Construction d'IHM avec PyQt

Alexis NÉDÉLEC

Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Introduction

Qt (pronounced cute /kju :t/ or cuty /kju :ti :/)

- API orientée objet en C++
- framework pour l'environnement KDE
- toolkit Graphique C++
- Evolution de Qt1 à Qt5 + QtQuick :
  - TrollTech, Qt Software, Nokia, Digia ...
  - `http://www.qt.io/developers`
- licences GNU LGPL, commerciale
- multiplateformes : OS classiques et mobiles
- devise : "write once, compile anywhere"

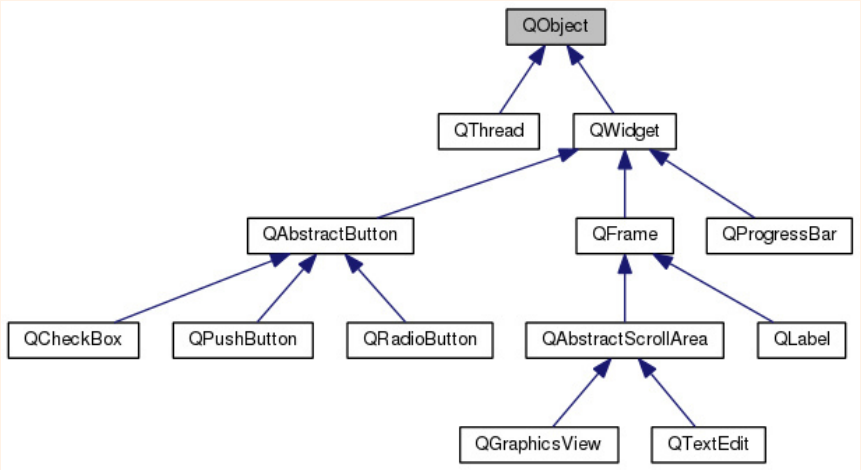
Qt API

Toolkit graphique ... mais pas seulement

- framework pour applications graphiques 2D/3D
- programmation événementielle, signaux/slots (`moc`)
- environnements de développement :
  - `Qt Designer` : générateur d'IHM (fichiers.ui)
  - `Qt Assistant` : documentation complète de Qt hors-ligne
  - `Qt Creator` : IDE Qt pour gestion de projet
- internationalisation (`tr()`, `Qt Linguist`)
- gestion de fichiers, connexion SGBD
- communication inter-processus, réseau
- W3C : XML, SAX, DOM
- multithreading
- ...

Qt API

Héritage de classes



Convention de nommage :

- Nom de classe : `Q` + `CamelCaseName`
  - `QPushButton`,`QGraphicsEllipseItem` ...
- Nom de méthode : `lowerCamelCaseName`
  - `QWidget::setMinimumSize()`

Qt API

Modules Qt Essentials

- `Qt Core` : classes de base pour tous les modules
- `Qt GUI` : composants graphiques 2D et 3D (OpenGL)
- `Qt Multimedia` : audio, video, radio et caméra
- `Qt NetWork` : faciliter la programmation réseaux
- `Qt QML` : pour les langages QML et javascript
- `Qt Quick` : création d'applications de manière déclarative
- `Qt SQL` : connexion, manipulation SGBD relationnels
- `Qt Test` : pour faire des test unitaires
- `Qt Widgets` : extension des fonctionnalités GUI
- ...

Qt API

Modules Qt Add-Ons

- `Qt 3D` : développement d'applications 3D
- `Qt Android Extras` : API spécifique pour Androïd
- `Qt Bluetooth` : Android, iOS, Linux, macOS, WinRT
- `Qt SCXML` : création de "State Machine" dans des applications
- `Qt Sensors` : données capteurs, reconnaissance de gestes
- `Qt Speech` : pour faire de la synthès vocale (text2speech)
- `Qt SVG` : affichage de contenu XML 2D
- `Qt XML` : SAX et DOM sur documents XML
- `Qt XML Pattern` : XPath, XQuery, XSLT,schemas XML ...
- ...

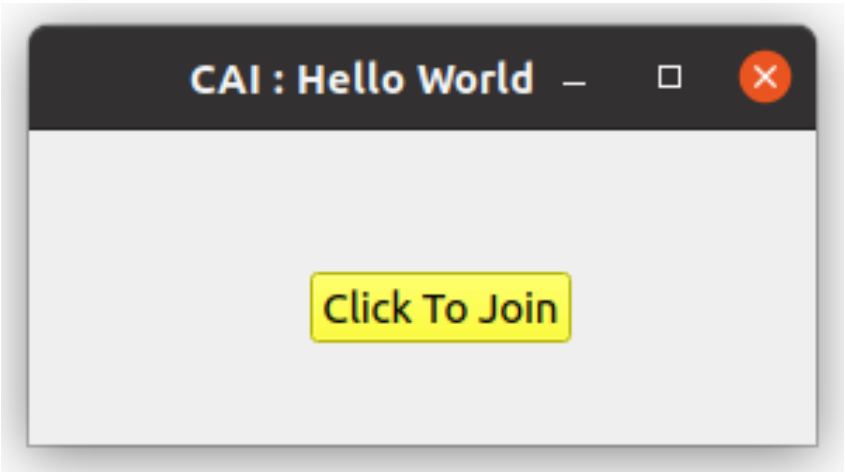
`https://doc.qt.io/qt-5/index.html`

Hello World

Fenêtre principale (main.cpp)

```
#include <QtWidgets>

int main(int argc, char *argv[]){
 QApplication app(argc,argv);
 QWidget mw;
 mw.resize(200,100);
 mw.setWindowTitle("CAI : Hello World");
```



Hello World

Composant graphique (main.cpp)

```
 QPushButton *button=
             new QPushButton("Click To Join",&mw);
 button->move(100,50);
 button->setStyleSheet("background-color:yellow;");
 mw.show();
 return app.exec();
}
```

Hello World

Environnement de développement

```
{logname@hostname}pwd
/chemin_depuis_la-racine/HelloWorld-1
{logname@hostname} qmake -project
{logname@hostname} tree
.
|-- main.cpp
|-- HelloWorld-1.pro
```

