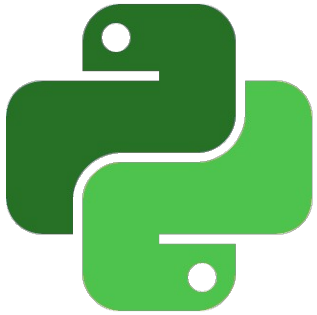


Modules Python

10-Interfaces graphiques : QT

- Introduction.
- Les bases.
- Menus & Barres d'outils.
- Mise en page.
- Gestion des événements.
- Boites de dialogue standards.
- Widgets.
- Drag & Drop.
- Dessin
- Personnalisation & Divers.

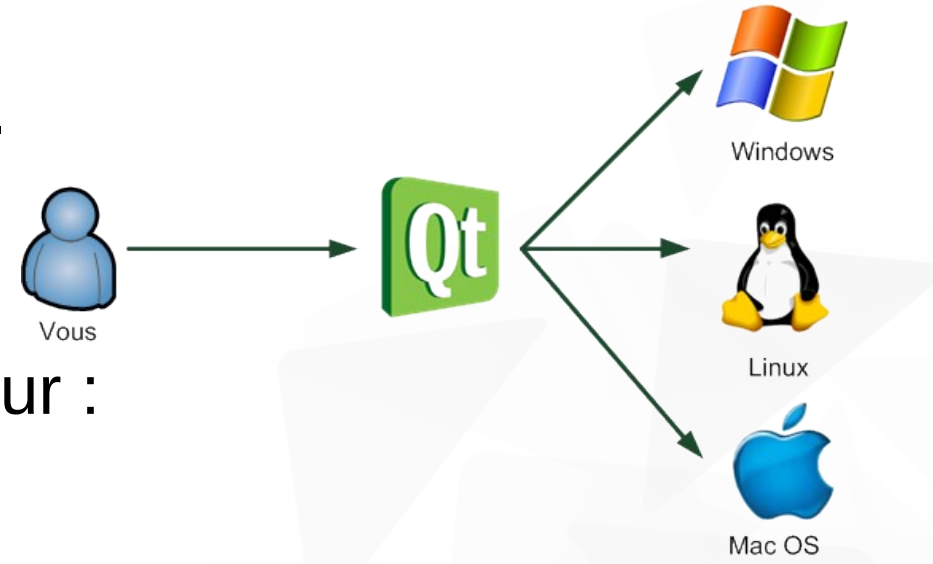
Simon CHOLLET - simon.chollet@ens.fr

Introduction

- Généralités.
- Modules internes.

Généralités

- **Qt** : **Framework** multi-plateformes **orienté objet** (C++).
- Offre des **composants** pas seulement graphiques :
  - Interface graphique (widgets).
  - Accès aux données.
  - Connexions réseaux.
  - Gestion des fils d'exécution.
  - Analyse XML.
  - Etc.
- **Portabilité** des applications sur :
  - Les « Unix Like ».
  - Windows et Mac OS X.



Modules internes

- **QtCore** : Fonctionnalités **non graphiques** → fichiers, types, threads, etc.
- **QtGUI** : Composants **graphiques** → boutons, fenêtres, barre d'outils, couleurs, etc.
- **QtNetwork** : Programmation autour du **réseau**.
- **QtXml** : Gestion fichiers **XML** avec SAX/DOM.
- **QtSvg** : Gestion fichiers **SVG**.
- **QtOpenGL** : Gestion affichage **2D/3D**.
- **QtSql** : Gestion des **bases de données**.

Les bases

- Fenêtre simple.
- Icône pour une fenêtre.
- Tooltips.
- Bouton « Quitter ».
- Boites de dialogue standards.
- Centrage sur l'écran.

Fenêtre simple

- Application principale : **moteur**.

```
import PyQt4.QtGui as QGui
app = QGui.QApplication(sys.argv)
```

- **Création** de la fenêtre :

```
widget = QGui.QWidget()
widget.resize(250, 150)
widget.setWindowTitle('simple')
```

- **Affichage** de la fenêtre :

```
widget.show()
```

- **Lancement** du moteur.

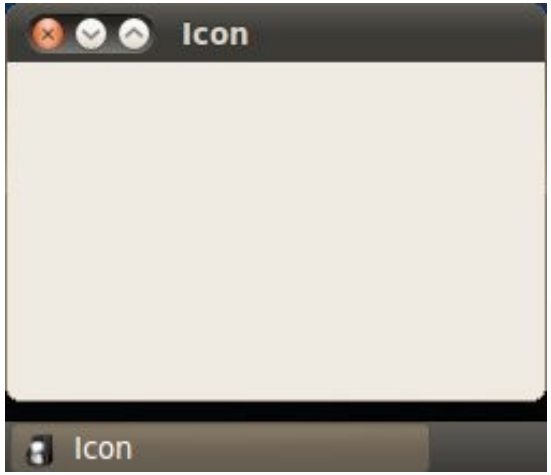
```
res = app.exec_()
# Quitte le programme
sys.exit(res)
```

code_qt_01.py

Icône pour fenêtre

- Création d'une **classe personnalisée**.

```
class IconWin(QGui.QWidget):
    """ Classe permettant de definir une fenetre
avec
    icone associe """
    def __init__(self, parent=None):
        """ Constructeur """
        # Appel du constructeur de la classe mere
        QGui.QWidget.__init__(self, parent)
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Icon')
        # Reglage de l'icone de la fenetre
        self.setWindowIcon(QGui.QIcon('web.png'))
```



- **Instance** et **affichage** de la nouvelle fenêtre.

```
# Creation de la fenetre avec icône
icon_win = IconWin()

# Affichage de la fenetre
icon_win.show()
```

code_qt_02.py

Tooltips sur fenêtre

- Création d'une **classe personnalisée**.

```
class TooltipWin(QGui.QWidget):
    """ Classe permettant de definir une fenetre
avec
    affichage de Tooltips """
    def __init__(self, parent=None):
        # Appel du constructeur de la classe mere
        QGui.QWidget.__init__(self, parent)
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Tooltip')
        # Reglage du controle de Tooltip
        self.setToolTip(
            'This is a <b>QWidget</b> widget')
        # Reglage de la police de caracteres
        QGui.QToolTip.setFont(
            QGui.QFont('OldEnglish', 10))
```

- **Instance** et **affichage** de la nouvelle fenêtre.

```
tooltip_win = TooltipWin()
tooltip_win.show()
```

code_qt_03.py

Bouton « Quitter »

- Création d'une **classe personnalisée**.

```
class QuitButtonWin(QtGui.QWidget):
    """ Classe permettant de definir une fenetre
    avec bouton quitter ""«
    def __init__(self, parent=None):
        ...
        # Creation du bouton "Close"
        quit = QtGui.QPushButton('Close', self)
        # Reglage des dimensions du bouton "Close"
        quit.setGeometry(10, 10, 60, 35)

        # Association de la commande de fermeture
        # de fenetre au bouton
        self.connect(quit,
QtCore.SIGNAL('clicked()'),
            QtGui.qApp, QtCore.SLOT('quit()'))
```

