
        item_str = item_var.toPyObject()
```

[Python Type of Item]

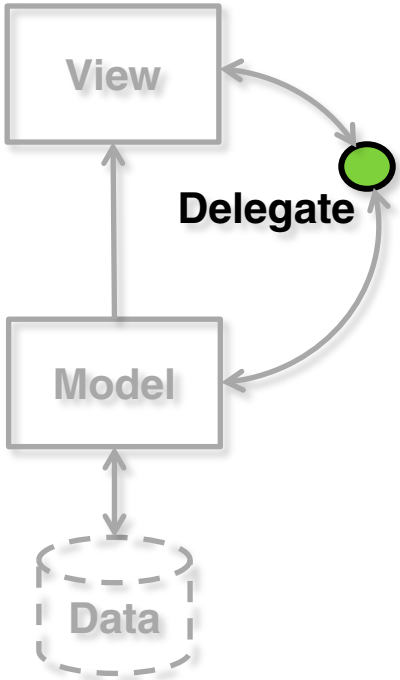
```
        opts = QStyleOptionProgressBarV2()
        opts.rect = option.rect
        opts.minimum = 0
        opts.maximum = 100
        opts.text = item_str
        opts.textAlignment = Qt.AlignCenter
        opts.textVisible = True
        opts.progress = int(item_str)

        QApplication.style().drawControl(
                QStyle.CE_ProgressBar, opts, painter)
```



Build Item Delegate ~ Draw Control

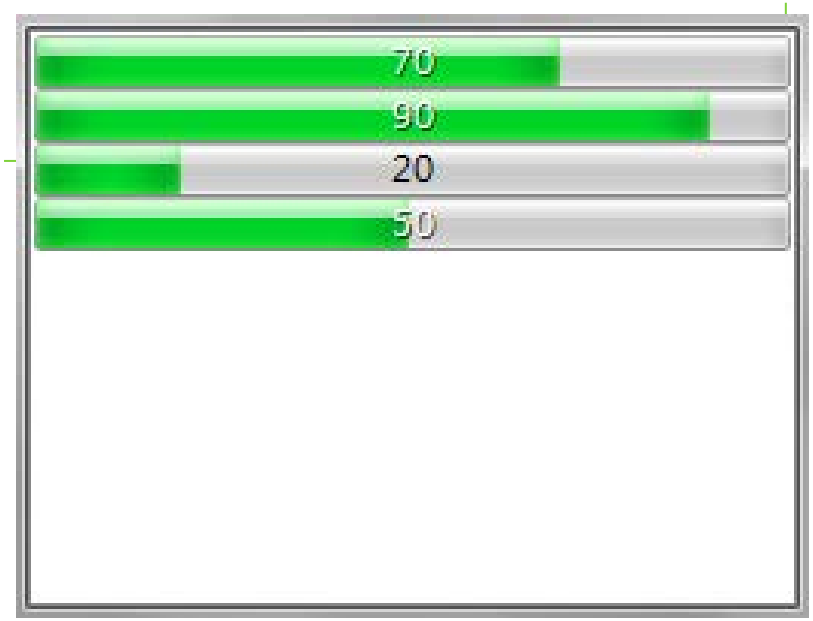
```
class MyDelegate(QStyledItemDelegate):
    def paint(self, painter, option, index):
        item_var = index.data(Qt.DisplayRole)
        item_str = item_var.toPyObject()

        opts = QStyleOptionProgressBarV2()
        opts.rect = option.rect
        opts.minimum = 0
        opts.maximum = 100
        opts.text = item_str
        opts.textAlignment = Qt.AlignCenter
        opts.textVisible = True
        opts.progress = int(item_str)

        QApplication.style().drawControl(
                QStyle.CE_ProgressBar, opts, painter)
```

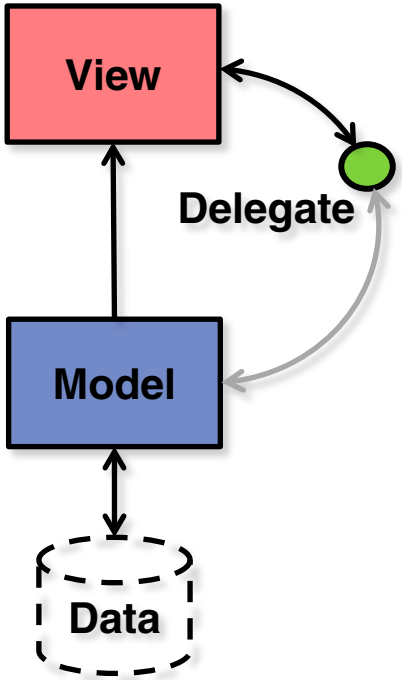
Draw Progress Bar



Set Delegate into View

```
app = QApplication(sys.argv)

model = MyListModel()
delegate = MyDelegate()

view = QListView()
view.setModel(model)
view.setItemDelegate(delegate)
view.show()

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

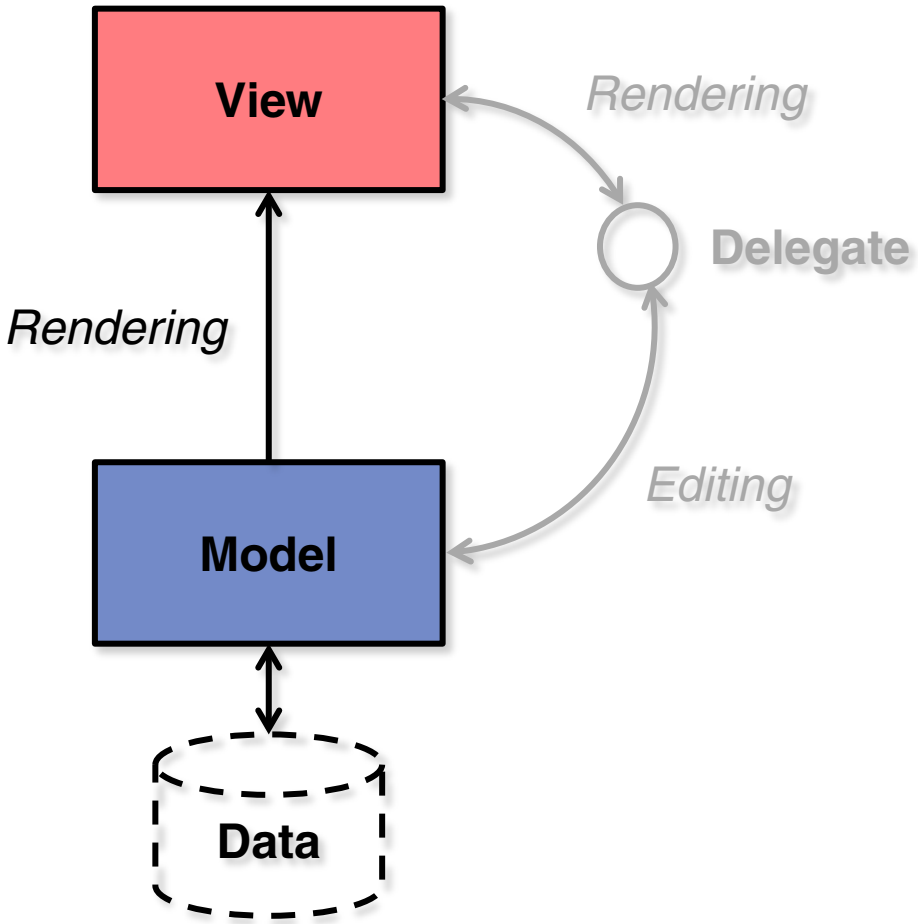




Edit Data

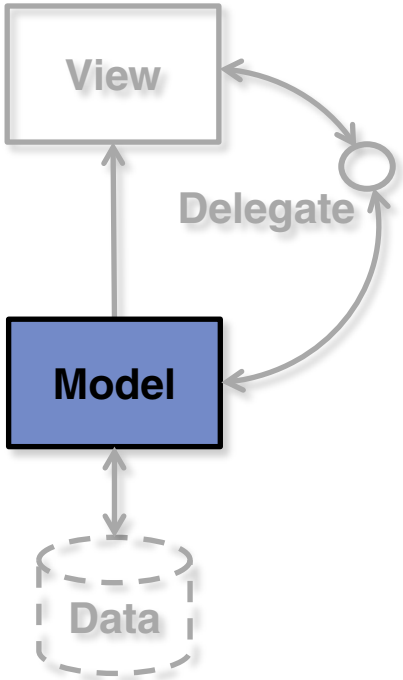
Use Default Editor

= **QLineEdit**



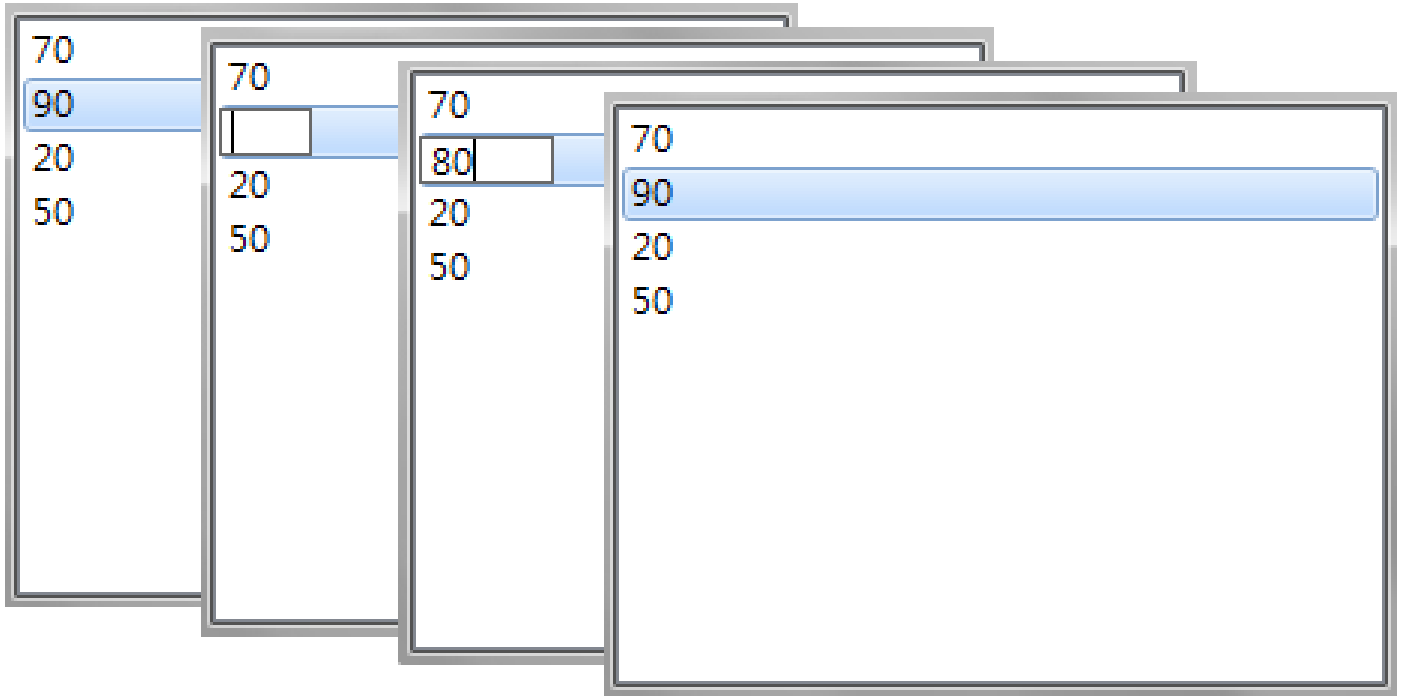
Activate Default Editor

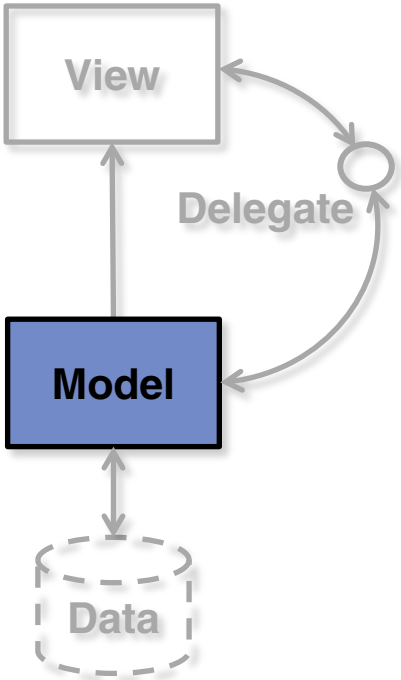
```
class MyListModel(QAbstractListModel):
    def flags(self, index):
        flag = super(MyListModel, self).flags(index)
        return flag | Qt.ItemIsEditable
```