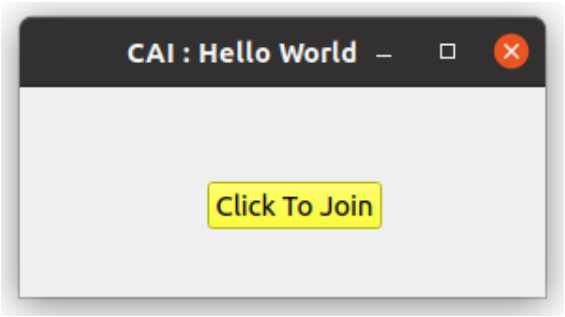
Fichier de configuration de projet (HelloWorld-1.pro)

```
QT += widgets  # to insert in generated file
SOURCES += main.cpp
TARGET = HelloWorld-1  # name of project directory
```

Hello World

Génération de `Makefile`, compilation, exécutable

```
{logname@hostname} qmake HelloWorld-1.pro; make
{logname@hostname} tree # in HelloWorld-1 directory
.
|-- HelloWorld-1
|-- HelloWorld-1.pro
|-- main.cpp
|-- main.o
|-- Makefile
{logname@hostname} HelloWorld-1
```



Interaction

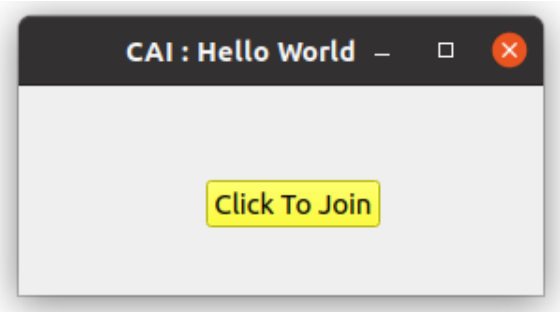
Signaux et slots

```
int main(int argc,char *argv[]){
...
  QPushButton *button= new QPushButton("Click To Join",
                                       &mw);
...
  Toplevel* top= new Toplevel(&mw);
  QWidget::connect(button,SIGNAL(clicked()),
                   top,SLOT(show()));
  window.show();
  return app.exec();
}
```

Interaction

Fenêtre secondaire (Toplevel)

```
#include <QtWidgets>
class Toplevel : public QDialog
{
  public :
    Toplevel(QWidget* parent);
};
```



Interaction

Fenêtre secondaire (Toplevel)

```
#include <toplevel.h>
Toplevel::Toplevel(QWidget* parent):QDialog(parent){
  this->setWindowTitle("CAI : Dialog Window");
  QVBoxLayout *layout= new QVBoxLayout();
  QLabel *image= new QLabel(this);
  image->setPixmap(QPixmap("pyqt.jpg"));
  QPushButton *button= new QPushButton("Hide me !",
                                       this);
  QWidget::connect(button,SIGNAL(clicked()),
                   this,SLOT(hide()));
  layout->addWidget(image);
  layout->addWidget(button);
  this->setLayout(layout);
}
```

Interaction

Configuration de projet(HelloWorld-2.pro)

```
QT += widgets

DEPENDPATH += . Include Src
INCLUDEPATH += . Include

HEADERS += Include/toplevel.h
SOURCES += Src/main.cpp Src/toplevel.cpp

TARGET = HelloWorld-2
```

Génération de Makefile, compilation, exécutable

```
logname@hostname} qmake -o Makefile HelloWorld-2.pro
logname@hostname} make
```

Interaction

Environnement de développement

```
{logname@hostname} tree
HelloWorld-2
|-- HelloWorld-2
|-- HelloWorld-2.pro
|-- Include
|   |-- toplevel.h
|-- main.o
|-- Makefile
|-- pyqt.jpg
|-- Src
|   |-- main.cpp
|   |-- toplevel.cpp
|-- toplevel.o
2 directories, 9 files
```

PyQt

Bindings pour Python

- `pyQt` : le plus ancien,développé par Riverbank Computing
- `pySide` : lancé par Nokia pour introduire une licence LGPL

PyQt vs Pyside

Hello World !

```
import sys
from PyQt5 import QtWidgets
# from PySide import QtWidgets

def gui(parent):
    button=QtWidgets.QPushButton("Click To Join",parent)
    button.move(100,50)
    button.setStyleSheet("background-color:yellow;")
```

PyQt

Hello World !

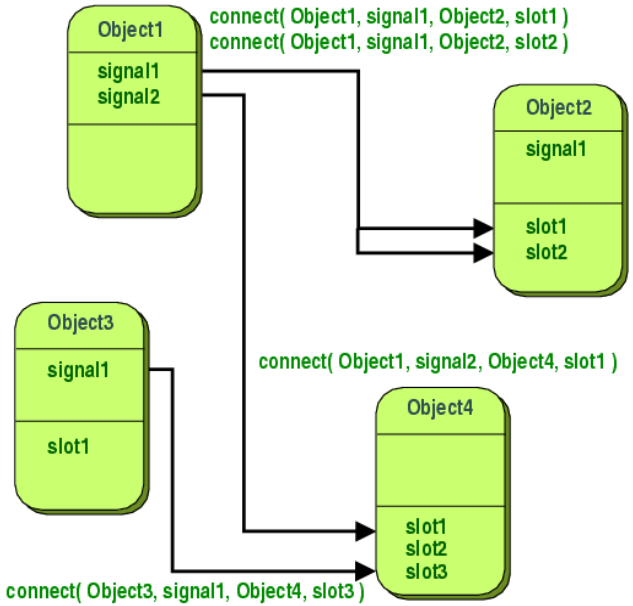
```
if __name__ == '__main__':
    app=QtWidgets.QApplication(sys.argv)
    mw=QtWidgets.QWidget()
    mw.setGeometry(300,150,300,400)
    mw.setWindowTitle("PyQt5 : Hello 1")
    gui(mw)
    mw.show()
    sys.exit(app.exec_())
```

- instancier un objet d'application (module `QtWidgets`)
- créer une fenêtre principale (composant `QtWidget`)
- système de fenêrage (origine en haut à gauche)
- création d'arbre de composants graphiques

Signaux et slots

Communication entre composants

- changement d'état d'un objet : émission de signal
- réception de signal par un objet : déclenchement d'un slot
- un slot est un comportement (une méthode) à activer
- programmation par composants (modèle "multi-agents")



Signaux et slots

Héritage QWidget

```
class SliderLCD(QtWidgets.QWidget):
  def __init__(self, parent=None):
    QtWidgets.QWidget.__init__(self, parent)
    lcd=QtWidgets.QLCDNumber(self)
    slider=QtWidgets.QSlider(QtCore.Qt.Horizontal,self)
    slider.valueChanged.connect(lcd.display)
```