- **Instance** et **affichage** de la nouvelle fenêtre.

```
qb_win = QuitButtonWin()
qb_win.show()
```

code_qt_04.py

Boite de dialogue

- Ajout méthode appelée sur **fermeture**.
- Utilisation du widget `MessageBox()`.
- **Accepte** ou **ignore** l'événement.

```
class MessageBox(QtGui.QWidget):
    ...
    def closeEvent(self, event):
        """ Methode appelee sur fermeture de la
fenetre """«
        # Creation boite de de confirmation
        reply = QtGui.QMessageBox.question(
            self, # Fenetre parente
            'Message', # Titre de la fenetre
            "Are you sure to quit?", # Message a
afficher
            QtGui.QMessageBox.Yes |
            QtGui.QMessageBox.No, # Boutons a
incorporer
            QtGui.QMessageBox.No) # Selectionne par
defaut
        # Selon la reponse
        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()
```

code_qt_05.py

Centrer la fenêtre

- Ajout méthode de **centrage**.
- Utilisation des **propriétés** espace affichage.
- **Bouge** la fenêtre au centre.

```
class CenterWin(QtGui.QWidget):
    ...
    def __init__(self, parent=None):
        ...
        # Mise au centre de la fenetre,
        # appel de la methode de centrage
        self.center()

    def center(self):
        # Recuperation de la taille de l'affichage
(ecran)
        screen =
QtGui.QDesktopWidget().screenGeometry()
        # Recuperation de la taille de la fenetre
        size =  self.geometry()
        # Change la position de la fenetre
        self.move((screen.width() - size.width()) / 2,
                  (screen.height() - size.height()) /
2)
```

code_qt_06.py

Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

Menus & Barres d'outils

- Ajouter une barre d’état.
- Ajouter un menu.
- Ajouter une barre d’outils.
- Ajouter une zone d’édition.

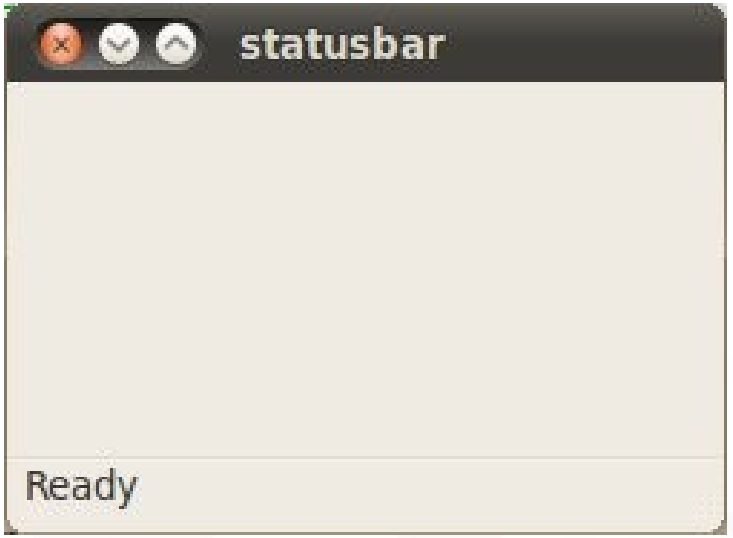
Barre d'état

- **Spécifique** à la classe **`CMainWindow`**.
- Barre d'état : **`statusBar()`**.
- Message à ajouter : **`showMessage()`**.

```
class MainWindow(QtGui.QMainWindow):
    ...
    def __init__(self):
        # Appel du constructeur de la classe mere
        QtGui.QMainWindow.__init__(self)

        # Reglage du titre de la fenetre
        self.setWindowTitle('statusbar')
        # Reglage des dimensions de fenetre
        self.resize(250, 150)

        # Ajout d'une barre de statut a la fenetre
        self.statusBar().showMessage('Ready')
```

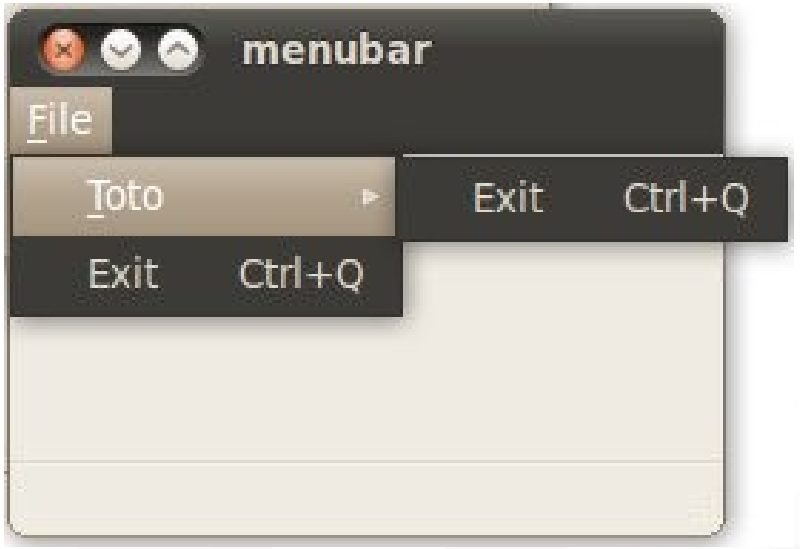


`code_qt_07.py`

Menu

- **Création** : action, raccourcis, message d'état.
- **Ajout** du menu, entrée, sous-menu.

```
class MainWindow(QtGui.QMainWindow):
    def __init__(self):
        ...
        # Creation de l'action du sous-menu
        exit = QtGui.QAction(QtGui.QIcon('exit.png'),
            'Exit', self)
        # Creation du raccourcis clavier
        exit.setShortcut('Ctrl+Q')
        # Reglage du tooltip associe a la commande
        exit.setStatusTip('Exit application')
        # Reglage de l'action
        self.connect(exit,
                     QtCore.SIGNAL('triggered()'),
                     QtCore.SLOT('close()'))
        ...
        # Ajout de la barre de menu
        menubar = self.menuBar()
        # Ajout du point d'entree de la barre d'outils
        file = menubar.addMenu('&File')
        # Ajout du sous-menu
        file.addAction(exit)
```



code_qt_08.py

Barre d'outils

- **Création** : action, raccourcis, icône.
- **Ajout** de la barre avec bouton.

```
class MainWindow(QtGui.QMainWindow):
    def __init__(self):
        ...
        # Creation action avec icone pour commande
        self.exit = QtGui.QAction(
            QtGui.QIcon('exit.png'),
           'Exit', self)
        # Creation raccourcis clavier
        self.exit.setShortcut('Ctrl+Q')
        # Reglage action a executer
        self.connect(self.exit,
            QtCore.SIGNAL('triggered()'),
            QtCore.SLOT('close()'))

        # Creation de la barre d'outils
        self.toolbar = self.addToolBar('Exit')
        # Ajout de la commande a la barre d'outils
        self.toolbar.addAction(self.exit)
```

code_qt_09.py

Zone d'édition

- **Création** : zone édition, barre outils/statut.
- **Centrage** de la zone.

```
class MainWindow(QtGui.QMainWindow):
    def __init__(self):
        ...
        # Creation d'une zone d'edition
        textEdit = QtGui.QTextEdit()
        # Centrage de la zone de texte
        self.setCentralWidget(textEdit)

        # Creation d'une barre d'outils
        ...
        # Reglage du tooltip
        ...
        # Creation d'une barre de statut
        ...
        # Creation de la barre de menu
        ...
        # Ajout d'une barre d'outils
        ...
```

