Formation Python

Ihab ABADI / UTOPIOS

SOMMAIRE

Operating System Interface

- Afin d'interagir avec le système d'exploitation en python, on peut utiliser le module OS
- Ce module peut être utilisé avec la commande « import os »
- Les fonctions utiles pour aider à l'utilisation de ce module sont dir(os) qui renvoie une liste de toutes les fonctions du module et help(os) qui renvoie une page de manuel créée à partir des docstrings du module

Operating Systems Interface

- La gestion des fichiers et des répértoires peut se faire à l’aide du module shutil.
- >>> import shutil
- >>> shutil.copyfile('data.db', 'archive.db')
- >>> shutil.move('/build/executables', 'installdir')

File Wildcards

- Le module glob est une fonction utilisée pour créer des listes à partir de recherches génériques dans les répertoires :

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

Redirection de la sortie d'erreur et arrêt du programme

- Les attributs stdin, stdout et stderr font également partie du module "sys"
- Ceux-ci sont utiles pour afficher des avertissements et des messages d'erreur
- Le moyen le plus direct de terminer un script est d'utiliser sys.exit()
- >>> sys.stderr.write('Warning, log file not found starting a new one\n')

  Warning, log file not found starting a new one

String Pattern Matching

- Le module re fournit des outils d'expression régulière pour le traitement des chaînes.
- >>> import re
- >>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
- ['foot', 'fell', 'fastest']
- >>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
- 'cat in the hat'
- Les méthodes de chaîne sont plus faciles à lire et à déboguer, elles sont donc préférées lorsque seules des fonctionnalités simples sont nécessaires
- >>> 'tea for too'.replace('too', 'two')
- 'tea for two'

Command Line Arguments

- Le module sys permet de de récupérer les arguments passés lors de l'execution de notre programme.
- La récupération se fait à l'aide de l'attribut argv
- Les arguments sont récupérés dans une list
- >>> import sys
- >>> print sys.argv
- ['demo.py', 'one', 'two', 'three']

Mathematics

- Le module math donne accés aux fonctions de la bibliothèque math de C
- >>> import math
- >>> math.cos(math.pi / 4.0)
- 0.70710678118654757
- >>> math.log(1024, 2)
- 10.0

Mathematics

- Le module random permet de générer un nombre aléatoire
- >>> import random
- >>> random.choice(['apple', 'pear', 'banana'])
- 'apple'
- >>> random.sample(xrange(100), 10) # sampling without replacement
- [30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
- >>> random.random() # random float
- 0.1797098769370618 6
- >>> random.randrange(6) # random integer chosen from range(6)
- 4

Internet Access

- Deux des modules les plus simples pour accéder à Internet sont urllib2 et smtplib.
- Urllib2 est utilisé pour récupérer des données

```
>>> import urllib2
>>> for line in urllib2.urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):
... if 'EST' in line or 'EDT' in line: # look for Eastern Time ... print line <BR>Nov. 25, 09:43:32 PM EST
```

Internet Access

- Smtplib est utilisé pour envoyer des emails.

```
>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org', ... """To: jcaesar@example.org
... From: soothsayer@example.org
... ... Beware the Ides of March.
... """)
>>> server.quit()
```

Dates and Times

- Le module datetime fournit des classes pour manipuler les dates et les heures.
- Ce module prend en charge les objets avec le fuseau horaire
- >>> # dates are easily constructed and formatted
- >>> from datetime import date
- >>> now = date.today()
- >>> now
- datetime.date(2003, 12, 2)
- >>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
- '12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

- >>> # dates support calendar arithmetic
- >>> birthday = date(1964, 7, 31)
- >>> age = now - birthday
- >>> age.days
- 14368

Data Compression

- Les formats communs d'archivage et de compression des données sont directement supportés par les modules :
- zlib, gzip, bz2, fichier zip et fichier tar
- >>> import zlib
- >>> s = 'witch which has which witches wrist watch'
- >>> len(s)
- 41
- >>> t = zlib.compress(s)
- >>> len(t)
- 37
- >>> zlib.decompress(t)
- 'witch which has which witches wrist watch'
- >>> zlib.crc32(s)
- 226805979

Mesure de performance

- De nombreux utilisateurs souhaitent connaître les performances de différentes approches d'un même problème
- Le module timeit peut nous donner une vision rapide des performances de nos applications
- >>> from timeit import Timer
- >>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit() 0.5753528626024577
- >>> Timer('a,b = b,a', 'a=1; b=2').timeit() 0.5496253708577079 1