Activate Default Editor

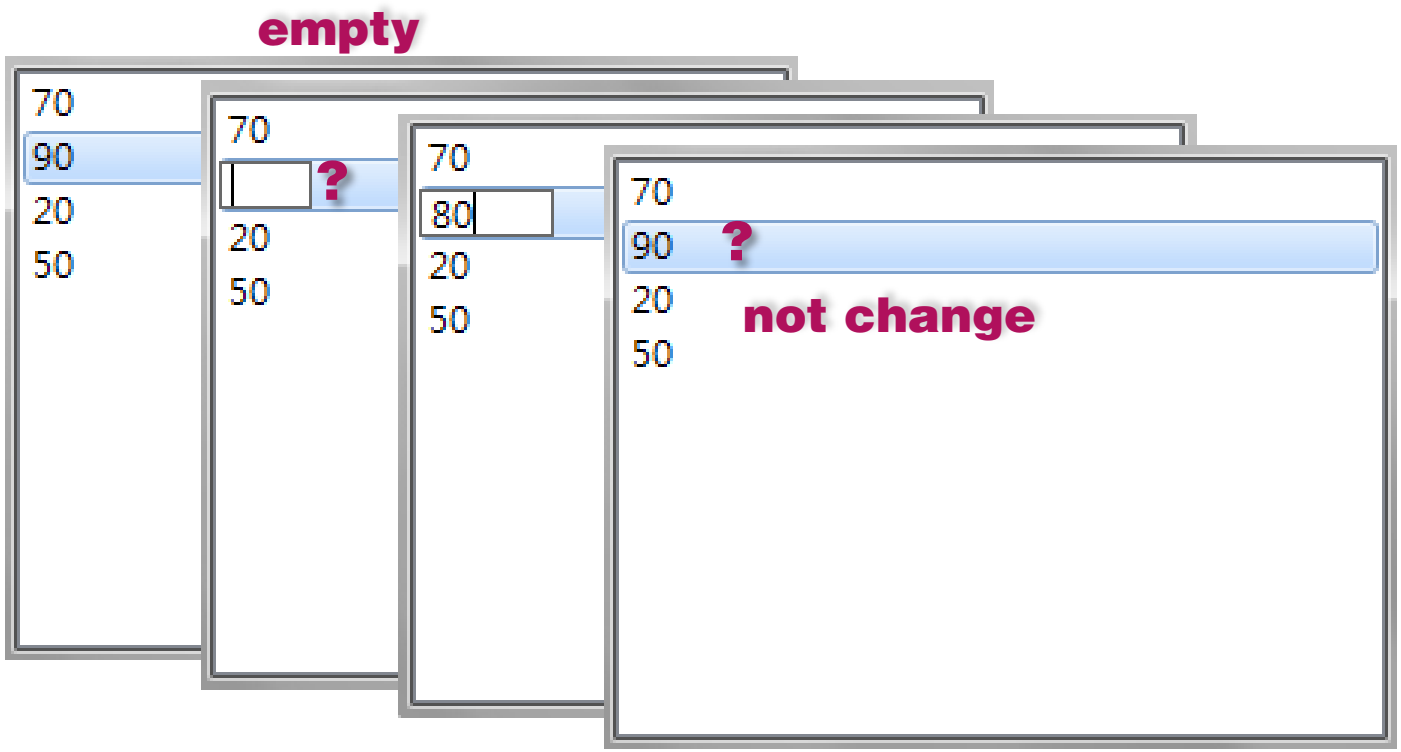
```
class MyListModel(QAbstractListModel):
    def flags(self, index):
        flag = super(MyListModel, self).flags(index)
        return flag | Qt.ItemIsEditable
```

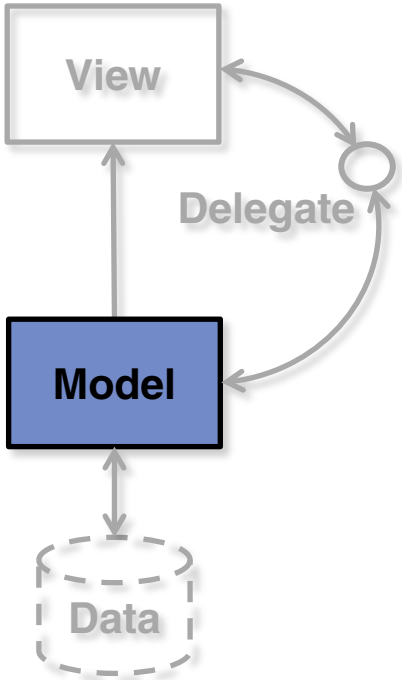




Activate Default Editor

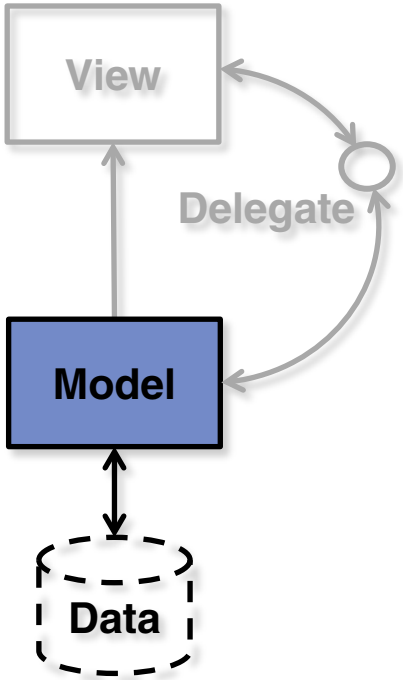
```
class MyListModel(QAbstractListModel):
    def flags(self, index):
        flag = super(MyListModel, self).flags(index)
        return flag | Qt.ItemIsEditable
```





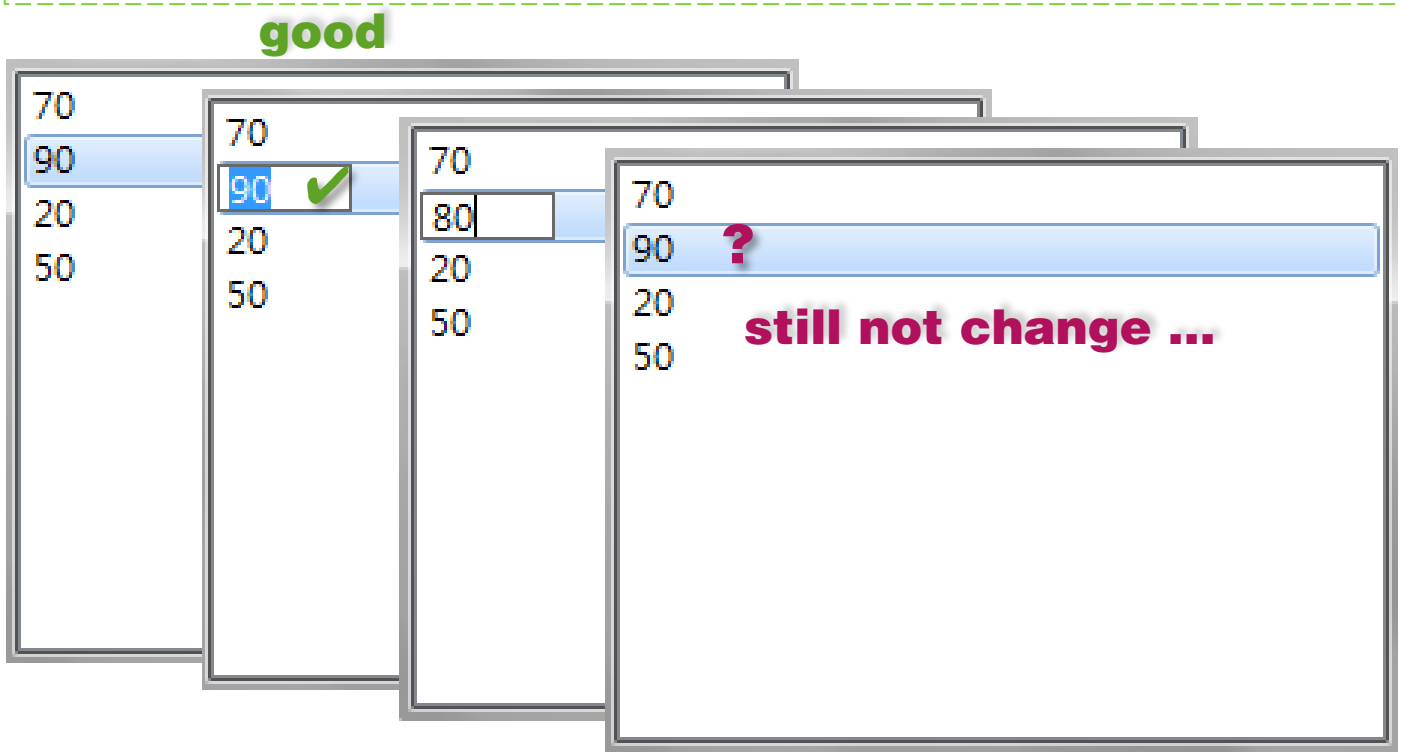
Load Model Data to Editor

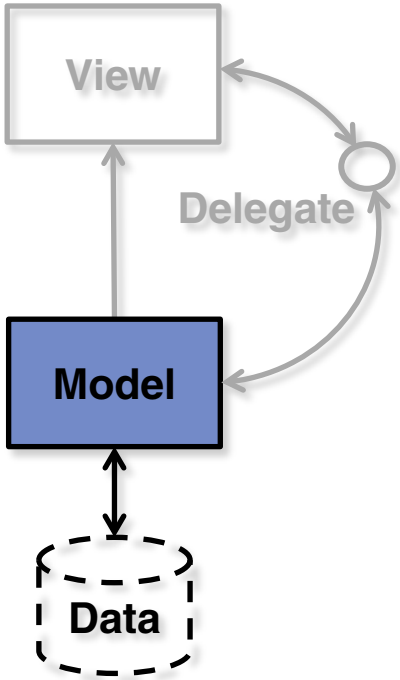
```
class MyListModel(QAbstractListModel):
    def data(self, index, role=Qt.DisplayRole):
        ...
        elif role == Qt.EditRole:
            return str(self._data[row])
        ...
```