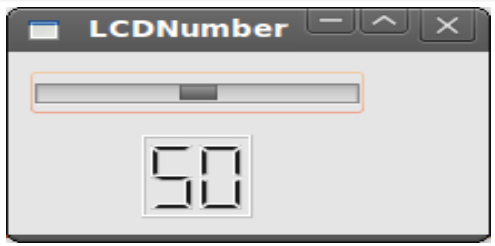
- PyQt4 :
  `connect(a,SIGNAL("signal(arg)",b, SLOT("slot(arg)")`
- PyQt5 :
  `a.signal.connect(b.slot)`

Signaux et slots

Héritage QWidget

```
        vbox=QtWidgets.QVBoxLayout()
        vbox.addWidget(slider)
        vbox.addWidget(lcd)
        self.setLayout(vbox)

if __name__ == "__main__" :
    app=QtWidgets.QApplication(sys.argv)
    w=SliderLCD()
    w.show()
    sys.exit(app.exec_())
```



Signaux et slots

Communication entre composants

- un signal, plusieurs slots et réciproquement
- l'émetteur n'a pas à connaître le récepteur et réciproquement
- l'émetteur ne sait pas si le signal est reçu (broadcast)
- un slot peut avoir moins de paramètres qu'un signal
- aspect central de la programmation Qt
- `SLOT, SIGNAL` macros : précompilation C++ (`moc`)

Signaux et slots

Héritage QObject

```
from PyQt5 import QtCore
from PyQt5.QtCore import pyqtSignal,pyqtSlot

class SigSlot (QtCore.QObject) :
  value_changed = pyqtSignal(int)

  def __init__(self):
    QtCore.QObject.__init__(self)
    self.value=0
```

Déclaration d'un signal : `pyqtSignal(arg)`

Signaux et slots

Héritage QObject

```
def get_value(self) :
  return self.value

def set_value(self,v) :
  if (v!=self.value) :
    self.value=v
    self.value_changed.emit(v)
```

Emission d'un signal `emit(arg)`

Signaux et slots

Héritage QObject

```
if __name__ == "__main__" :
  a,b=SigSlot(),SigSlot()
  a.value_changed.connect(b.set_value)
#  b.value_changed.connect(a.set_value)
  b.set_value(10)
  print(a.get_value()) # 0 or 10 ?
  a.set_value(100)
  print(b.get_value()) # 10 or 100  ?
```

Signaux et slots

Passage d'arguments

```
from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot
class TalkAndListen(QObject):
  signal_talk = pyqtSignal(str)
  def __init__(self,name):
    QObject.__init__(self)
    self.name=name
  def listen_to_me(self,text):
    self.signal_talk.emit(self.name+" who said : "+text)
  @pyqtSlot(str)
  def slot_listen(self,text):
    print(self.name+" listen to "+text)
```

Transmission de données entre composants :

- `pyqtSignal(arg),pyqtSlot(arg)`

Signaux et slots

Passage d'arguments

```
if __name__ == "__main__" :
    talker = TalkAndListen("Dupont")
    listener=TalkAndListen("Durand")

    talker.signal_talk.connect(listener.slot_listen)
    talker.listen_to_me("Did you hear what I say !")

    listener.signal_talk.connect(talker.slot_listen)
    listener.listen_to_me("I'm not deaf !")
```

```
{logname@hostname} python talker.py
Durand listen to Dupont who said : Did you hear what I say !
Dupont listen to Durand who said : I'm not deaf !
```

Signaux et slots

Passage d'arguments

```
class Keypad(QtWidgets.QWidget):
  def __init__(self,nbuttons=10,parent=None) :
    super(Keypad, self).__init__()
    self.layout = QtWidgets.QGridLayout(self)
    self.buttons=[]
    self.create_buttons(nbuttons)
   ...
```

Comment récupérer les chiffres du pavé numérique ?

Signaux et slots

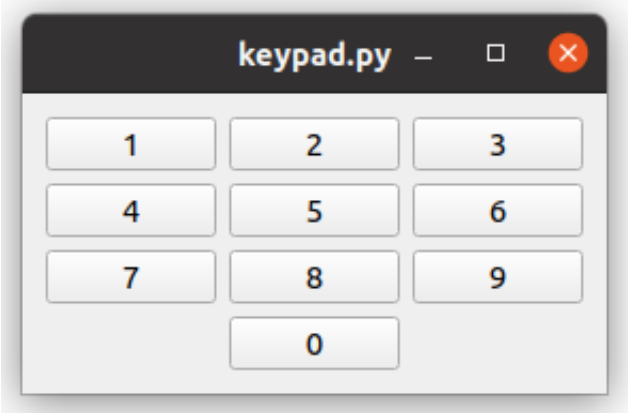
Passage d'arguments : fonctions anonymes (lambda)

```
def create_buttons(self,number) :
    for i in range(1,number) :
      button=QtWidgets.QPushButton(str(i),self)
      self.layout.addWidget(button,i//3,i%3)
      button.clicked.connect(lambda state,x=i : \
                    self.on_selected(state,x) )
      self.buttons.append(button)
      self.layout.addWidget(button)
    button=QtWidgets.QPushButton(str(0),self)
    button.clicked.connect(lambda state,x=0: \
                  self.on_selected(state,x) )
    self.buttons.append(button)
    self.layout.addWidget(button,4,1)
```

Signaux et slots

Passage d'arguments : fonctions anonymes (lambda)

```
  def on_selected(self,state,index):
    print('state', state)
    print('index', index)
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    keypad = Keypad()
    keypad.show()
    sys.exit(app.exec_())
```



Événements

Classes d'événements : dérivées de la classe abstraite `QEvent`

- `QKeyEvent, QMouseEvent` : actions clavier,souris
- `QResizeEvent, QPaintEvent` : retailler, redessiner
- `QExposeEvent, QHideEvent` : affichage de fenêtres
- ...
- `QTouchEvent, QGestureEvent` : application Post-WIMP

Signal/Slots vs Evénements

- un événement n'émet pas de signal, est généré de l'"extérieur"
- les signaux/slots permettent la communication inter-classes
- les signaux sont utiles pour la manipulation de widgets
- les événements sont nécessaires pour implémenter des widgets