Module QA

- Afin de développer des logiciels de haute qualité, vous devez écrire des tests pour chaque fonction et les exécuter fréquemment au cours du processus de développement.
- Il existe une multitude de module pour nous aider à tester notre application telque doctest, unittest
- def average(values):
- """Computes the arithmetic mean of a list of numbers.
- >>> print average([20, 30, 70])
- 40.0
- """“
- return sum(values, 0.0) / len(values)
- import doctest
- doctest.testmod() # automatically validate the embedded tests

Module QA

- Le module unittest permet de mettre en place des tests unitaires dans un developpement en TDD
- import unittest
- class TestStatisticalFunctions(unittest.TestCase):
- def test_average(self):
- self.assertEqual(average([20, 30, 70]), 40.0)
  self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
  self.assertRaises(ZeroDivisionError, average, [])
  self.assertRaises(TypeError, average, 20, 30, 70)
- unittest.main() # Calling from the command line invokes all tests

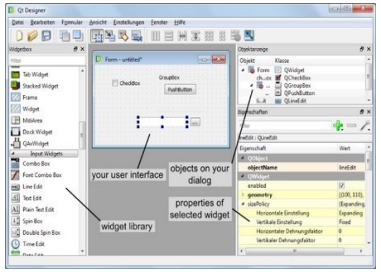
Ressource

- http://docs.python.org/tutorial/stdlib.html

Introduction

- **Qt** est une bibliothèque de classes offrant entre autres des composants d'interface graphique appelés **widgets**.

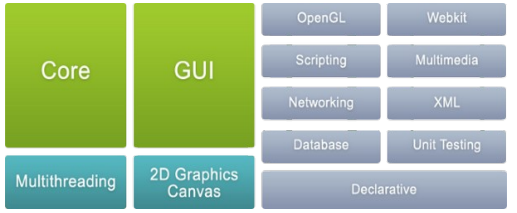
- Qt est **multi-plateformes** (portable) et **open-source** (licence GNU LGPL permettant son utilisation légale et gratuite par des logiciels propriétaires).



- Qt est initialement écrit en langage C++.

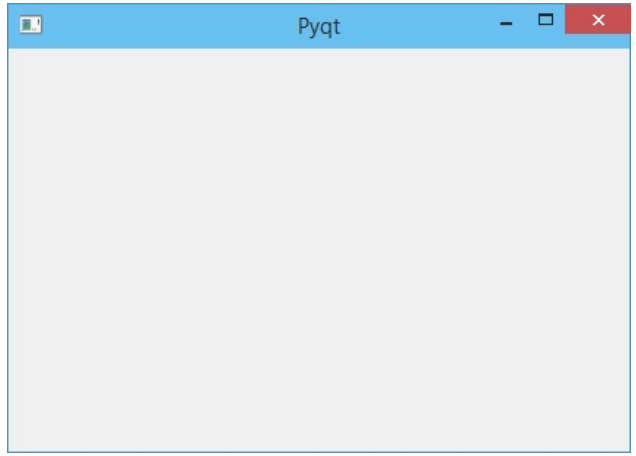
- **PyQt** est un *binding* de Qt pour le langage Python.
  - PyQt n'est pas gratuit pour une utilisation commerciale
  - Autre binding Python de Qt : PySide

- Alternatives à Qt :
  - Python : Tk (intégré au langage), wxWidgets (wxPython)
  - C++ : Gtk, wxWidgets, MFC (Microsoft), etc.



Première Application

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')

def main():
    app = QApplication(sys.argv)
    fenetre = MaFenetre()
    fenetre.show()
    app.exec()