Load Model Data to Editor

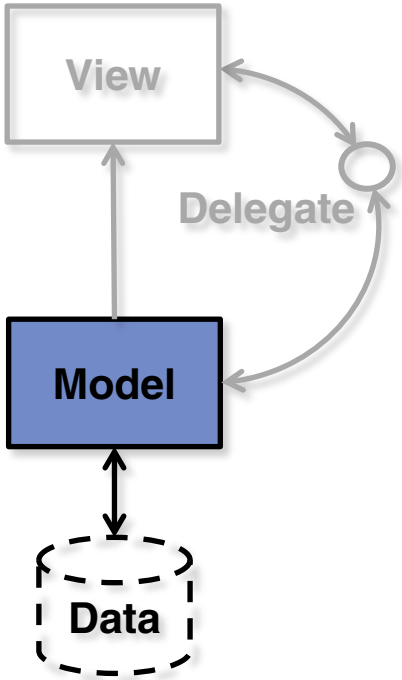
```
class MyListModel(QAbstractListModel):
    def data(self, index, role=Qt.DisplayRole):
        ...
        elif role == Qt.EditRole:
            return str(self._data[row])
        ...
```





After Editing ~ Set Data to Model

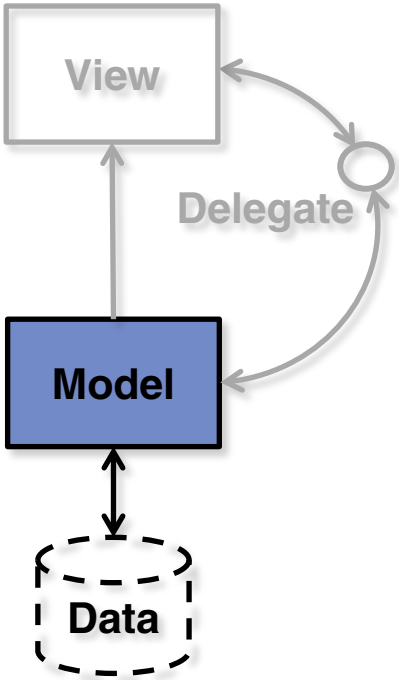
```
class MyListModel(QAbstractListModel):
    def setData(self, index, value, role=Qt.EditRole):
        ...
        if role == Qt.EditRole:
            value_int, ok = value.toInt()
            if ok:
                self._data[row] = value_int
                self.dataChanged.emit(index, index)
                return True
            return False
        ...
```



After Editing ~ Set Data to Model

```
class MyListModel(QAbstractListModel):
    def setData(self, index, value, role=Qt.EditRole):
        ...
        if role == Qt.EditRole:
            value_int, ok = value.toInt()
            if ok:
                self._data[row] = value_int
                self.dataChanged.emit(index, index)
                return True
            return False
        ...
```

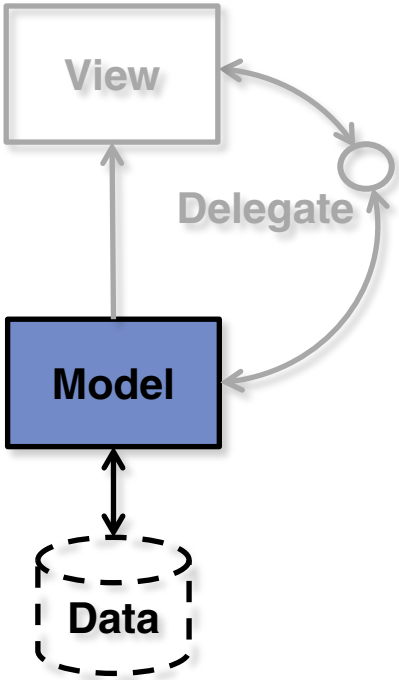
[QVariant → int]



After Editing ~ Set Data to Model

```
class MyListModel(QAbstractListModel):
    def setData(self, index, value, role=Qt.EditRole):
        ...
        if role == Qt.EditRole:
            value_int, ok = value.toInt()
            if ok:
                self._data[row] = value_int
                self.dataChanged.emit(index, index)
                return True
            return False
        ...
```

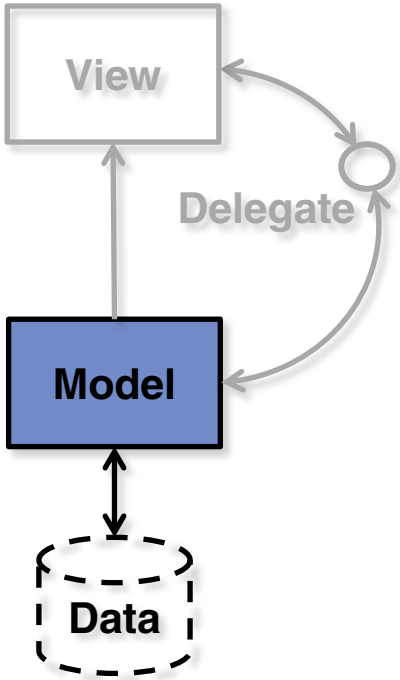
Update internal data



After Editing ~ Set Data to Model

```
class MyListModel(QAbstractListModel):
    def setData(self, index, value, role=Qt.EditRole):
        ...
        if role == Qt.EditRole:
            value_int, ok = value.toInt()
            if ok:
                self._data[row] = value_int
                self.dataChanged.emit(index, index)
                return True
            return False
        ...
```

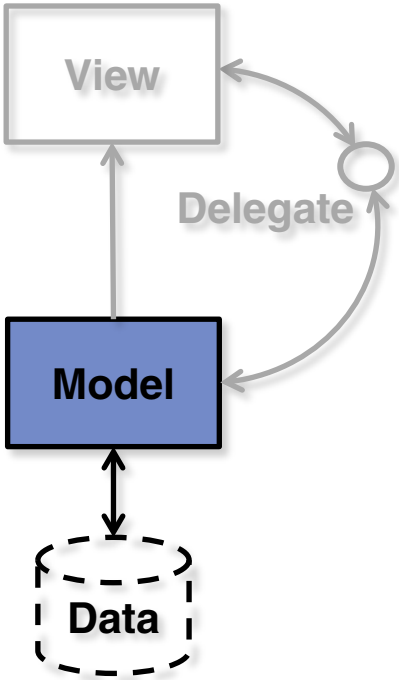
Notify view that data have changed



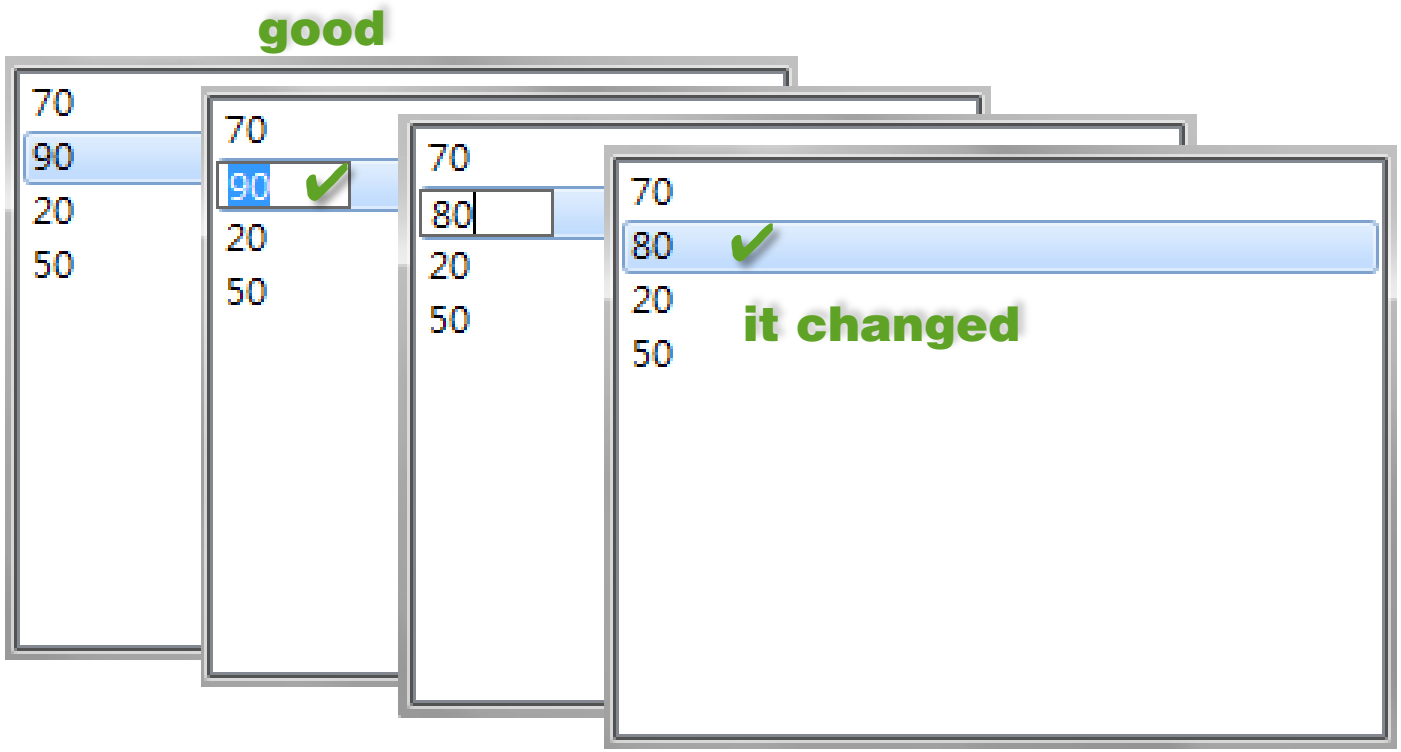
After Editing ~ Set Data to Model

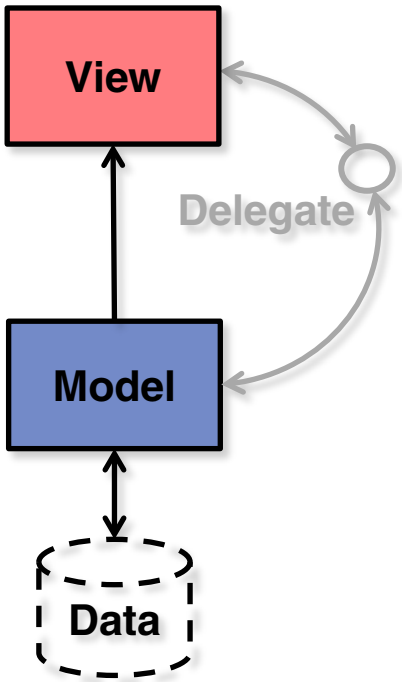
```
class MyListModel(QAbstractListModel):
    def setData(self, index, value, role=Qt.EditRole):
        ...
        if role == Qt.EditRole:
            value_int, ok = value.toInt()
            if ok:
                self._data[row] = value_int
                self.dataChanged.emit(index, index)
                return True
            return False
        ...
```