Événements

Héritage QWidget : surdéfinition de méthodes

```
class Scribble(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        ...
    def mousePressEvent(self,event) :
        ...
    def mouseMoveEvent(self,event) :
        ...
        self.update()
    def mouseReleaseEvent(self,event) :
        ...
        self.update()
    def paintEvent(self,event) :
        ...
```

Événements

Héritage QWidget : surdéfinition de méthodes

```
import sys
from PyQt5 import QtCore,QtGui,QtWidgets
// TODO  : class Scribble

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    mw = Scribble()
    mw.resize(300,200)
    mw.show()
    app.exec_()
```

Événements

Héritage `QWidget` : surdéfinition de méthodes

```
class Scribble(QtWidgets.QWidget):
  def __init__(self):
    super().__init__()
    self.start=QtCore.QPoint(0,0)
    self.end=QtCore.QPoint(0,0)
    self.pen_color=QtCore.Qt.blue;
    self.pen_width=3;
```

Événements

Héritage `QWidget` : surdéfinition de méthodes

```
def mousePressEvent(self,event) :
  if event.button() == QtCore.Qt.LeftButton :
    self.start = self.end = event.pos();
def mouseMoveEvent(self,event) :
  if event.buttons() & QtCore.Qt.LeftButton :
    self.end = event.pos();
  self.update()
def mouseReleaseEvent(self,event) :
  if  event.button()== QtCore.Qt.LeftButton :
    self.update()
```

Événements

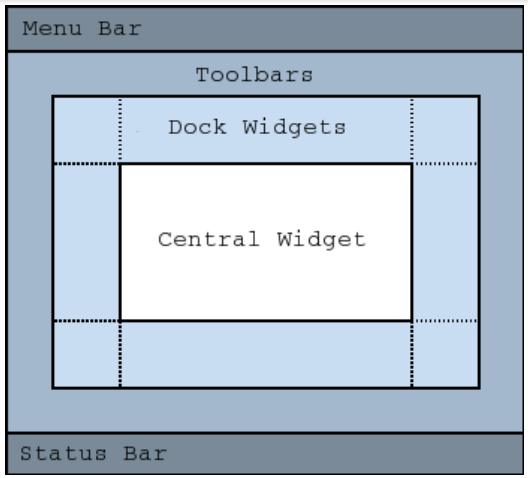
Héritage QWidget : surdéfinition de méthodes

```
  def paintEvent(self,event) :
    painter=QtGui.QPainter(self)
    painter.setPen(QtGui.QPen(
                                self.pen_color,
                                self.pen_width,
                                QtCore.Qt.SolidLine,
                                QtCore.Qt.RoundCap,
                                QtCore.Qt.RoundJoin
                               )
                   )
    painter.drawLine(self.start,self.end);
  def resizeEvent(self,event) :
    print(self.width(),self.height())
```

Création d'IHM

`QMainWindow` : Fenêtre principale

- barres de menu, d'outils, de statut
- zone centrale (cliente)
- autres fonctionnalités



Création d'une fenêtre principale

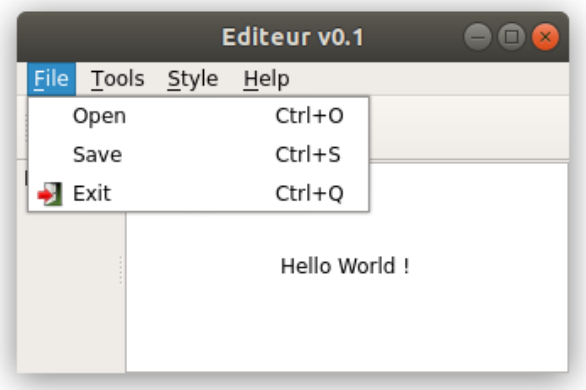
- héritage de `QMainWindow`
- création des zones de travail dans le constructeur

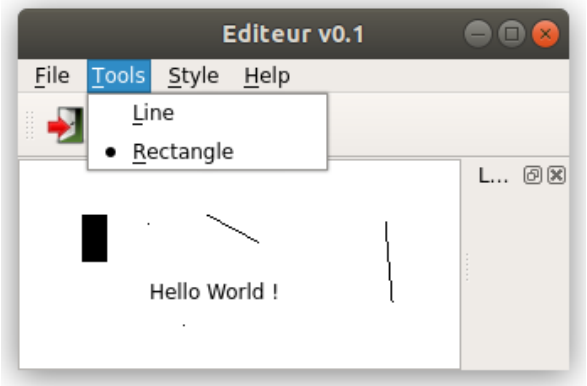
Création d'IHM

Création d'une fenêtre principale

```
from PyQt5.QtWidgets import QApplication
from mainwindow import MainWindow

if __name__ == "__main__" :
    app=QApplication(sys.argv)
    mw=MainWindow()
    mw.show()
    sys.exit(app.exec_())
```





Création d'IHM

MainWindow : création de la fenêtre principale

```
class MainWindow(QMainWindow):
  def __init__(self):
    super(MainWindow, self).__init__(*args,**kwargs)
    self.setWindowTitle("PyQt5 : MainWindow (v.1)")
    self.create_scene()
    self.create_actions()
    self.create_toolbars()
    self.create_menus()
```

MainWindow.create_scene() : zone cliente

- `QMainWindow.setCentralWidget(widget)`
- Editeurs : `QTextEdit, QGraphicsView ...`

Création d'IHM

MainWindow.create_scene() : zone cliente

```
def create_scene(self) :
  view=QGraphicsView()
  self.scene=Scene(self)  # QGraphicsScene inheritance
  view.setScene(self.scene)
  self.setCentralWidget(view)
```

- `QGraphicsView` : La Vue
- `QGraphicsScene` : Le Modèle associé à la vue
- `self.scene=Scene(self)` : notre Modèle de scène
- `setCentralWidget(view)` : zone cliente (La vue)

Création d'IHM

Scene : gestion de la zone cliente

```
class Scene (QGraphicsScene) :
  def __init__(self,*args,**kwargs):
    super(Scene, self).__init__(*args,**kwargs)
    self.pen=QPen()
    self.pen.setWidth(2)
    self.pen.setColor(Qt.red)
    self.brush=QBrush(Qt.green)
    self.begin=QtCore.QPoint(0,0)
    self.end=QtCore.QPoint(0,0)
    self.create()          # creation d'items graphiques
```