code_qt_10.py

Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

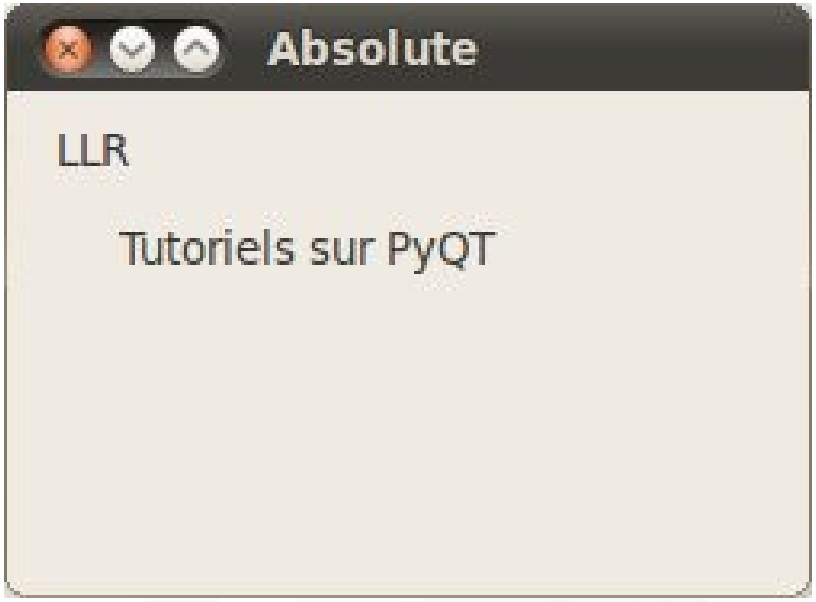
Mise en page

- Mise en page absolue.
- Arrangements vertical / horizontal.
- Arrangements en grille.

Zone de texte fixe

- Autre méthode, **appel parent** : **`super()`**.
- Méthode initialisation fenêtre : **`InitGUI()`**.
- **Création** / **positionnement** contrôle texte fixe.

```
class ExampleWin(QtGui.QWidget):
    def __init__(self):
        # Appel du constructeur de la classe mere
        super(ExampleWin, self).__init__()
        # Appel de la methode d'initialisation
        self.initGUI()
    def initGUI(self):
        # Creation d'une zone de texte fixe
        label1 = QtGui.QLabel('LLR', self)
        # Positionnement de la zone de texte fixe
        label1.move(15, 10)
        # Creation d'une zone de texte fixe
        label2 = QtGui.QLabel(
            'Tutoriels sur PyQT', self)
        # Positionnement de la zone de texte fixe
        label2.move(35, 40)
        ...
```



`code_qt_11.py`

Mise en place de boutons

- **Création** d'arrangements Vertical/Horizontal.
- **Combinaison** des arrangements.
- **Mise en place** boutons dans arrangements.
- Peut **modifier** taille de fenêtre : les boutons **suivent**.

```
class ExampleWin(QtGui.QWidget):
    ...
    def initGUI(self):
        ...
        # Creation d'un arrangement horizontal
        hbox = QtGui.QHBoxLayout()
        # Regle le facteur d'etirement
        hbox.addStretch(1)
        # Ajout des boutons
        hbox.addWidget(okButton)
        hbox.addWidget(cancelButton)
        # Creation d'un arrangement vertical
        vbox = QtGui.QVBoxLayout()
        vbox.addStretch(1)
        # Combinaison des arrangements
        vbox.addLayout(hbox)
        self.setLayout(vbox)
```

code_qt_12.py

Grille pour mise en page

- **Création** de la grille d'arrangement en forme de grille.
- **Tableau** pour la position des boutons dans la grille.
- **Mise en place** des boutons dans la grille.
- **Association** de la grille à la mise en page de fenêtre.

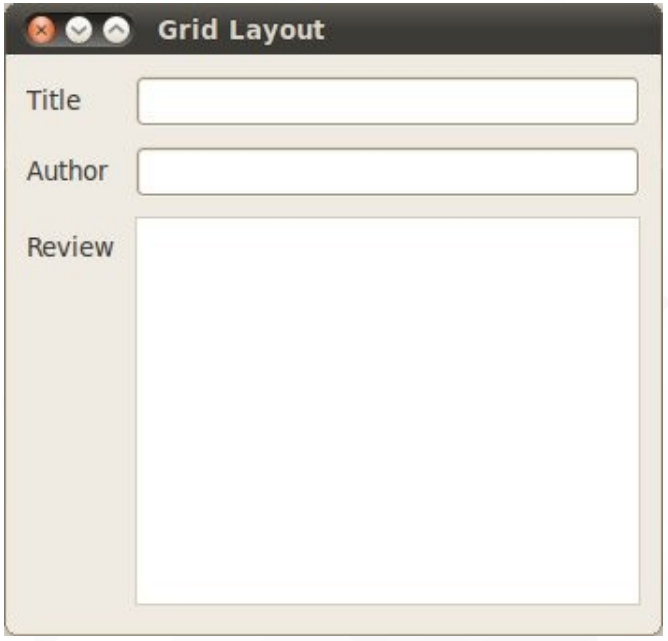
```
class ExampleWin(QtGui.QWidget):
        ...
        # Creation de la grille de repartition
        grid = QtGui.QGridLayout()
        ...
        # Position des boutons dans la grille
        pos = [(0, 0), (0, 1), ...]
        # Pour tous les boutons definis
        for i in names:
            button = QtGui.QPushButton(i)
            grid.addWidget(button,
                pos[j][0], pos[j][1])
        # Association de la grille de repartition
        self.setLayout(grid)
```

code_qt_13.py

Grille pour mise en page

- **Création** de la grille d'arrangement en forme de grille.
- **Création** contrôles à mettre dans la grille.
- **Mise en place** des contrôles dans la grille.
- **Association** de la grille à la mise en page de fenêtre.

```
class ExampleWin(QtGui.QWidget):
    def initGUI(self):
        # Creation des controles fixes d'intitules
        title  = QtGui.QLabel('Title')
        ...
        # Creation des controles de saisie de texte
        titleEdit  = QtGui.QLineEdit()
        ...
        # Creation de la grille de repartition
        grid = QtGui.QGridLayout()
        # Reglage des espaces entre controles
        grid.setSpacing(10)
        ...
        # Ajoute les controles du titre
        grid.addWidget(title, 1, 0)
        grid.addWidget(titleEdit, 1, 1)
        ...
```



code_qt_14.py

Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

Gestion des événements

- Changement d'état.
- Les touches clavier.
- Sur les boutons.
- La souris.