Return whether data have successfully set or not

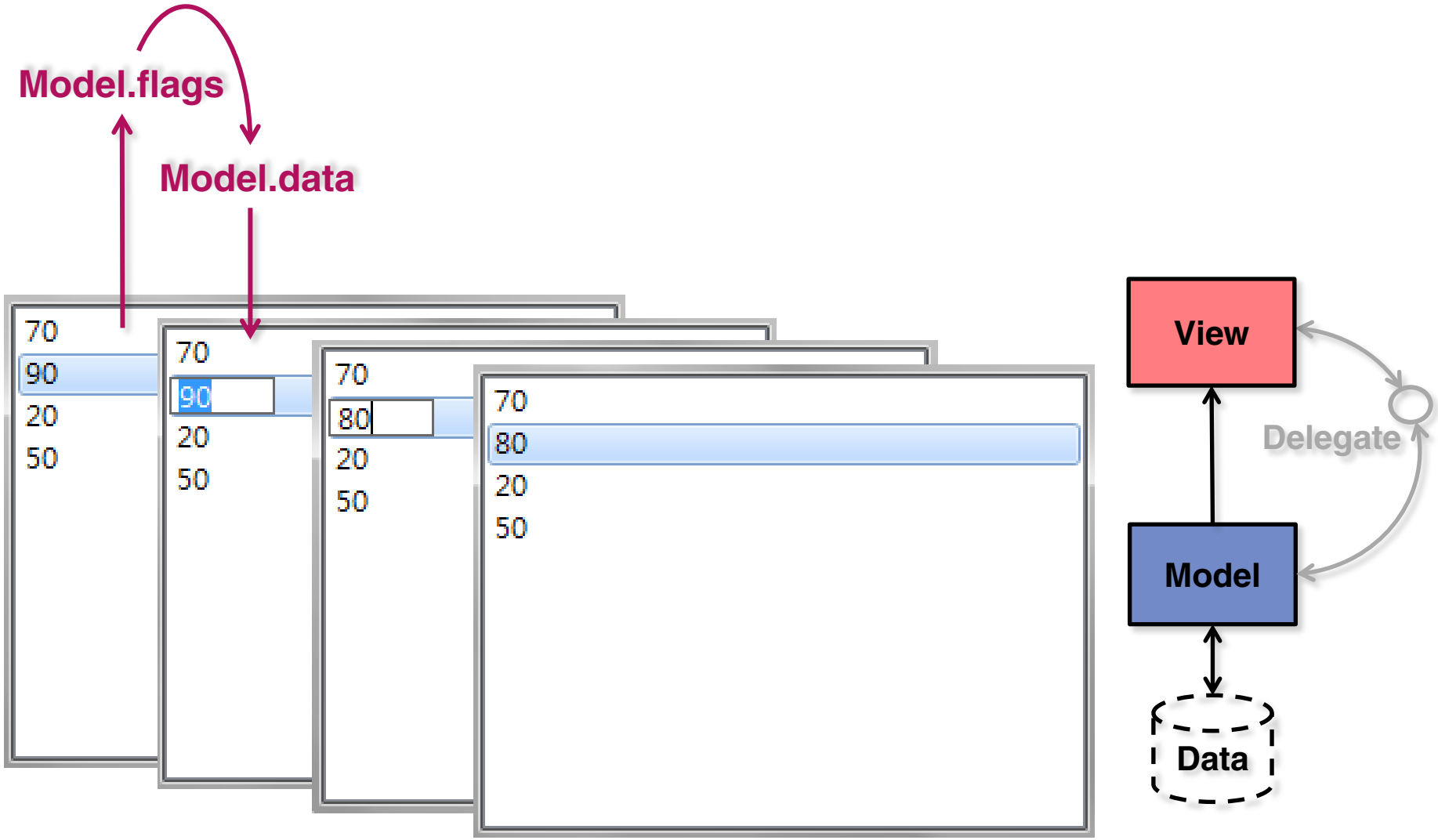


Edit Data with Default Editor

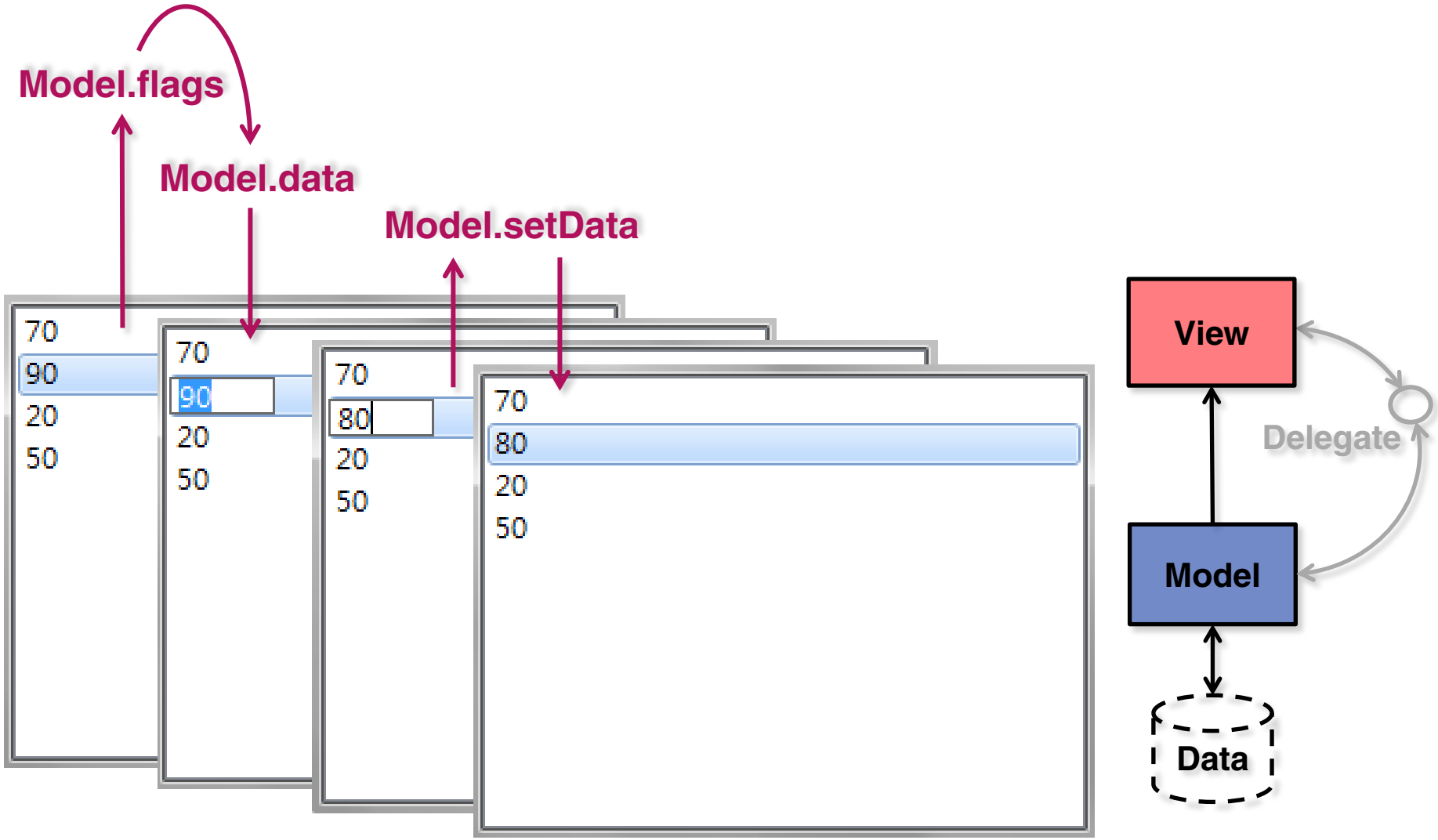




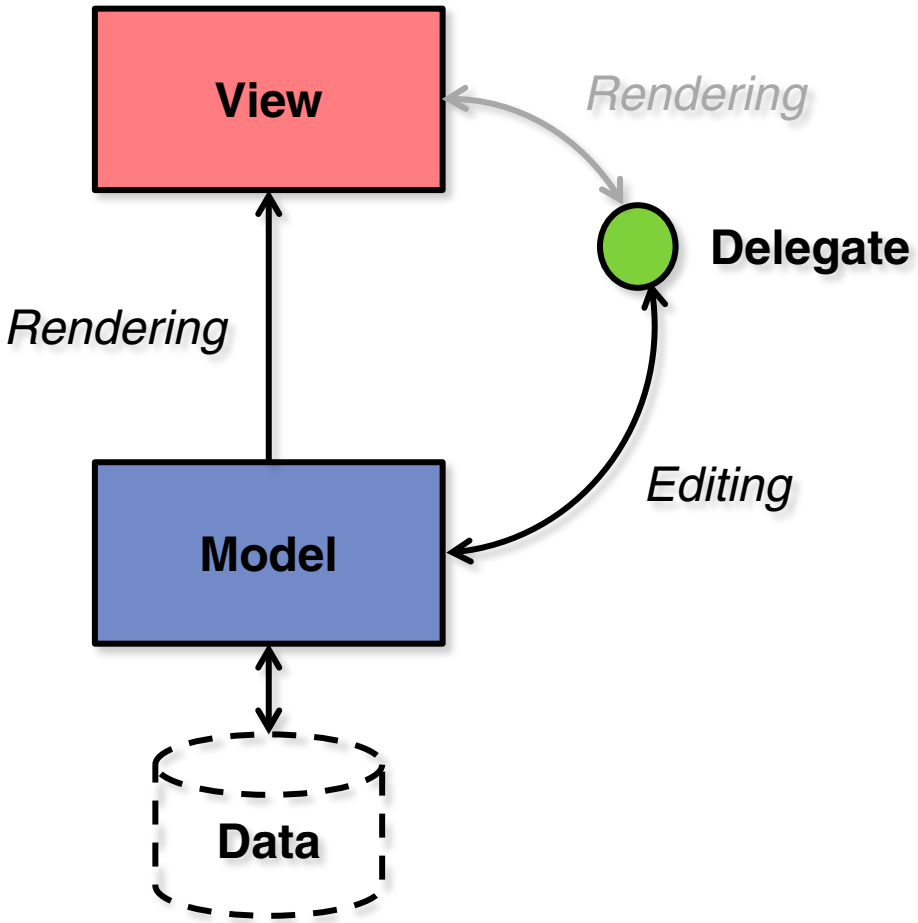
Edit Data with Default Editor



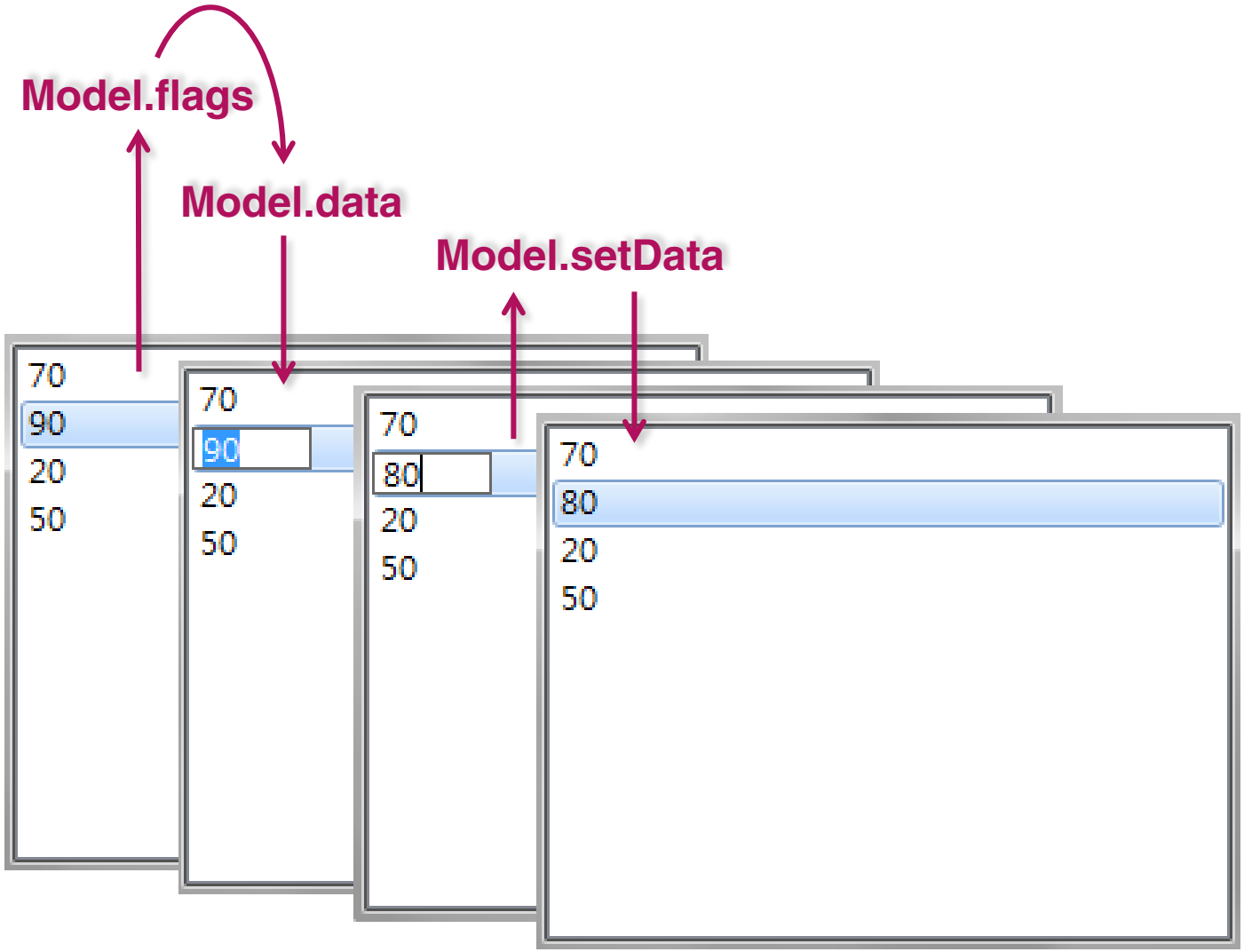
Edit Data with Default Editor



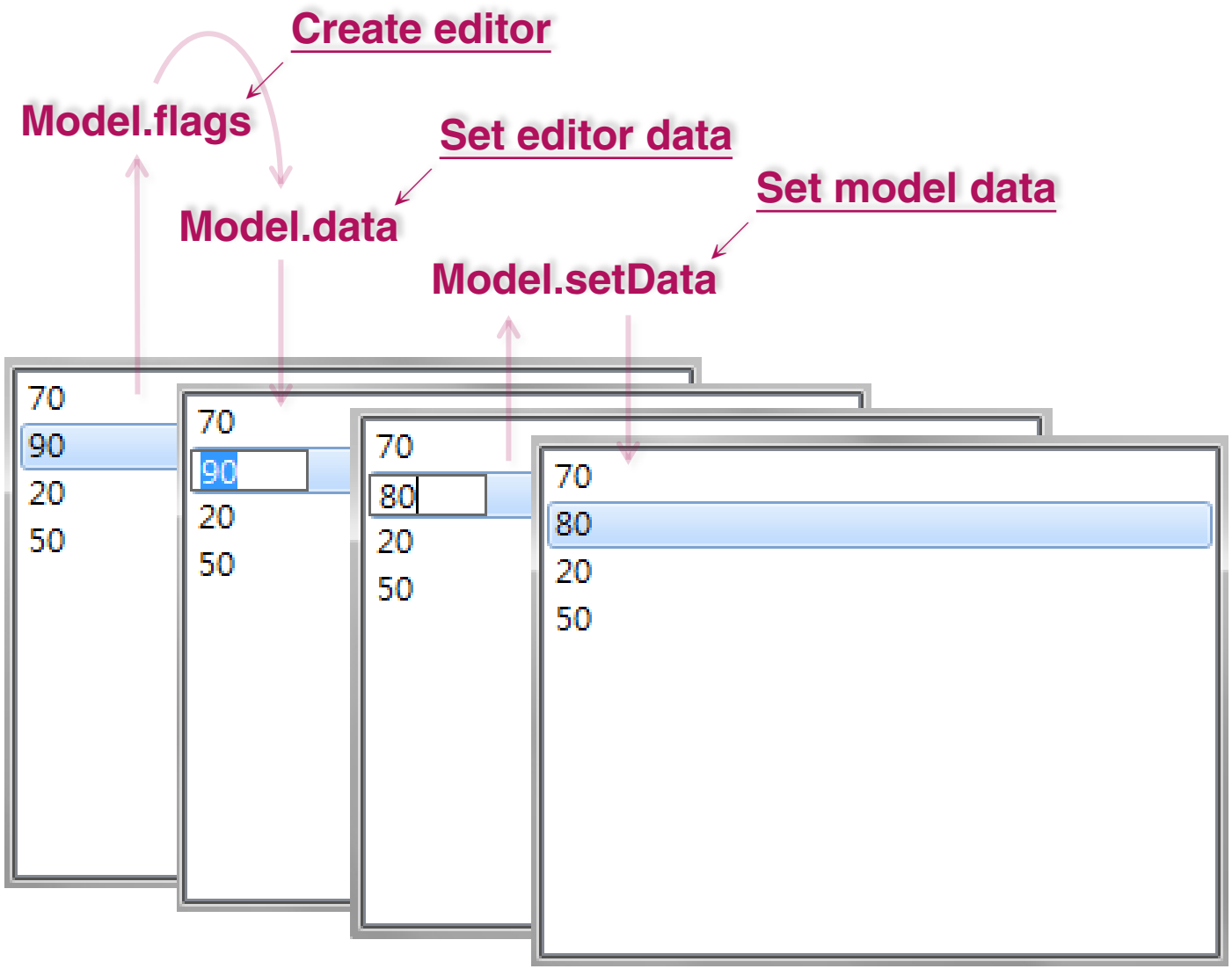
Edit Data with Delegate



Recall : Default Editor

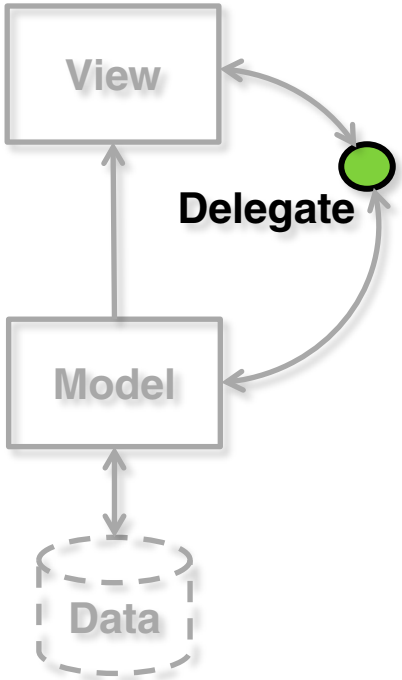


Recall : Default Editor



Build Edit Delegate

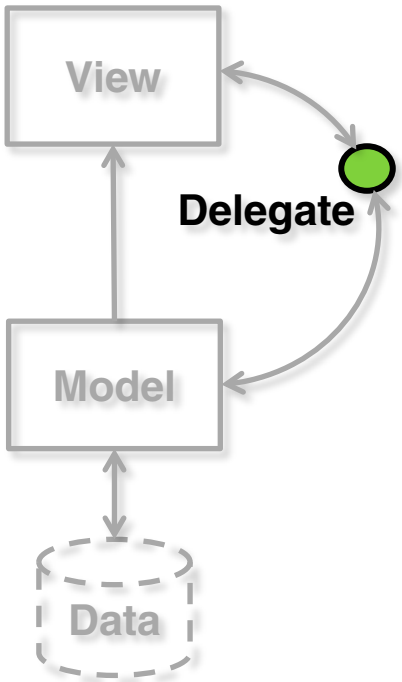
```