Fenêtre principale

Scene.create() : création de la scène

```
def create(self) :
  text=self.addText("Hello World !") # add item
  text.setPos(0,0)
  text.setVisible(True)
  rect=QGraphicsRectItem(50,100,200,50)
  rect.setFlag(QGraphicsItem.ItemIsMovable)
  rect.setPen(self.pen)
  rect.setBrush(self.brush)
  self.addItem(rect)                    # add item
```

Fenêtre principale

Scene : gestion d'événements

```
def mousePressEvent(self,event):
  self.begin=self.end=event.scenePos()
def mouseMoveEvent(self,event):
  self.end=event.scenePos()
def mouseReleaseEvent(self,event):
  self.end=event.scenePos()
  rect=QtWidgets.QGraphicsRectItem(
                    self.begin.x(),self.begin.y(),
                    self.end.x()-self.begin.x(),
                    self.end.y()-self.begin.y())
  rect.setPen(self.pen)
  rect.setBrush(self.brush)
  self.addItem(rect)                # add item
```

Fenêtre principale

MainWindow.create_actions() : création des actions

```
  def create_actions(self) :
    name="Open"
    self.action_file_open=QAction(
        QIcon('Icons/open.png'),name,self)
    self.action_file_open.setStatusTip("Open File")
    self.action_file_open.setCheckable(True)
    self.action_file_open.triggered.connect(
        lambda status,selection=name :
        self.on_triggered_action(status,selection))
    ...
```

Convention de nommage :

`self.action_<menu_name>_<menu_item>=QAction(...)`

Fenêtre Principale

MainWindow.create_actions() : création des actions

```
self.actions_tools=QtWidgets.QActionGroup(self)
self.action_tools_line=QtWidgets.QAction(
        self.tr("&Line"),self)
self.action_tools_line.setCheckable(True)
self.action_tools_line.setChecked(True)
self.action_tools_rect=QtWidgets.QAction(
        self.tr("&Rectangle"),self)
...
self.actions_tools.addAction(self.action_line)
self.actions_tools.addAction(self.action_rect)
...
```

Regroupement d'actions : `QActionGroup`

Fenêtre Principale

MaiWindow.create_toolbars() : création de barre d'outils

```
def create_toolbars(self) :
  self.toolbar=QToolBar("Main Toolbar")
  self.addToolBar(self.toolbar)
  self.toolbar.setIconSize(QSize(16,16))
  self.setStatusBar(QStatusBar(self))
  self.toolbar.addAction(self.action_file_open)
  self.toolbar.addAction(self.action_file_exit)
  ...
```

`QToolBar` : pour les actions les plus fréquemment utilisées

Fenêtre Principale

MaiWindow.create_menus() : création de barre d'Actions

```
def create_menus(self) :
  menubar=self.menuBar()
  # menu file
  self.menu_file=menubar.addMenu("&File")
  self.menu_file.addAction(self.action_file_open)
  ...
  # menu style
  self.menu_style=menubar.addMenu("&Style")
  # menu style : submenu Pen
  self.pen_style=QMenu('Pen',self)
  self.menu_style.addMenu(self.pen_style)
  self.pen_style.addAction(self.action_style_pen_color)
  ....
```

Fenêtre Principale

MainWindow : connexion action/comportement (Signal/Slot)

```
self.action_..._<menu_item>.triggered.connect(
    lambda status,selection=name :
    self.on_triggered_action(status,selection))
```

MainWindow.on_triggered_action()

```
def on_triggered_action(self,status,selection):
  if selection=="Open" :
    name=self.file_open() # dialog (to select file)
    self.scene.load(name)
  elif selection=="Pen Color" :
    color=self.style_color() # dialog (to select color)
    if color :
      self.scene.set_pen_color(color)
    ...
```

Fenêtre Principale

MainWindow : interaction sur la zone client

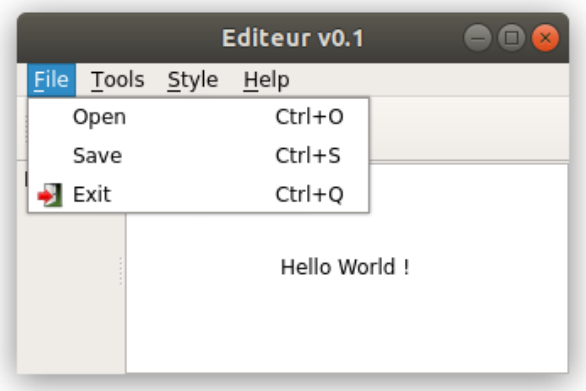
```
class Scene (QtWidgets.QGraphicsScene) :
  def __init__(self,*args,**kwargs):
    super(Scene, self).__init__(*args,**kwargs)
    self.pen=QPen()
    ...
    self.tool="Line"
    self.item=None
  def set_pen_color(self,color) :
    self.pen.setColor(color)
  ...
  def set_tool(self,tool) :
    self.tool=tool
  ...
```

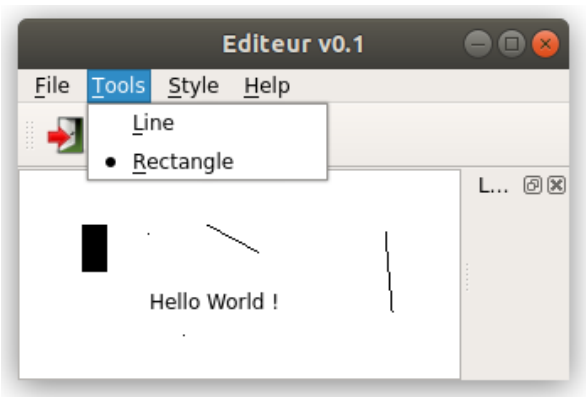
Fenêtre Principale

Interaction sur la zone client

```
def mouseReleaseEvent(self, event):
  self.end =event.scenePos()
  if self.tool=="Line" :
    item=QtWidgets.QGraphicsLineItem(
          self.begin.x(),self.begin.y(),
          self.end.x(),self.end.y()
        )
    item.setPen(self.pen)
  elif self.tool=="Rectangle" :
    ...
```