Association changement état

- **Création** d'un LCD.
- **Création** Slider.
- **Association** événements Slider / valeurs LCD.

```
class ExampleWin(QtGui.QWidget):
    def initGUI(self):
        # Creation d'un controle de type LCD
        lcd = QtGui.QLCDNumber(self)
        # Creation d'un slider
        slider = QtGui.QSlider(
            QtCore.Qt.Horizontal, self)
        vbox = QtGui.QVBoxLayout()
        # Ajoute les controles LCD et slider
        vbox.addWidget(lcd)
        vbox.addWidget(slider)
        # Associe le repartiteur
        self.setLayout(vbox)
        # Evennement slider a l'affichage LCD
        self.connect(slider,
            QtCore.SIGNAL('valueChanged(int)'),
            lcd, QtCore.SLOT('display(int)'))
```

code_qt_15.py

Touches Clavier

- **Création** d'une fenêtre simple.
- **Création** méthode appelée sur appui touche.
- **Vérification** type de touche : teste si **`ESC`**.
- Liste touches dans classe **`QtCore.Qt.Key`**.

```
class ExampleWin(QtGui.QWidget):
    def keyPressEvent(self, event):
        # Selon la touche pressee
        if event.key() == QtCore.Qt.Key_Escape:
            # Si touche 'Esc', on quitte
            self.close()
```

`code_qt_16.py`

Evénements sur boutons

- **Création** des boutons.
- **Association** événements sur boutons.
- **Création** méthode gestion événements bouton.

```
class ExampleWin(QtGui.QMainWindow):
    def initGUI(self):
        # Creation boutons
        button1 = QtGui.QPushButton(
            "Button 1", self)
        ...
        # Reglage de la methode a appeler
        self.connect(
            button1, QtCore.SIGNAL('clicked()'),
            self.buttonClicked)
        ...
    def buttonClicked(self):
        # Reglage d'un envoyeur d'evenements
        sender = self.sender()
        # Reglage de la barre de statut
        self.statusBar().showMessage(
            sender.text() + ' was pressed')
```

code_qt_17.py

Evénements Souris

- **Création** d'une fenêtre simple.
- Création **méthode** appelée sur événement souris.
- Envoi le **signal** de fermeture de l'application.

```
class ExampleWin(QtGui.QWidget):
    def __init__(self):
        # Regle le signal personnel
        self.connect(
            self, QtCore.SIGNAL('closeEmitApp()'),
            QtCore.SLOT('close()'))
    def mousePressEvent(self, event):
        # Envoi le signal de demande de fermeture
        self.emit(QtCore.SIGNAL('closeEmitApp()'))
```

code_qt_18.py

Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

Boites de dialogue standard

- Boite de saisie de valeurs.
- Boite de choix de couleur dans une palette.
- Boite de choix de polices de caractères.
- Boite de sélection de fichiers / répertoire.

Boite de saisie de valeur

- **Création** d'une fenêtre avec zone de texte.
- **Boite de dialogue** avec demande d'informations.
- Mise à jour fenêtre **parente**.

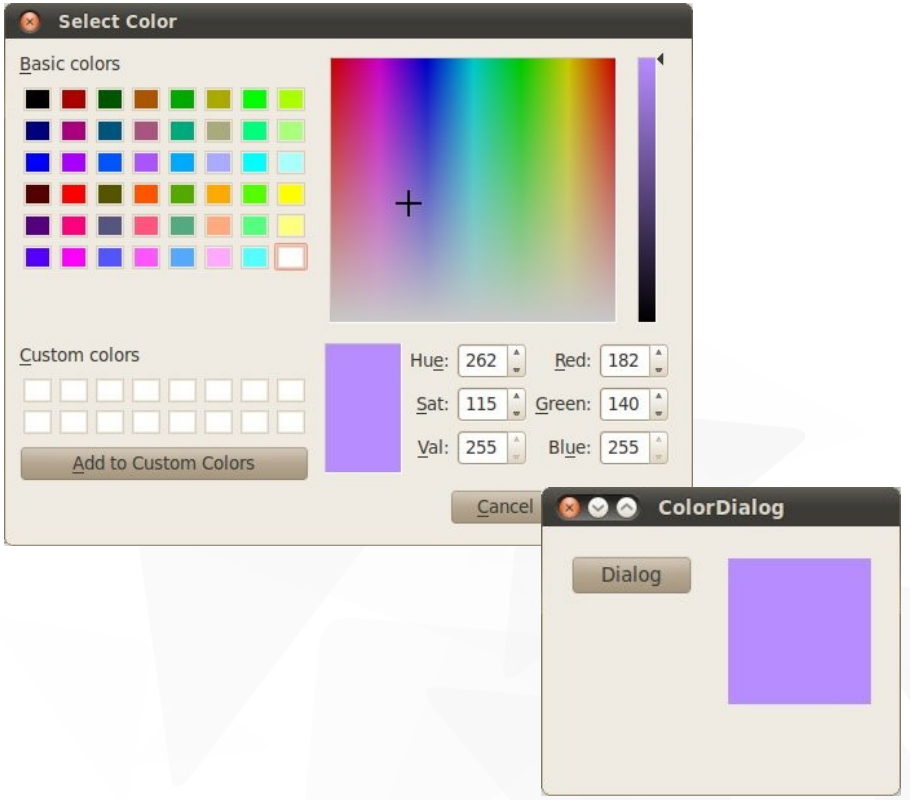
```
class InputDialog(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation d'un bouton d'interrogation
        self.button = QtGui.QPushButton(
            'Dialog', self)
        # Associe l'action d'ouverture au bouton
        self.connect(self.button,
            QtCore.SIGNAL('clicked()'),
            self.showDialog)
        # Ajoute une zone de texte de saisie
        self.label = QtGui.QLineEdit(self)
    def ShowDialog(self):
        # Affichage de la boite de dialogue
        text, ok = QtGui.QInputDialog.getText(
            self, 'Input Dialog','Enter your name:')
        # Si validation de la reponse
        if ok:
            self.label.setText(str(text))
```

code_qt_19.py

Choix de couleurs

- **Création** d'une fenêtre avec zone de couleur.
- **Boite de dialogue** avec palette de couleur.
- Mise à jour fenêtre **parente**.

```
class ColorDialog(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation d'une couleur, ici noir
        color = QtGui.QColor(0, 0, 0)
        ...
        # Creation d'une zone coloree
        self.widget = QtGui.QWidget(self)
        # Reglage de la couleur de la zone coloree
        self.widget.setStyleSheet(
            "QWidget{background-color: %s}" %
            color.name())
    def showDialog(self):
        # Ouverture de la boite choix de couleurs
        col = QtGui.QColorDialog.getColor()
        # Si une couleur a ete choisie
        if col.isValid():
            self.widget.setStyleSheet(
                "Qwidget{background-color: %s}" %
                col.name())
```



code_qt_20.py

Polices de caractères

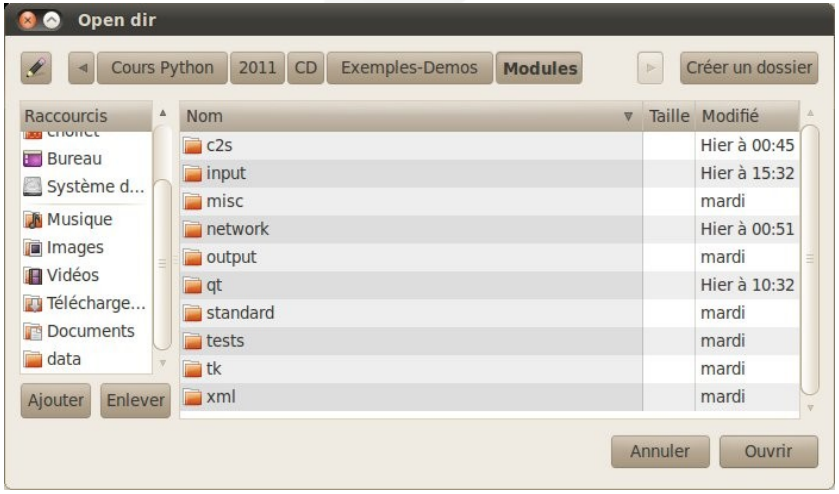
- **Création** d'une fenêtre avec zone de texte.
- **Boite de dialogue** avec polices de caractères.
- Mise à jour fenêtre **parente**.