class MyEditDelegate(QStyledItemDelegate):
    def createEditor(self, parent, option, index):




    def setEditorData(self, editor, index):




    def setModelData(self, editor, model, index):
```



Build Edit Delegate ~ createEditor

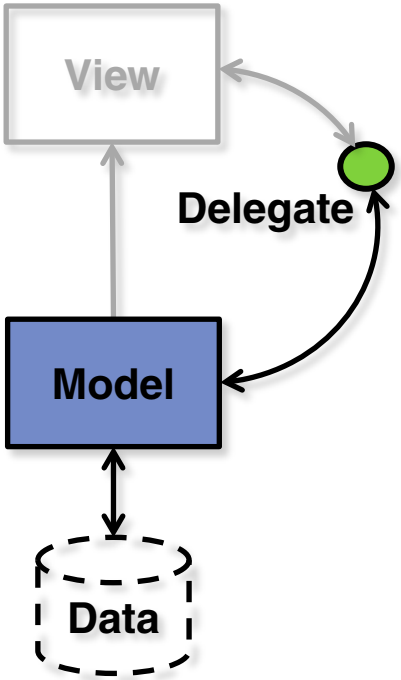
```
class MyEditDelegate(QStyledItemDelegate):
    def createEditor(self, parent, option, index):
        sbox = QSpinBox(parent)
        sbox.setRange(0, 100)
        return sbox

    def setEditorData(self, editor, index):



    def setModelData(self, editor, model, index):
```