Fenêtre Principale





Palette d'outils

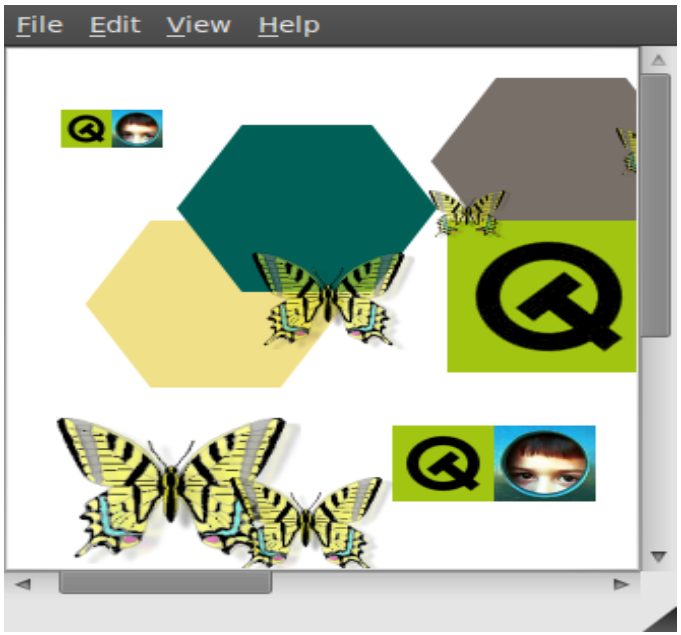
```
self.dock=QtWidgets.QDockWidget("Left Right Dock",\
                                self)
self.dock.setAllowedAreas(\
            QtCore.Qt.LeftDockWidgetArea\
          | QtCore.Qt.RightDockWidgetArea )
self.addDockWidget(\
            QtCore.Qt.LeftDockWidgetArea,\
            self.dock )

...
```

Graphics Framework

Créer et interagir avec des objets graphiques (`QGraphicsItem`)

- `QGraphicsScene` : la scène
- `QGraphicsView` : les vues
- `QGraphicsItem` : les objets



Graphics Framework

QGraphicsScene

Conteneur d'objets (items) graphiques

- gérer un grand nombres d'éléments graphiques
- propager les évènements aux objets graphiques
- gérer les états des éléments (sélection, focus ...)
- fonctionnalités de rendu
- ...

Graphics Framework

QGraphicsView

- widget de visualisation de la scène
- associer plusieurs vues à une scène
- ...

QGraphicsItem

Eléments standards :

- rectangle : `QGraphicsRectItem`
- ellipse : `QGraphicsEllipseItem`
- texte : `QGraphicsTextItem`
- ...

Graphics Framework

Création de scène

```
import sys
from PyQt5 import QtCore,QtGui
from PyQt5.QtWidgets import QApplication,QWidget,\
QGraphicsScene,QGraphicsView,QGraphicsItem

app=QApplication(sys.argv)
scene=QGraphicsScene()
#------------- scene creation --------------------
rect=scene.addRect(QtCore.QRectF(0,0,100,100))
rect.setFlag(QGraphicsItem.ItemIsMovable)
#-------------------------------------------------
view=QGraphicsView(scene)
view.show()
sys.exit(app.exec_())
```

Graphics Framework

Repérage de scène

```
view = QtWidgets.QGraphicsView()
view.setGeometry(QtCore.QRect(0,0,600,500))
#------------- scene creation --------------------
scene.setSceneRect(-150,-150,300,300)
top=QtCore.QLineF(scene.sceneRect().topLeft(),
                  scene.sceneRect().topRight())
left=QtCore.QLineF(scene.sceneRect().topLeft(),
                   scene.sceneRect().bottomLeft())
right=QtCore.QLineF(scene.sceneRect().topRight(),
                    scene.sceneRect().bottomRight())
bottom=QtCore.QLineF(scene.sceneRect().bottomLeft(),
                     scene.sceneRect().bottomRight())
view.setScene(scene)
```

Graphics Framework

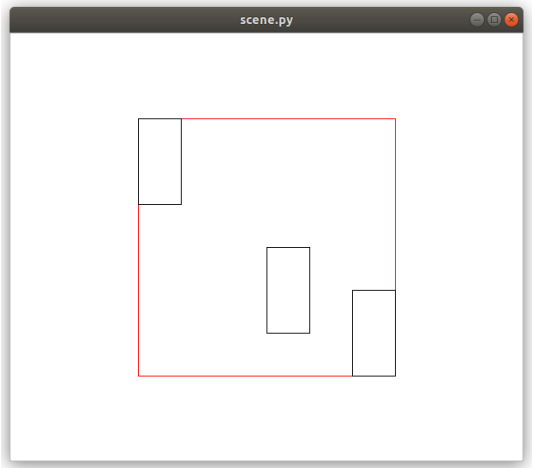
Repérage de scène

```
pen=QtGui.QPen(QtCore.Qt.red)
scene.addLine(top,pen)
scene.addLine(left,pen)
scene.addLine(right,pen)
scene.addLine(bottom,pen)
```

Graphics Framework

Repérage de scène

```
item=QtWidgets.QGraphicsRectItem(-150,-150, 50, 100)
scene.addItem(item)
item=QtWidgets.QGraphicsRectItem(0,0,50,100)
scene.addItem(item)
item=QtWidgets.QGraphicsRectItem(100,50, 50, 100)
scene.addItem(item)
#-----------------------------------------
```



Graphics Framework

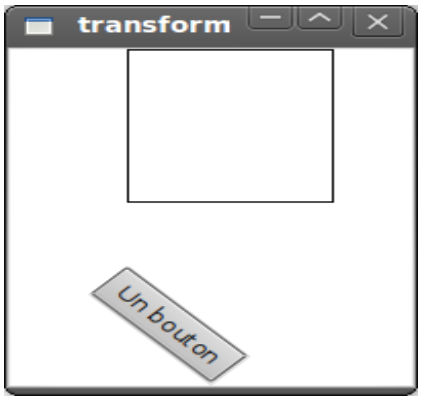
Transformations géométriques

```
#------------- scene creation -------------------
rect=scene.addRect(QtCore.QRectF(0, 0, 100, 100))
rect.setFlag(QGraphicsItem.ItemIsMovable)
button=QPushButton("Un bouton")
proxy=QGraphicsProxyWidget()
proxy.setWidget(button)
scene.addItem(proxy)
scene.setSceneRect(0,0, 300, 300)
matrix=QtGui.QTransform()
matrix.rotate(45)
matrix.translate(100,0)
matrix.scale(1,2)
proxy.setTransform(matrix);
#-----------------------------------------------
```

Graphics Framework

Intégration d'applications

```
#------------- scene creation -------------------
web = QWebView()
web.load(QtCore.QUrl("http://www.enib.fr"))
rect=scene.addRect(QtCore.QRectF(0, 0, 100, 100))
proxy = QGraphicsProxyWidget()
proxy.setWidget(web)
scene.addItem(proxy)
#------------------------------------------------
```