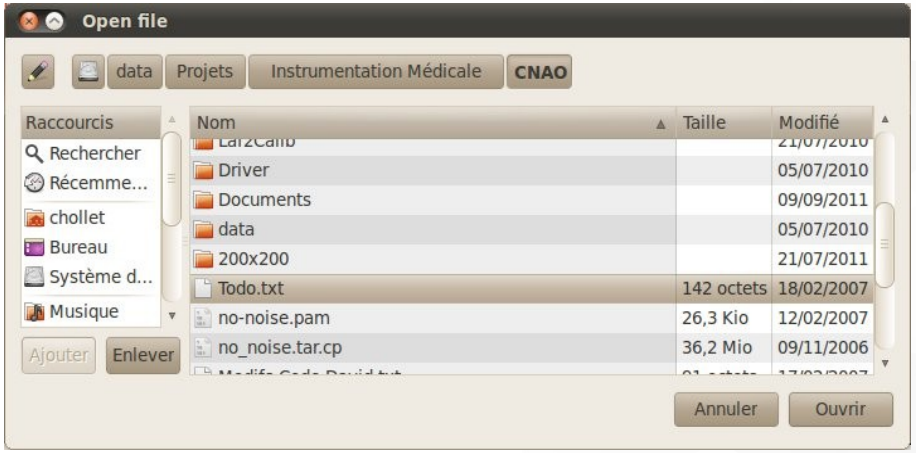
```
class FontDialog(QtGui.QWidget):
    def __init__(self, parent=None):
        ...
        # Zone de texte fixe
        self.label = QtGui.QLabel(
            'Test Polices', self)
        ...
    def ShowDialog(self):
        # Ouverture de la boite de choix de polices
        font, ok = QtGui.QFontDialog.getFont()
        # Si un choix a ete fait
        if ok:
            # Reglage de la police au texte fixe
            self.label.setFont(font)
```

code_qt_21.py

Sélecteur de fichiers / répertoires

- **Création** d'une fenêtre avec zone de texte.
- Création d'un **menu** avec raccourcis clavier.
- Boite de dialogue avec sélection :
  - Fichiers : **`QtGui.QFileDialog.getOpenFileName()`**.
  - Répertoire : **`QtGui.QFileDialog.getExistingDirectory()`**.
- Contenu fichier / répertoire dans zone de texte.





`code_qt_22.py`

Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

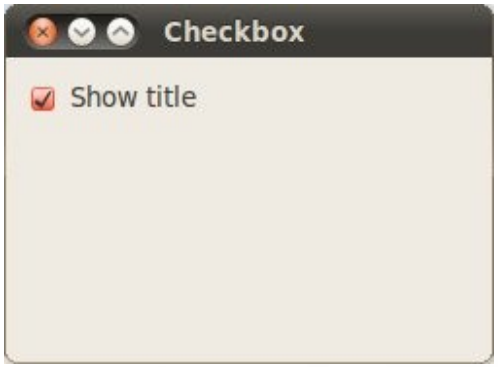
Widgets

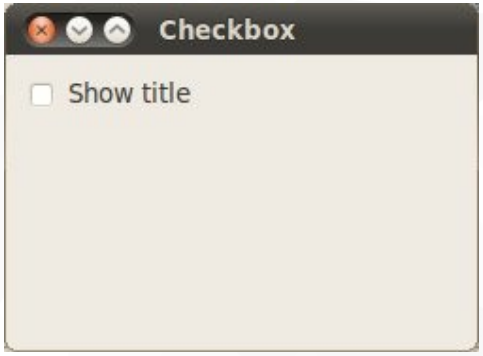
- Case à cocher.
- Bouton à 2 états.
- Curseur sur échelle.
- Barre progression.
- Calendrier.
- Afficheur d'image.
- Zone d'édition.
- Séparateur zones.
- Sélections.

Case à cocher

- **Création** d'une fenêtre avec case à cocher.
- Association d'une **méthode** au changement.
- Mise à jour fenêtre **parente**.

```
class CheckBoxWin(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation d'une CheckBox
        self.cb = QtGui.QCheckBox(
            'Show title', self)
        # Reglage de la propriete de basculement
        self.cb.toggle()
        # Associe l'action de cocher la case
        self.connect(self.cb,
            QtCore.SIGNAL('stateChanged(int)'),
            self.changeTitle)
    def changeTitle(self, value):
        # Selon l'etat de la case a cocher
        if self.cb.isChecked():
            self.setWindowTitle('Checkbox')
        else:
            self.setWindowTitle('')
```





code_qt_23.py

Boutons à 2 états

- **Création** d'une fenêtre avec 3 boutons.
- Association d'une **méthode** au changement.
- Mise à jour fenêtre **parente**.

```
class ToggleButtonWin(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation d'une couleur
        self.color = QtGui.QColor(0, 0, 0)
        # Creation du bouton composante rouge
        self.red = QtGui.QPushButton('Red', self)
        # Reglage toggle du bouton
        self.red.setCheckable(True)
        # Associe une methode a l'appui du bouton
        self.connect(self.red,
            QtCore.SIGNAL('clicked()'), self.setRed)
        ...
        # Creation d'une zone de couleur
        self.square = QtGui.QWidget(self)
    def setRed(self):
        ...
        self.square.setStyleSheet("QWidget
{background-color: %s}" % self.color.name())
```

code_qt_24.py

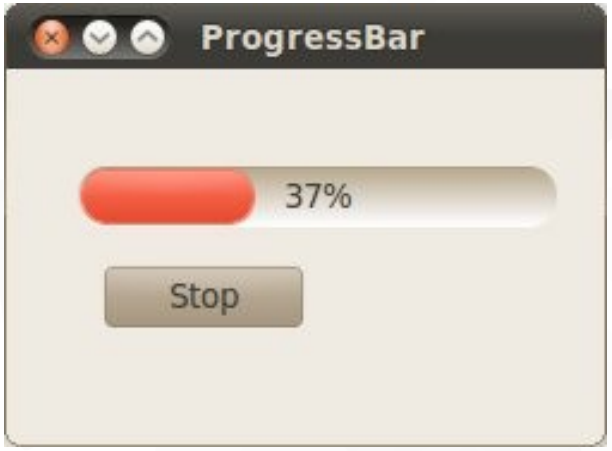
Curseur sur échelle

- **Création** d'une fenêtre avec un « **`slider`** ».
  - Méthode : **`QtGui.Qslider()`**.
- Association **méthode** au changement valeur du « **`slider`** ».
- **Chargement** d'une image dans un « **`label`** ».
- **Réglage** de l'icône selon la valeur.

`code_qt_25.py`

Barre de progression

- **Création** d'une fenêtre avec une barre de progression : `QtGui.QProgressBar()`.
- **Ajout** d'un bouton Start / Stop.
- Création d'un **timer**.
- **Méthode** appelée par timer : mise à jour.
- Sur Start : démarrage du timer, appelé toutes les **100 ms**.
- A la fin, **remise à zéro**.