← **Create editor ... and return itself**



Build Edit Delegate ~ setEditorData

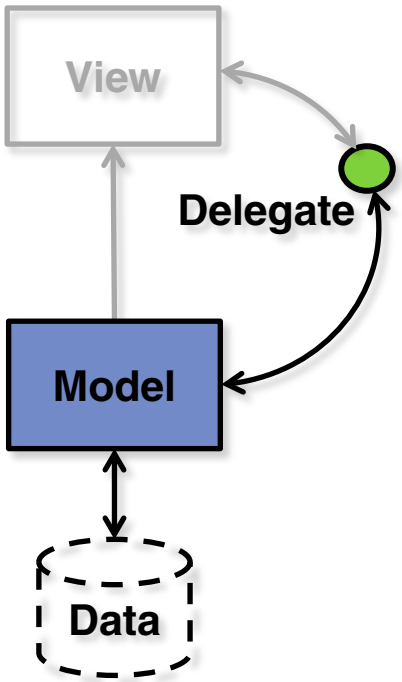
```
class MyEditDelegate(QStyledItemDelegate):
    def createEditor(self, parent, option, index):
        sbox = QSpinBox(parent)
        sbox.setRange(0, 100)
        return sbox

    def setEditorData(self, editor, index):
        item_var = index.data(Qt.DisplayRole)
        item_str = item_var.toPyObject()
        item_int = int(item_str)
        editor.setValue(item_int)    ← Set editor data

    def setModelData(self, editor, model, index):
```



Build Edit Delegate ~ setModelData

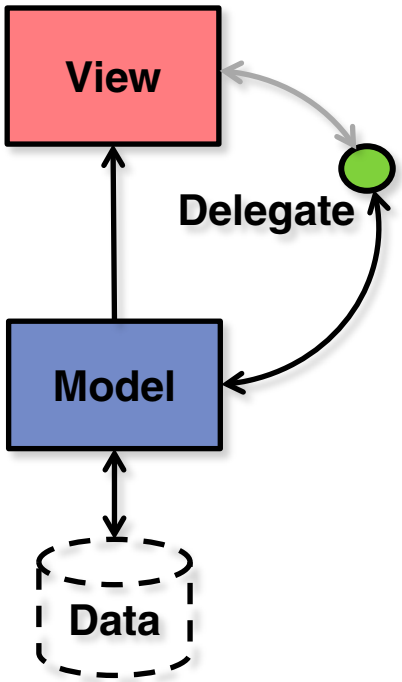
```
class MyEditDelegate(QStyledItemDelegate):
    def createEditor(self, parent, option, index):
        sbox = QSpinBox(parent)
        sbox.setRange(0, 100)
        return sbox

    def setEditorData(self, editor, index):
        item_var = index.data(Qt.DisplayRole)
        item_str = item_var.toPyObject()
        item_int = int(item_str)
        editor.setValue(item_int)

    def setModelData(self, editor, model, index):
        data_int = editor.value()
        data_var = QVariant(data_int)
        model.setData(index, data_var)
```

Set model data

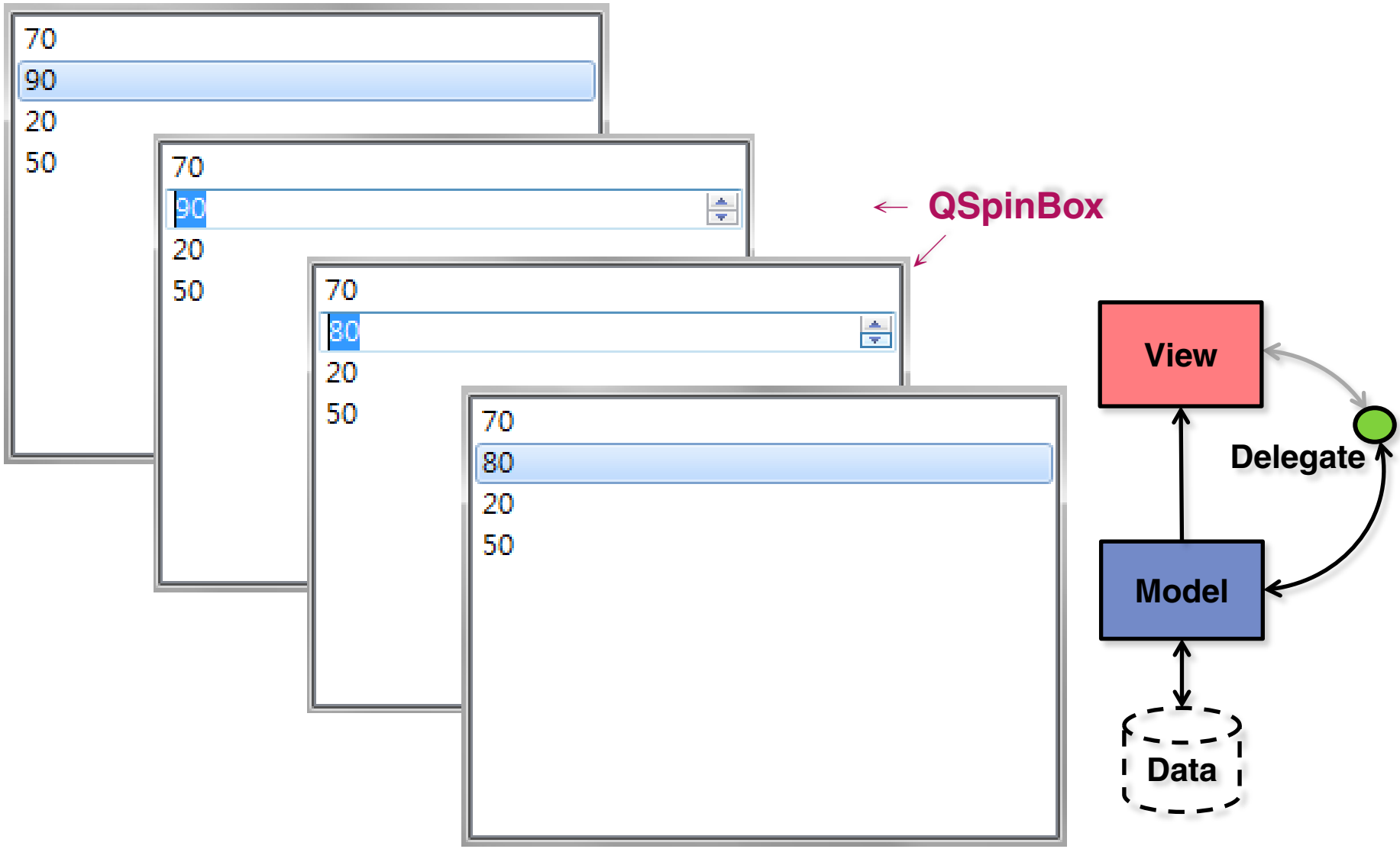


Set Delegate into View

```
app = QApplication(sys.argv)

model = MyListModel()
delegate = MyEditDelegate()

view = QListView()
view.setModel(model)
view.setItemDelegate(delegate)
view.show()

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



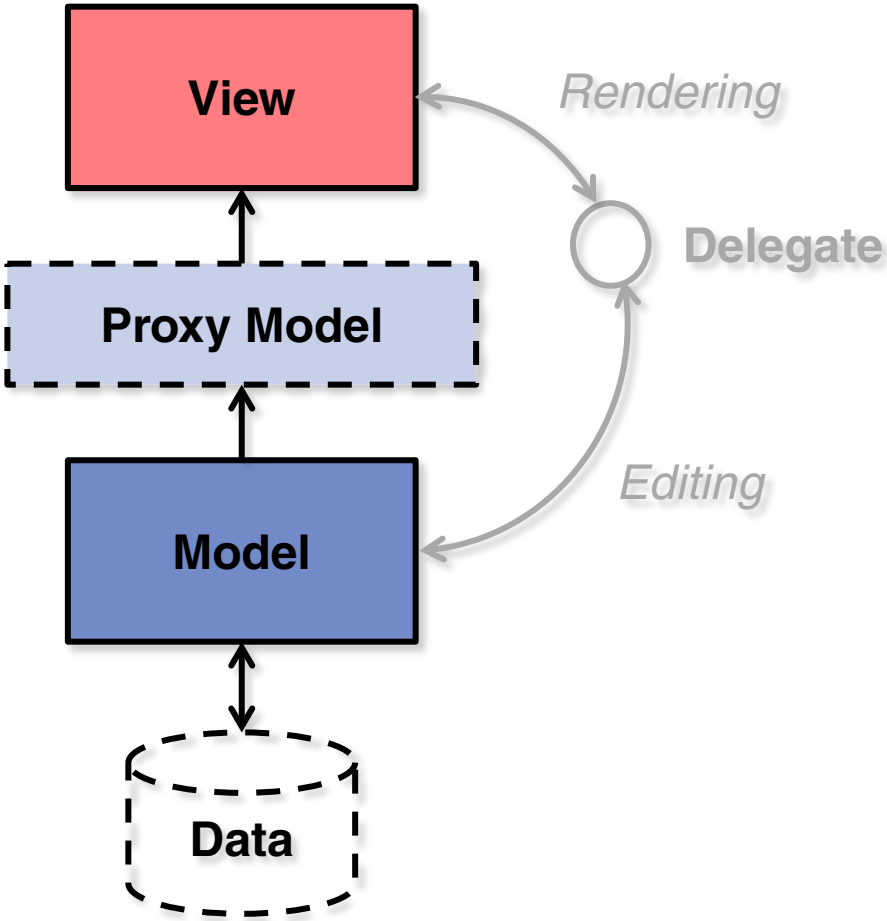
Edit Data with Delegate



Sort ?
Filter ?

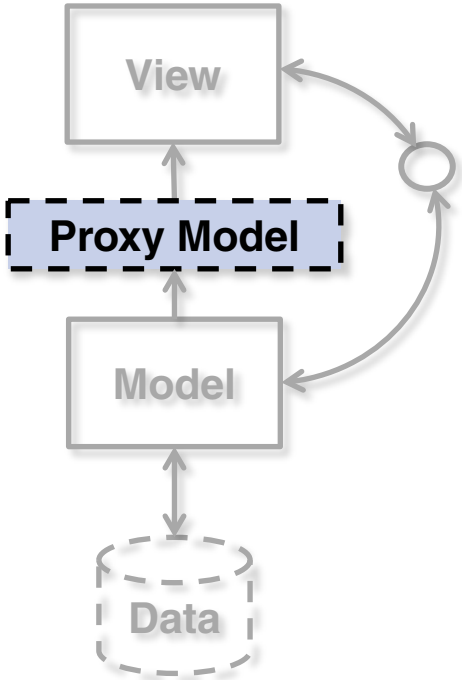
Use

QSortFilterProxyModel



Sort Data ~ QSortFilterProxyModel

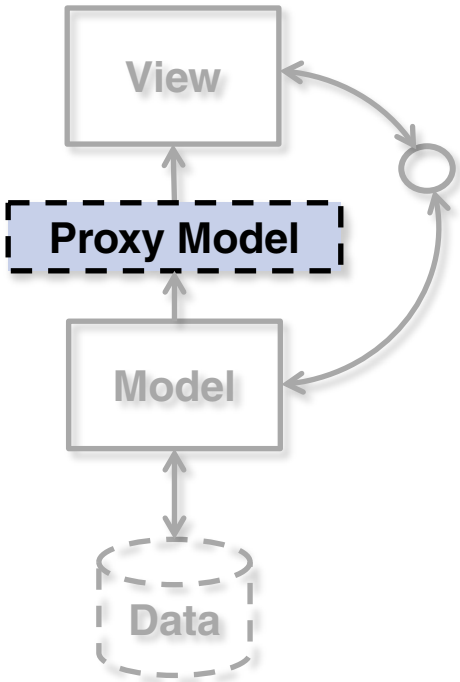
```
class SortProxyModel(QSortFilterProxyModel):
    def lessThan(self, left_index, right_index):
```



Sort Data ~ QSortFilterProxyModel

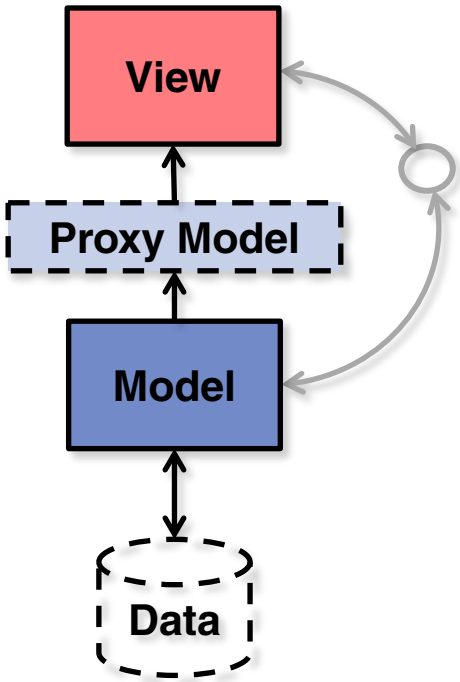
```
class SortProxyModel(QSortFilterProxyModel):
    def lessThan(self, left_index, right_index):
        left_var = left_index.data(Qt.DisplayRole)
        right_var = right_index.data(Qt.DisplayRole)