Graphics Framework

Interaction avec les items graphiques

```
class WebProxy(QGraphicsProxyWidget) :
  def __init__(self):
    super().__init__()
    self.angle=0.0
  def get_rotate(self) :
    return self.angle
  def set_rotate(self,angle) :
    self.angle=angle
```

Graphics Framework

Interaction avec les items graphiques

```
class Scene(QGraphicsScene) :
  def __init__(self):
    super().__init__()
    #------------- scene creation ------------------
    self.setSceneRect(0,0,1000,800)
    web=QWebView()
    web.load(QtCore.QUrl("http://www.developpez.com"))
    self.proxy=WebProxy()
    self.proxy.setWidget(web)
    self.addItem(self.proxy)
    #-----------------------------------------------
```

Graphics Framework

Interaction avec les items graphiques

```
def mouseMoveEvent(self,event) :
  if (event.buttons() & QtCore.Qt.LeftButton) :
    delta=QtCore.QPointF(event.scenePos() \
                         - event.lastScenePos())
    rotation=delta.x()
    self.proxy.set_rotate(rotation \
                          + self.proxy.get_rotate())
  matrix=QtGui.QTransform()
  matrix.translate(self.proxy.widget().width()/2.0,
                   self.proxy.widget().height()/2.0)
  matrix.rotate(self.proxy.get_rotate(),
                QtCore.Qt.YAxis)
  self.proxy.setTransform(matrix)
```

Graphics Framework

Interaction avec les items graphiques

```
if __name__ == "__main__" :
  app=QApplication(sys.argv)
  button = QPushButton("Quit application")
  button.move(100,100)
  button.clicked.connect(qApp.quit)
  proxy = QGraphicsProxyWidget()
  proxy.setWidget(button)
  scene=Scene()
  scene.addItem(proxy)
  view=QGraphicsView(scene)
  view.setWindowTitle("CAI : Scene WEB")
  view.show()
```

Editeur Graphique

Labos : implémenter un éditeur graphique

- `mainwindow.py` : fenêtre principale (`QMainWindow`)
- `scene.py` : zone cliente (`QGraphicsSene`)

classe `MainWindow`

```
class MainWindow(QMainWindow):
  def __init__(self, *args, **kwargs):
    super(MainWindow, self).__init__(*args, **kwargs)
    self.setWindowTitle("PyQt5 : MainWindow")
    self.create_scene()
    self.create_actions()
    self.create_toolbars()
    self.create_menus()
```

Editeur Graphique

classe Scene :création de la scène

```
class Scene (QGraphicsScene) :
  def __init__(self,*args,**kwargs):
    super(Scene, self).__init__(*args,**kwargs)
    self.pen=QPen()
    self.pen.setWidth(2)
    self.pen.setColor(Qt.red)
    self.brush=QBrush(Qt.green)
    self.begin=QPointF(0.0,0.0)
    self.end=QPointF(0.0,0.0)
    self.item=None
    self.create()
```

Editeur Graphique

classe Scene : gestion d'événements

```
def mousePressEvent(self, event):
  self.begin=self.end = event.scenePos()
  self.item=self.itemAt(self.begin,QtGui.QTransform())
  if self.item :
    self.offset =self.begin-self.item.pos()
def mouseMoveEvent(self,event):
  if self.item :
    self.item.setPos(event.scenePos()-self.offset)
    self.end=event.scenePos()
```

Editeur Graphique

classe Scene : gestion d'événements

```
def mouseReleaseEvent(self, event):
  self.end=event.scenePos()
  if self.item :
    self.item.setPos(event.scenePos()-self.offset)
    self.item=None
  elif self.tool=="line" :
    self.addLine(self.begin.x(),self.begin.y(),
                 self.end.x(),self.end.y(),self.pen)
  else :
    print("no item selected and nothing to draw !")
```

Annexe : onglets de test

Classe d'onglets : création des onglets

```
class Toplevel(QtWidgets.QWidget) :
  def __init__(self,parent=None):
    QtWidgets.QWidget.__init__(self,parent)
    self.setWindowTitle("CAI : Dialog Window");
    self.tabs=QtWidgets.QTabWidget(self)
    self.tabs.addTab(self.dislay_image(),"Image");
    self.tabs.addTab(self.progressbar(),"Progression");
    self.tabs.addTab(self.widget_to_test(),
                     "Widgets to test");
```

Onglets de tests

Classe d'onglets : création des onglets

```
button=QtWidgets.QPushButton("Hide me !", self)
button.clicked.connect(self.hide)
layout=QtWidgets.QVBoxLayout()
layout.addWidget(self.tabs)
layout.addWidget(button)
self.setLayout(layout)
```

Chaque onglet contiendra un programme de test :

- `self.dislay_image()` : affichage de pixmap
- `self.progressbar()` : barre de progression
- `self.widget_to_test()` : instanciation de widgets
- ...

Onglets de tests

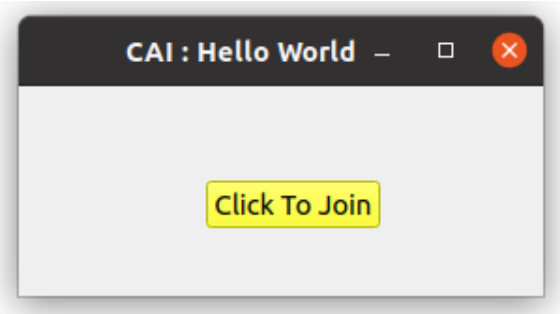
Application de test

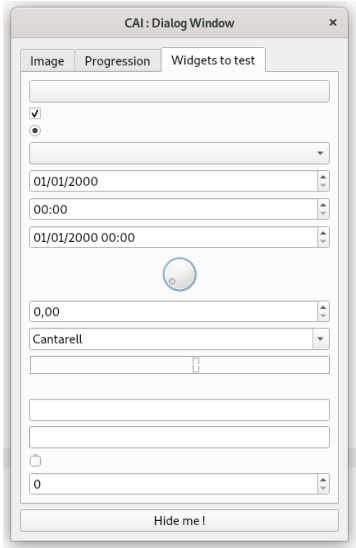
```
if __name__ == "__main__" :
  app=QtWidgets.QApplication(sys.argv)
  mw=QtWidgets.QWidget()
  mw.resize(200, 100)
  mw.setWindowTitle("CAI : Onglets")
  layout=QtWidgets.QVBoxLayout()
  button=QtWidgets.QPushButton("Click to Join",mw)
  button.move(100, 50)
  button.setStyleSheet("background-color:yellow;")
  layout.addWidget(button)
```