`code_qt_26.py`

Calendrier

- **Création** d'une fenêtre avec un calendrier.
- Association d'une **méthode** au changement de date.
- Mise à jour zone texte selon la **date** choisie.

```
class CalendarWin(QtGui.QWidget):
    def __init__(self, parent=None):
        ...
        # Creation du Widget Calendrier
        self.cal = QtGui.QCalendarWidget(self)
        # Affichage de la grille dans le Calendrier
        self.cal.setGridVisible(True)
        # Reglage des dimensions de case de Calendrier
        self.cal.move(20, 20)
        # Associe l'action de selection d'un jour
        self.connect(self.cal,
            QtCore.SIGNAL('selectionChanged()'),
            self.ShowDate)
        ...
    def ShowDate(self):
        date = self.cal.selectedDate()
        self.label.setText(str(date.toPyDate()))
```

code_qt_27.py

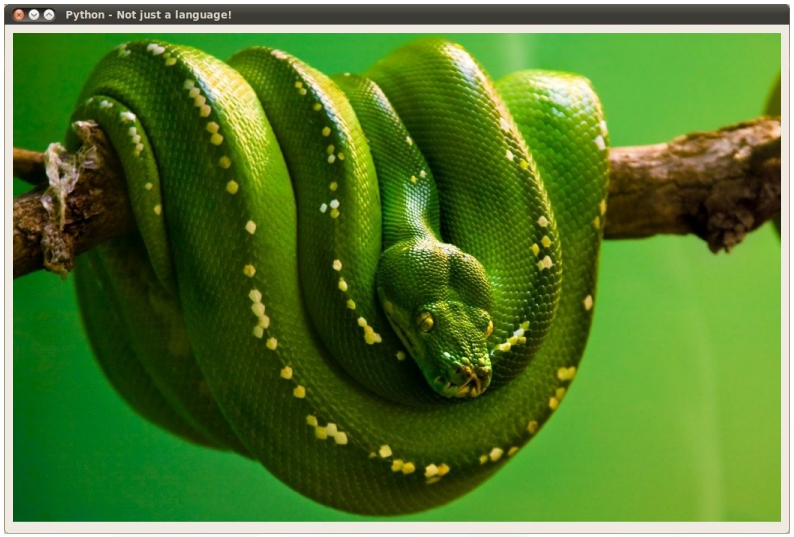
Visionneur d'images

- **Création** d'une fenêtre avec un conteneur.
- **Chargement** de l'image.
- Mise à jour du **conteneur**.

```
class ExampleWin(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation de l'objet : image a afficher
        pixmap = QtGui.QPixmap("python.jpg")

        # Creation du conteneur d'image
        label = QtGui.QLabel(self)
        # Chargement de l'image dans le conteneur
        label.setPixmap(pixmap)

        # Ajoute le controle d'affichage au
repartiteur
        hbox.addWidget(label)
        # Reglage du repartiteur a la fenetre
        self.setLayout(hbox)
        ...
```

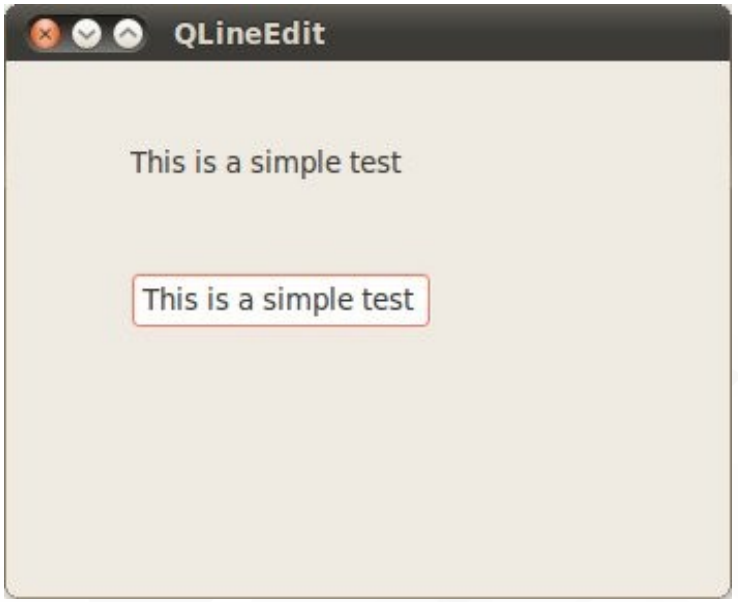


code_qt_28.py

Zone de texte dynamique

- **Création** d'une fenêtre avec 2 zones de texte.
- **Association** de la saisie à une méthode.
- Mise à jour de la **zone fixe**, sur saisie de texte.

```
class DynTextWin(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation de la zone de texte fixe
        self.label = QtGui.QLabel(self)
        # Creation de la zone de texte de saisie
        edit = QtGui.QLineEdit(self)
        # Associe l'action de modification
        self.connect(edit,
            QtCore.SIGNAL('textChanged(QString)'),
            self.onChanged)
        ...
    def onChanged(self, text):
        # Mise a jour du texte dans la zone fixe
        self.label.setText(text)
        # Ajustement de la taille de la zone fixe
        self.label.adjustSize()
```



code_qt_29.py

Séparateur de zones

- **Création** de 3 zones : **`QFrame()`**.
- Construction des « **`splitters`** ».
- **Association** de chaque zone au « **`splitter`** ».

```
class SplitWin(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation d'un repartiteur horizontal
        hbox = QtGui.QHBoxLayout(self)
        # Creation d'une zone de fenetre : Haut-Gauche
        topleft = QtGui.QFrame(self)
        # Reglage des proprietes de la frame

topleft.setFrameShape(QtGui.QFrame.StyledPanel)
        # Creation d'une zone de fenetre : Haut-Droit
        ...
        # Creation du decoupeur de fenetres N°1
        splitter1 =
QtGui.QSplitter(QtCore.Qt.Horizontal)
        splitter1.addWidget(topleft)
        splitter1.addWidget(topright)
        # Creation du decoupeur de fenetres N°2
        ...
        # Ajoute le splitter N°1 au splitter N°2
        splitter2.addWidget(splitter1)
```

`code_qt_30.py`

Boite de sélection

- **Création** d'une boite de sélection de valeurs.
- **Association** de la sélection à une méthode.
- Mise à jour **texte fixe**, sur modification de sélection.

```
class ComboBoxWin(QtGui.QWidget):
    def __init__(self, parent=None):
        # Creation d'une zone de texte fixe
        self.label = QtGui.QLabel("Ubuntu", self)

        # Creation de la liste deroulante
        combo = QtGui.QComboBox(self)
        # Ajoute les options de la liste deroulante
        combo.addItem("Ubuntu")
        combo.addItem("CentOS")
        ...
        # Associe l'action de changement
        self.connect(combo,
            QtCore.SIGNAL('activated(QString)'),
            self.onActivated)
    def onActivated(self, text):
        self.label.setText(text)
        self.label.adjustSize()
        ...
```

code_qt_31.py

Exercices ...