        left_str = left_var.toPyObject()
        right_str = right_var.toPyObject()

        left_int = int(left_str)
        right_int = int(right_str)

        return (left_int < right_int)
```

return True or False

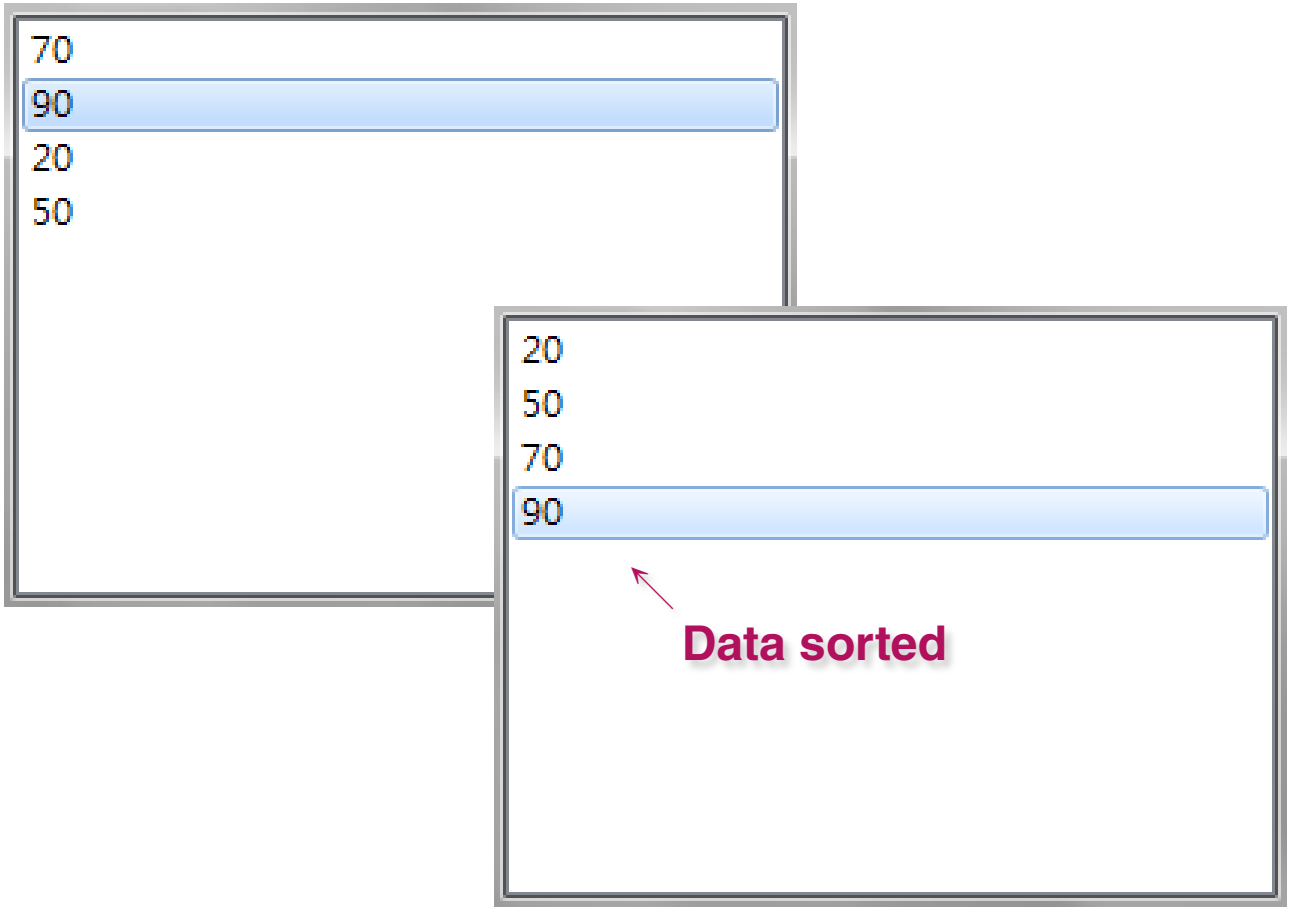


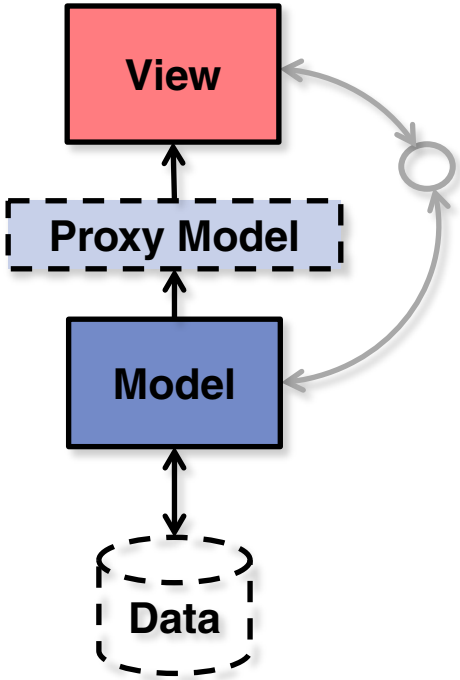
Apply SortProxyModel to Model and View

```
app = QApplication(sys.argv)

model = MyListModel()

proxy = SortProxyModel()
proxy.setSourceModel(model)
proxy.sort(0) ← Sort data by column 0

view = QListView()
view.setModel(proxy)
view.show()

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



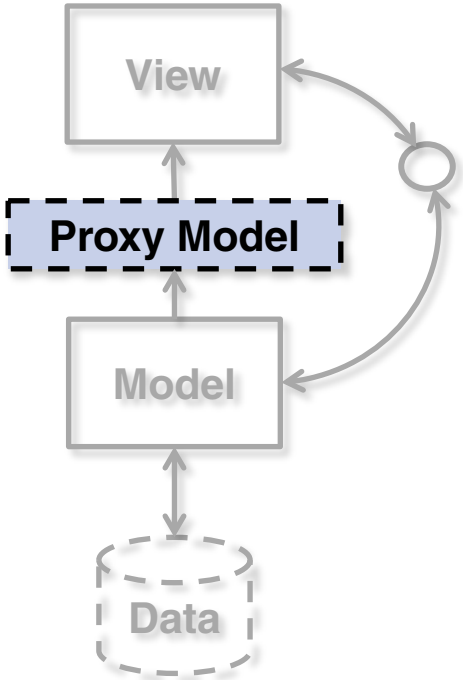
Sort Data





Filter Data ~ QSortFilterProxyModel

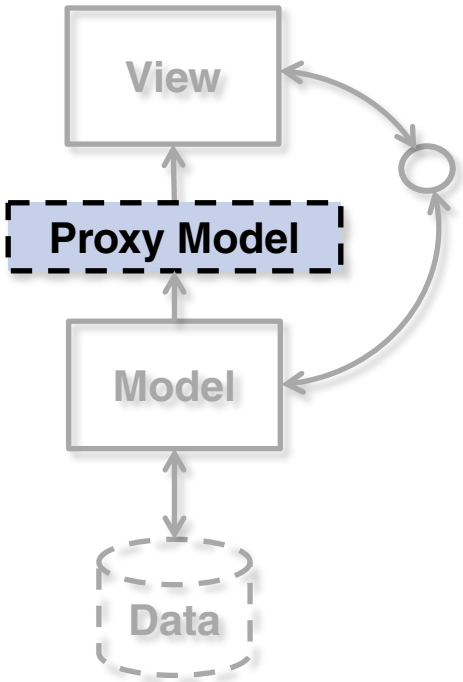
```
class FilterProxyModel(QSortFilterProxyModel):
    def filterAcceptsRow(self, src_row, src_parent):
```



Filter Data ~ QSortFilterProxyModel

```
class FilterProxyModel(QSortFilterProxyModel):
    def filterAcceptsRow(self, src_row, src_parent):
        src_model = self.sourceModel()
        src_index = src_model.index(src_row, 0)

        item_var = src_index.data(Qt.DisplayRole)
        item_int = int(item_var.toPyObject())

        return (item_int >= 60)
```