Onglets de tests

Application de test

```
mw.setLayout(layout)
mw.show()
top=Toplevel();
button.clicked.connect(top.show)
mw.show()
app.exec()
```





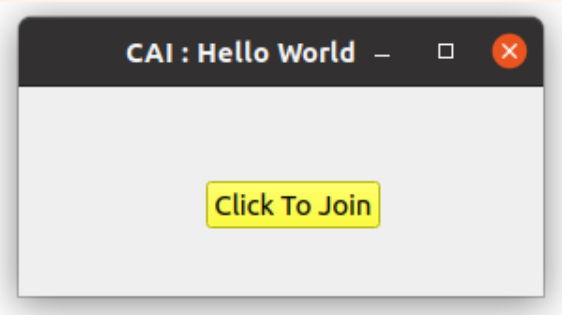
Onglets de tests

Toplevel.dislay_image() : affichage de pixmap

```
def dislay_image(self,name="pyqt.jpg") :
    width,height=200,100
    image=QtWidgets.QLabel()
    pixmap=QtGui.QPixmap(name)
    image.setPixmap(pixmap.scaled(width,height))
    onglet=QtWidgets.QWidget()
    vbox=QtWidgets.QVBoxLayout()
    vbox.addWidget(image)
    onglet.setLayout(vbox)
    onglet.setStyleSheet("background-color:black;")
    return onglet
```

Onglets de tests

Application de test



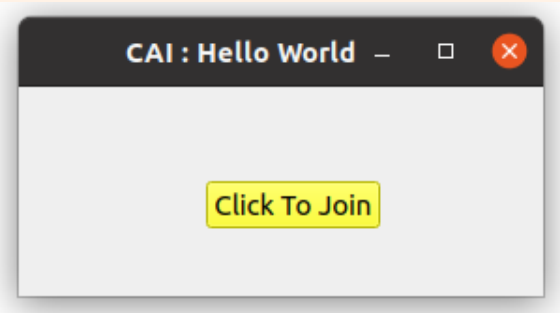
Onglets de tests

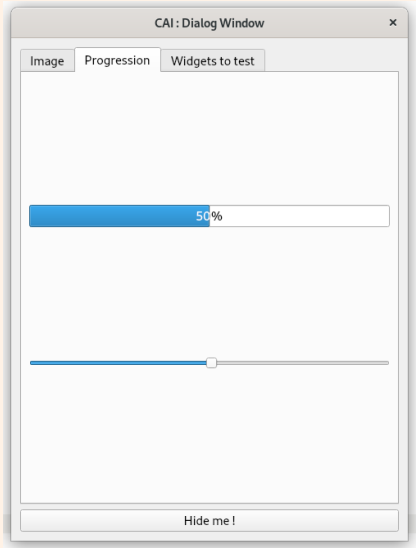
Toplevel.progressbar() : test de barre de progression

```
def progressbar(self) :
    value=50
    progress=QtWidgets.QProgressBar()
    progress.setValue(value)
    slider=QtWidgets.QSlider(QtCore.Qt.Horizontal)
    slider.setValue(value)
    slider.sliderMoved.connect(progress.setValue)
    onglet=QtWidgets.QWidget()
    vbox=QtWidgets.QVBoxLayout()
    vbox.addWidget(progress)
    vbox.addWidget(slider)
    onglet.setLayout(vbox)
    return onglet
```

Onglets de tests

Application de test





Onglets de tests

Toplevel.widget_to_test() : création de widgets

```
def widget_to_test(self) :
 widgets=[QtWidgets.QPushButton,QtWidgets.QCheckBox,
          QtWidgets.QRadioButton,QtWidgets.QComboBox,
          QtWidgets.QDateEdit,QtWidgets.QTimeEdit,
          QtWidgets.QDateTimeEdit,
          QtWidgets.QDial,
          QtWidgets.QDoubleSpinBox,
          QtWidgets.QFontComboBox,
          QtWidgets.QLCDNumber,
          QtWidgets.QLabel,QtWidgets.QLineEdit,
          QtWidgets.QProgressBar,
          QtWidgets.QSlider,QtWidgets.QSpinBox
         ]
```

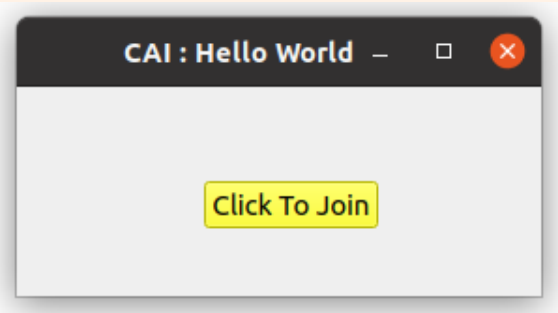
Onglets de tests

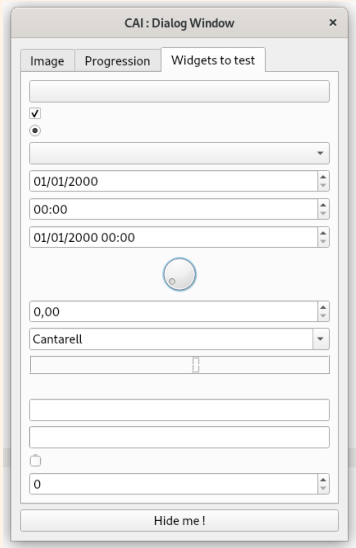
Toplevel.widget_to_test() : création de widgets

```
    onglet=QtWidgets.QWidget()
    layout = QtWidgets.QVBoxLayout()
    for w in widgets:
        layout.addWidget(w())
        value=50
    onglet.setLayout(layout)
    return onglet
```

Onglets de tests

Application de test





Bibliographie

Adresses "au Net"

- Qt : http://doc.qt.io
- pyQt : https://www.riverbankcomputing.com
- pySide : https://wiki.qt.io/PySide
- Martin Fitzpatrick : https://www.learnpyqt.com
- developpez.com : http://qt-devnet.developpez.com
- Thierry Vaira : http://tvaira.free.fr
- le CRD de l'ENIB : https://portail.enib.fr