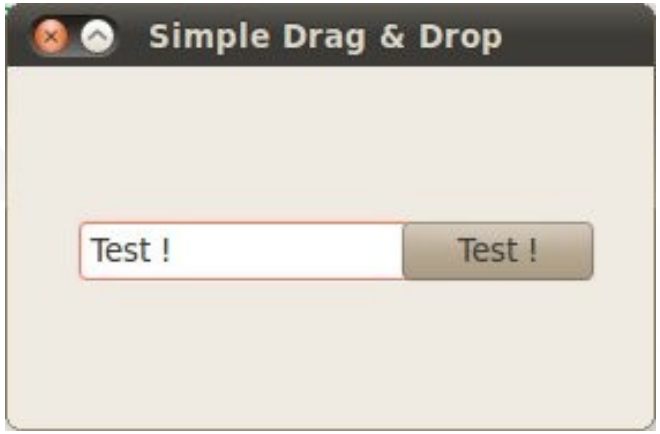
- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

Drag & Drop

- Drag & Drop Texte.
- Drag & Drop déplacement.

Drag & Drop Texte

- **Création** d'une fenêtre simple.
- **Autorise** le Drag & Drop sur la fenêtre.
- Classe **bouton** avec possibilité Drag & Drop.
- Création **zone** pour saisie de texte à « dropper » sur le bouton.
- Création **méthode** appelée sur action Drop, avec mise à jour texte bouton.
- Possible de « dropper » du texte externe.



code_qt_32.py

Drag & Drop déplacement

- **Création** d'une fenêtre simple.
- **Autorise** le Drag & Drop sur fenêtre.
- Classe **bouton** avec possibilité Drag & Drop.
- Création méthode appelée sur action Drop, avec mise à jour de la position du bouton.
- Possible de déplacer bouton avec clic droit.
- Possible de « dropper » du texte externe : crée un nouveau bouton.

code_qt_33.py

Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

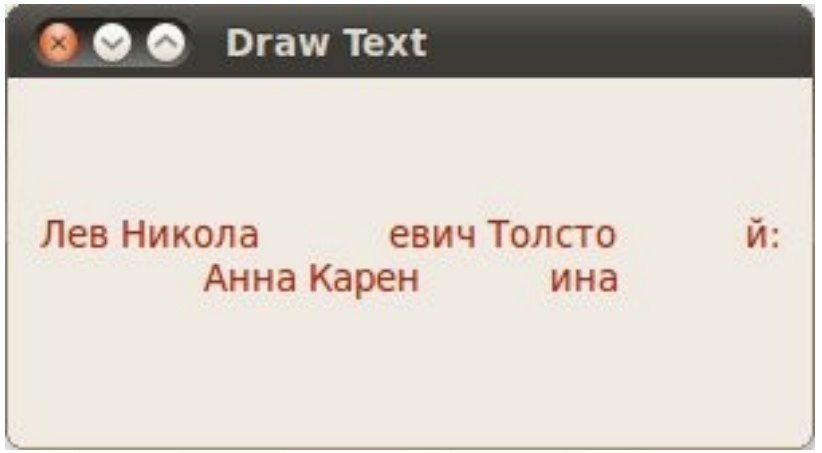
Dessin

- Evénement de dessin de fenêtre.
- Dessin de points.
- Dessin en couleurs.
- Types de ligne.
- Types de remplissage.

Evénement de dessin

- **Création** d'une fenêtre simple.
- Gestion de l'**événement** de dessin de fenêtre.

```
class UnicodeTextWin(QtGui.QWidget):
    def __init__(self):
        ...
        # Le texte a afficher en codage Unicode
        self.text = u'\u041b\u0435\u0432...'
        ...
    def paintEvent(self, event):
        # Creation d'un objet de base de dessin
        qp = QtGui.QPainter()
        # Demarrage
        qp.begin(self)
        # Appel de la methode d'ecriture du texte
        self.drawText(event, qp)
        # Fin
        qp.end()
    def drawText(self, event, qp):
        # Reglage de la couleur du crayon
        qp.setPen(QtGui.QColor(168, 34, 3))
        # Reglage de la police de caracteres du crayon
        qp.setFont(QtGui.QFont('Decorative', 10))
```

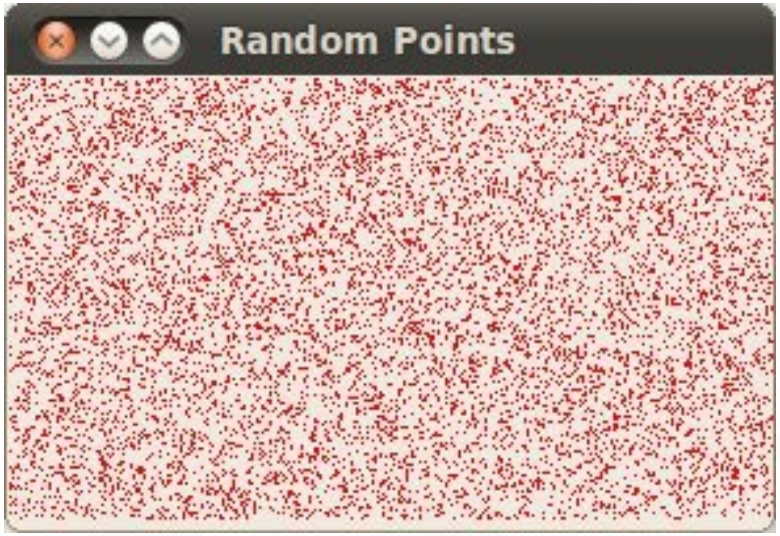


code_qt_34.py

Dessin de points

- **Création** d'une fenêtre simple.
- Gestion de l'**événement** de dessin de fenêtre.


```
class RandomWin(QtGui.QWidget):
    ...
    def paintEvent(self, e):
        # Creation d'un objet de base de dessin
        qp = QtGui.QPainter()
        # Demarrage
        qp.begin(self)
        self.drawPoints(qp)
        qp.end()
    def drawPoints(self, qp):
        ...
        size = self.size()
        # On trace 10000 points
        for i in range(10000):
            # Calcul aleatoire des coordonnees
            x = random.randint(1, size.width() - 1)
            y = random.randint(1, size.height() - 1)
            # Dessine le point
            qp.drawPoint(x, y)
```



code_qt_35.py

Pinceaux en couleurs

- **Création** d'une fenêtre simple.
- Gestion de l'**événement** de dessin de fenêtre.
- Dessine des **rectangles** de couleurs différentes.

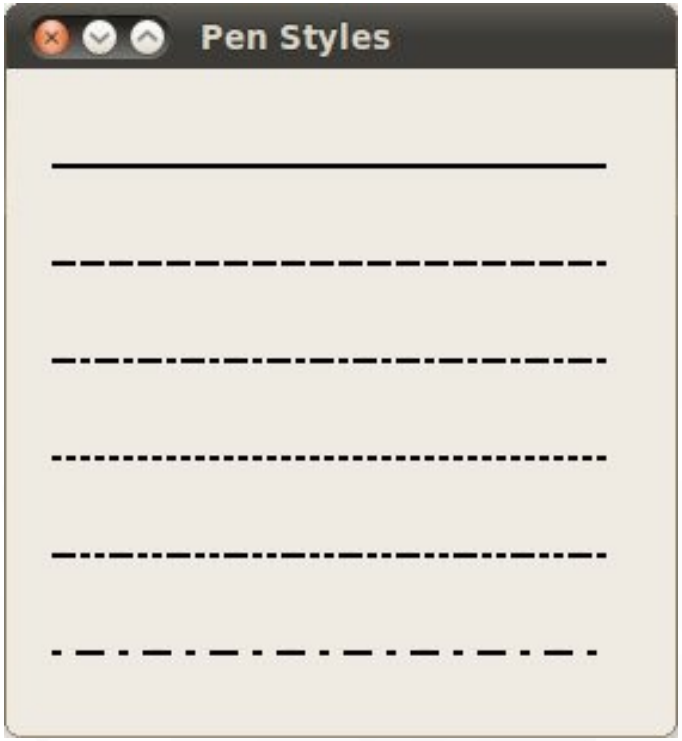
```
class RectColorWin(QtGui.QWidget):
    ...
    def paintEvent(self, e):
        # Creation d'un objet de base de dessin
        qp = QtGui.QPainter()
        qp.begin(self)
        self.drawRectangles(qp)
        qp.end()
    def drawRectangles(self, qp):
        # Couleur de remplissage
        color = QtGui.QColor(0, 0, 0)
        # Reglage de la couleur
        color.setNamedColor('#d4d4d4')
        # Reglage de la couleur du crayon
        qp.setPen(color)
        # Reglage de la couleur de remplissage
        qp.setBrush(QtGui.QColor(255, 0, 0, 80))
        # Dessine un rectangle
        qp.drawRect(10, 15, 90, 60)
        ...
```