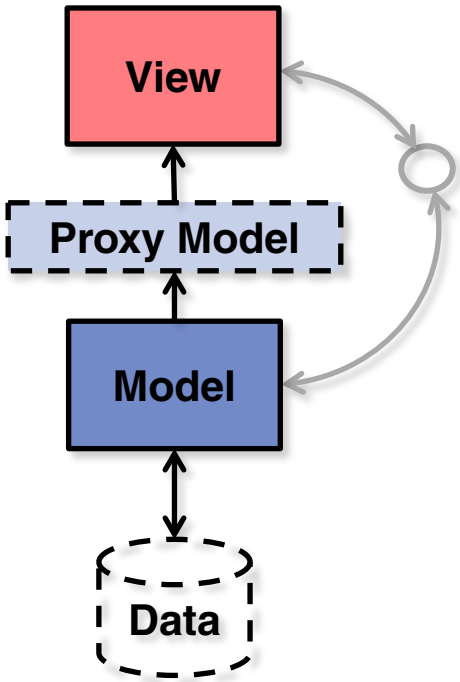
return True or False



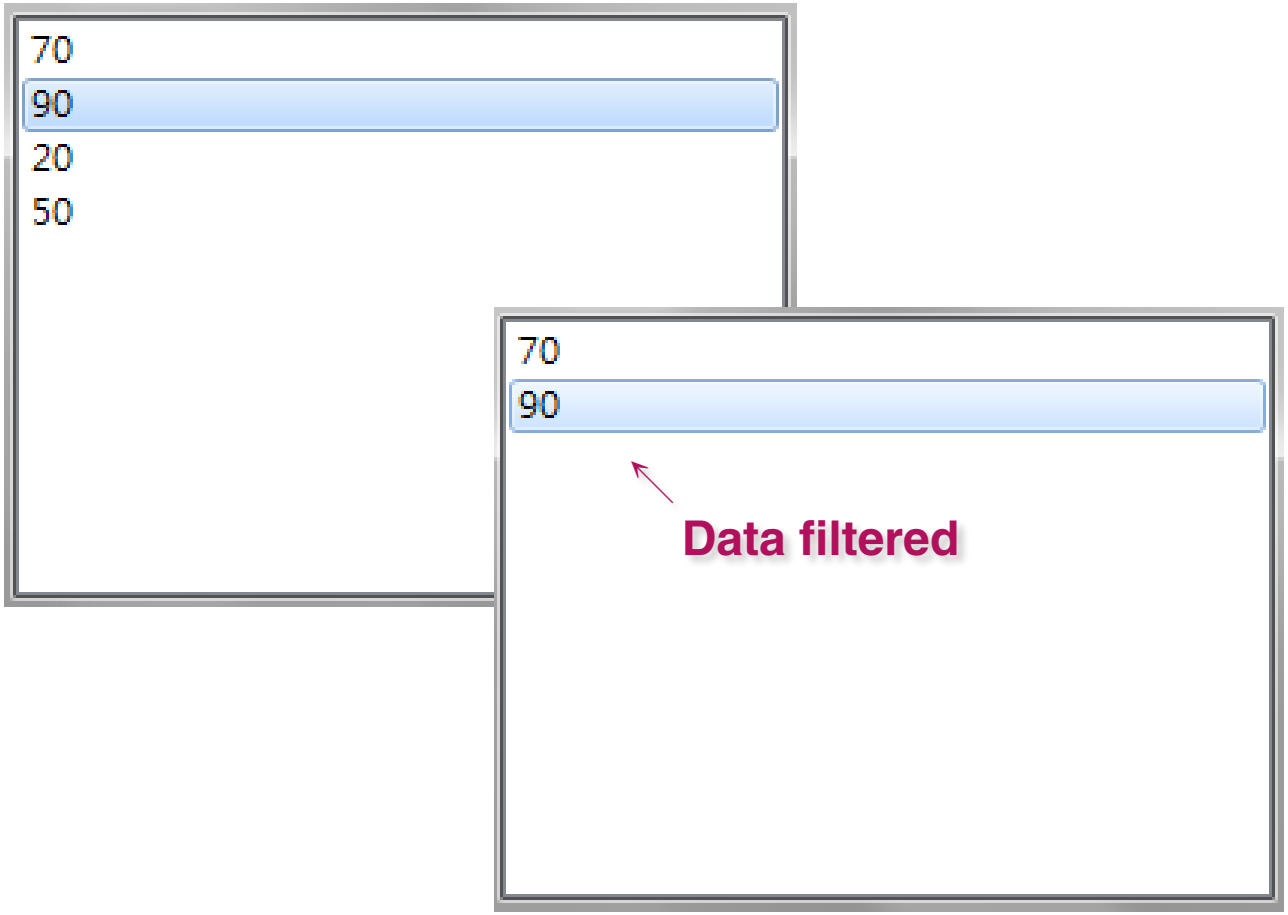
Apply FilterProxyModel

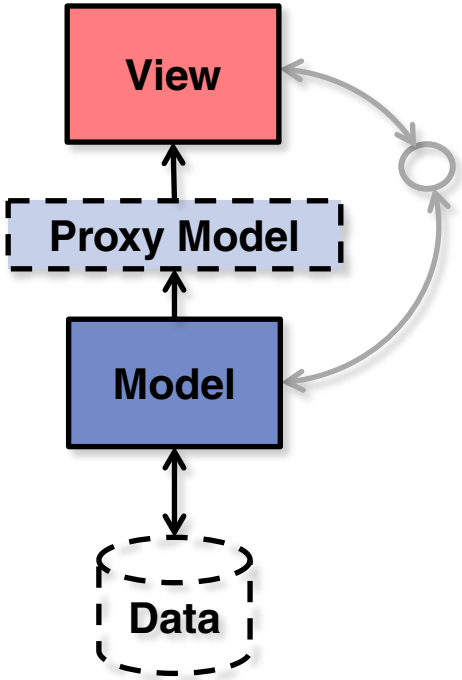
```
app = QApplication(sys.argv)

model = MyListModel()

proxy = FilterProxyModel()
proxy.setSourceModel(model)
proxy.setDynamicSortFilter(True)

view = QListView()
view.setModel(proxy)
view.show()

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

If True,
data will re-filter when original model is changed



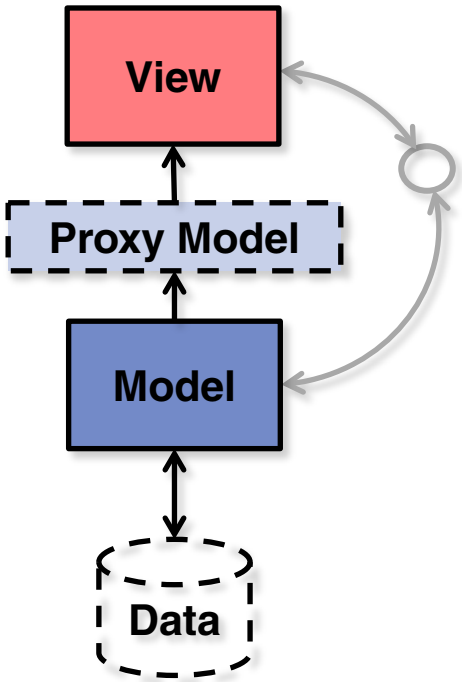
Filter Data





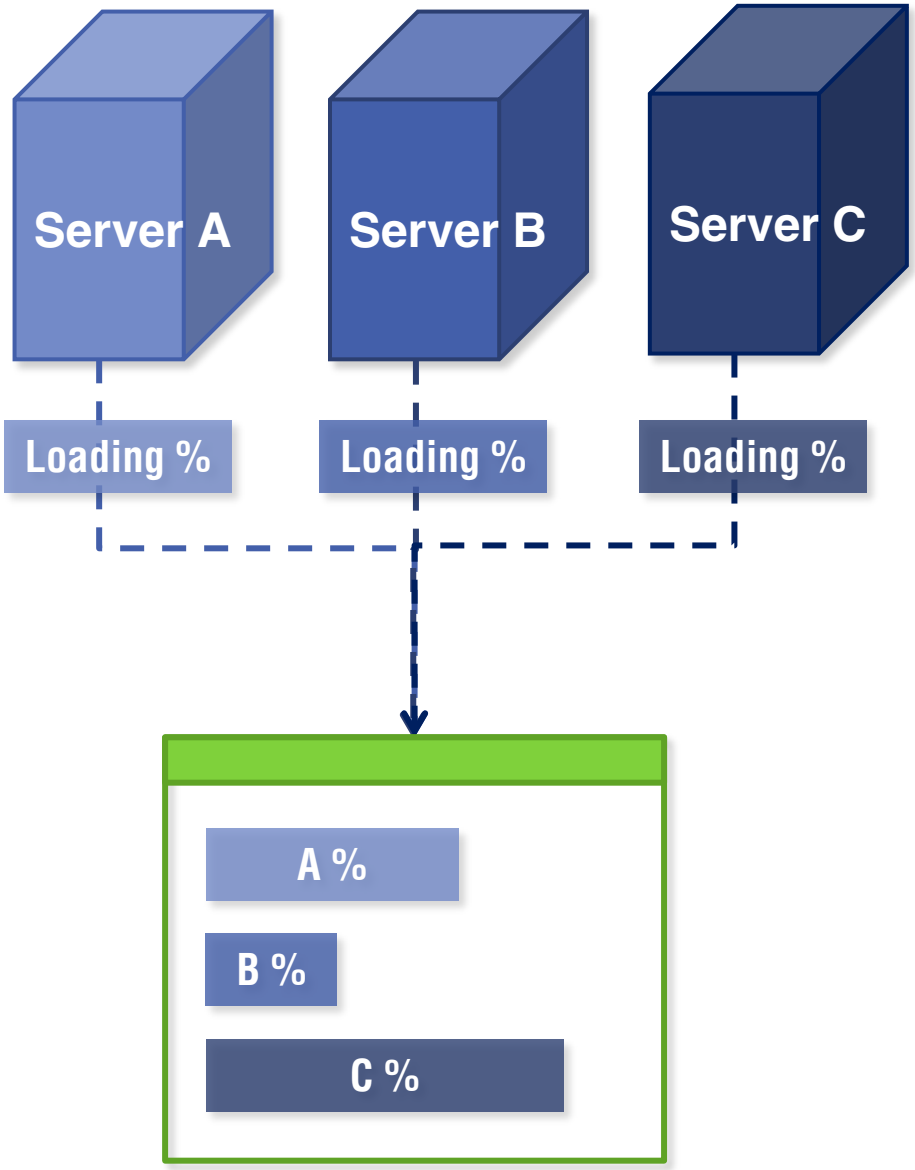
Sort & Filter ~ Recap

```
class ProxyModel(QSortFilterProxyModel):

    def lessThan(self, left_index, right_index)

    def filterAcceptsRow(self, src_row, src_parent)
```



Live DEMO

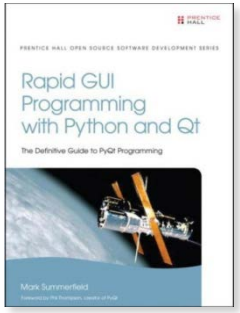
Server Loading Monitor



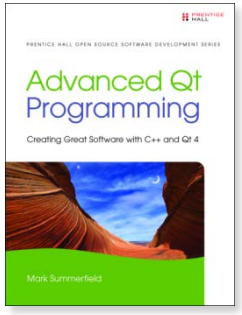
References

- **PyQt (Riverbank)** http://www.riverbankcomputing.co.uk/
- **Qt** http://qt.nokia.com/
- **PySide** http://www.pyside.org/

- **Rapid GUI Programming with Python and Qt**
  - by Mark Summerfield
  - ISBN: 978-0132354189

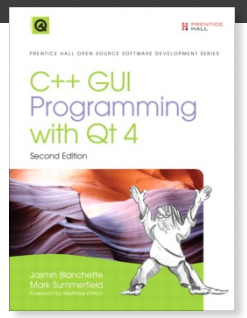


- **Advanced Qt Programming**
  - by Mark Summerfield
  - ISBN: 978-0321635907



References

- **C++ GUI Programming with Qt 4**
  - by Jasmin Blanchette and Mark Summerfield
  - ISBN: 978-0132354165



Q & A

Contacts

陳俊嘉
a.k.a CCC

- PTT cccx
- Plurk ccc_
- Facebook ccc.larc
- Google ccc.larc
- Email ccc.larc@gmail.com