code_qt_36.py

Types de ligne

- **Création** d'une fenêtre simple.
- Gestion de l'**événement** de dessin de fenêtre.
- Dessine des **rectangles** de couleurs différentes.

```
class LinesWin(QtGui.QWidget):
    ...
    def paintEvent(self, e):
        # Creation d'un objet de base de dessin
        qp = QtGui.QPainter()
        qp.begin(self)
        self.doDrawing(qp)
        qp.end()
    def doDrawing(self, qp):
        # Ligne pleine noire
        pen = QtGui.QPen(QtCore.Qt.black,
            2, QtCore.Qt.SolidLine)
        qp.setPen(pen)
        # Trace la ligne
        qp.drawLine(20, 40, 250, 40)
        ...
        # Ligne avec motif personnalise noire
        pen.setStyle(QtCore.Qt.CustomDashLine)
        # Reglage motif
        pen.setDashPattern([1, 4, 5, 4])
```

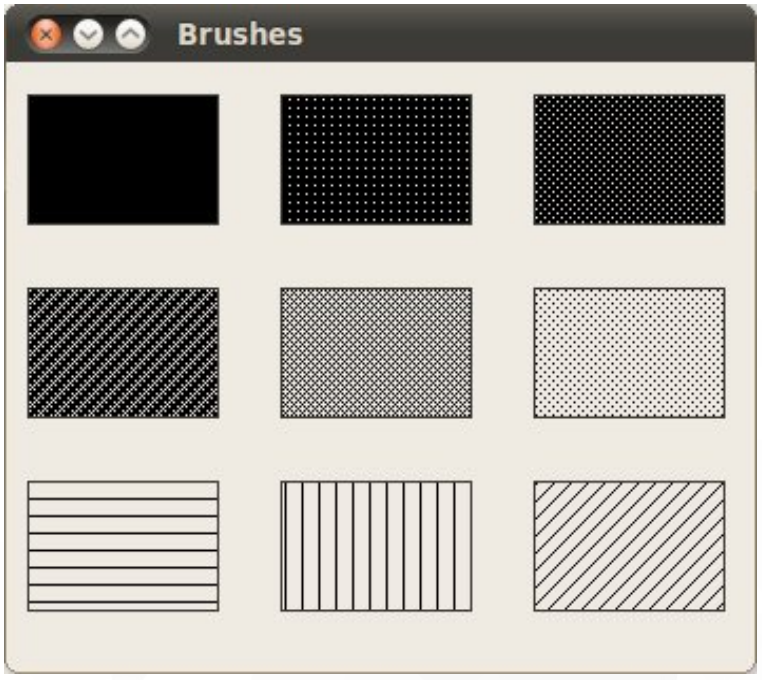


code_qt_37.py

Types de remplissage

- **Création** d'une fenêtre simple.
- Gestion de l'**événement** de dessin de fenêtre.
- Dessine des **rectangles** de remplissages différents.

```
class RectFillWin(QtGui.QWidget):
    ...
    def paintEvent(self, e):
        qp = QtGui.QPainter()
        qp.begin(self)
        self.drawBrushes(qp)
        qp.end()
    def drawBrushes(self, qp):
        # Creation d'un remplissage plein
        brush = QtGui.QBrush(QtCore.Qt.SolidPattern)
        qp.setBrush(brush)
        # Dessine le rectangle
        qp.drawRect(10, 15, 90, 60)
```



code_qt_38.py

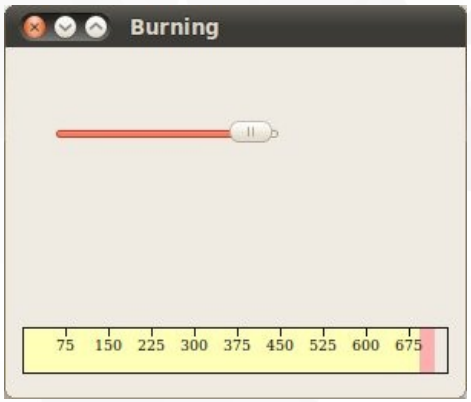
Exercices ...

- **Tester** les codes précédemment vus.
- **Modifier** des paramètres pour voir les effets.

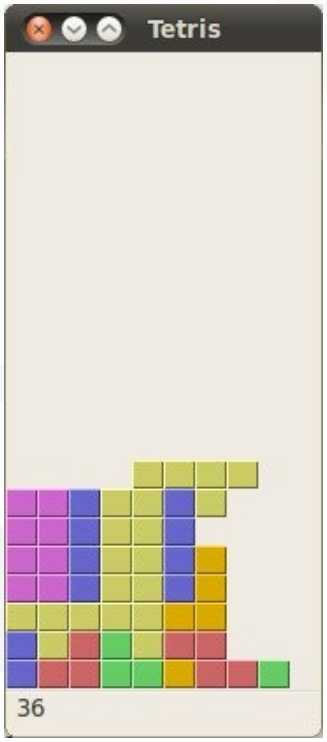
Personnalisation & Divers

- Création d'un Widget personnalisé.
- Visualisation d’adresse Web.
- Tetris.

Widgets personnalisés



code_qt_39.py



code_qt_40.py

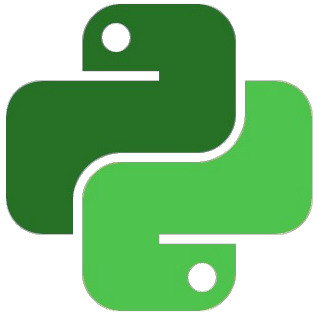
code_qt_web_viewer.py

Exercices ...

- **Tester** les codes précédemment vus.

Conclusions

- Demande de connaître **les bases** avant.
- **Qt** est maintenu et multi-plateformes.
- **Beaucoup** de contrôles disponibles.
- Possible de **personnaliser** ses propres contrôles.
- Trouver de **nouveaux** contrôles :
  - http://code.google.com/p/pyqt4-extrawidgets/
  - http://qt-apps.org/
- Existe un constructeur d'**interfaces** : **`QtDesigner`**.



Modules Python

Merci ...

- … de votre attention.
- … de ne pas hésiter à poser des questions.
- … de rester motivé pour la fin.