if __name__ == '__main__':
    main()
```

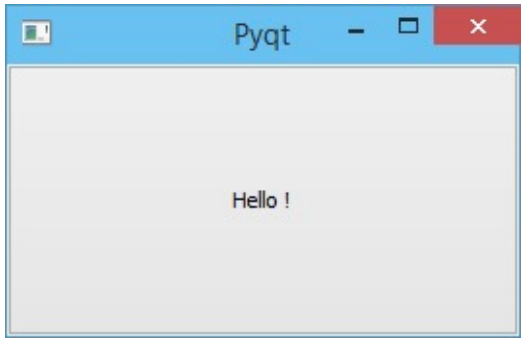


Layout de QMainWindow

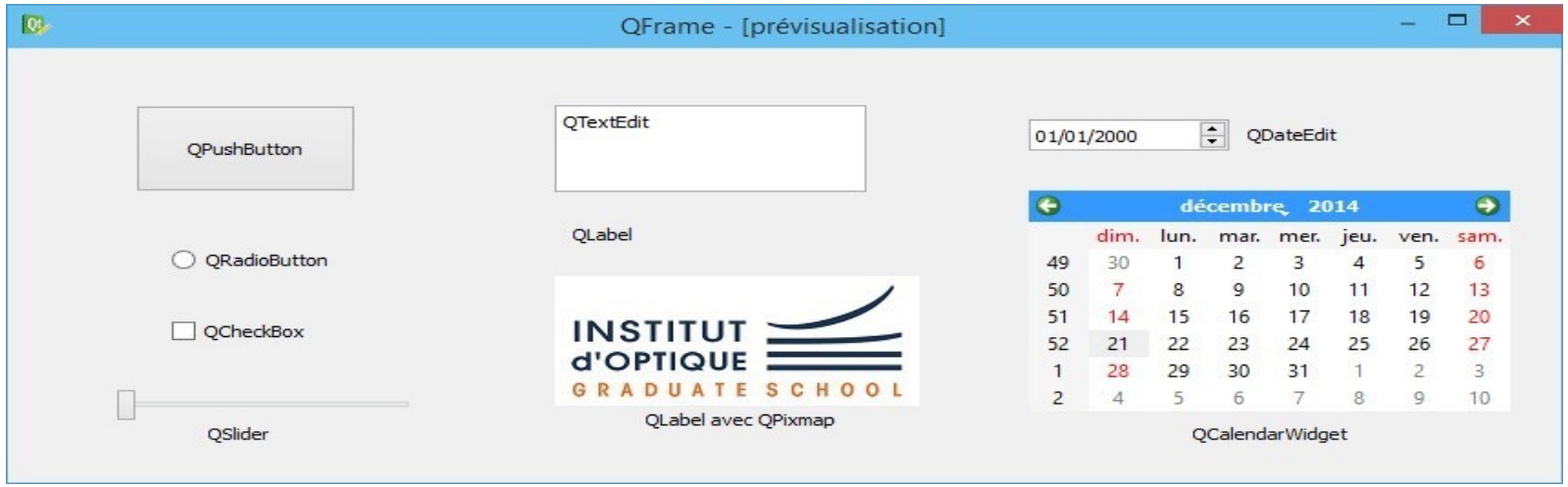
Central Widget

- Central Widget = un objet dérivant de QWidget
  - Soit un widget prédéfini
  - Soit un widget personnalisé, défini par une classe dérivant de QWidget

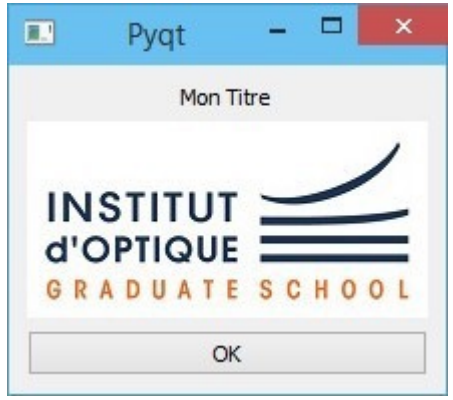
```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')

        button = QPushButton('Hello !')

        self.setCentralWidget(button)
```



Widgets prédéfinis




```
label = QLabel('Mon Titre', self)
label.setAlignment(Qt.AlignHCenter)
image = QLabel(self)
image.setPixmap(QPixmap('logo.jpg'))
bouton = QPushButton('OK', self)
```

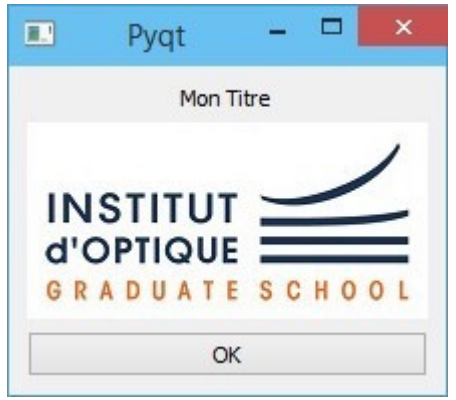


Widget personnalisé

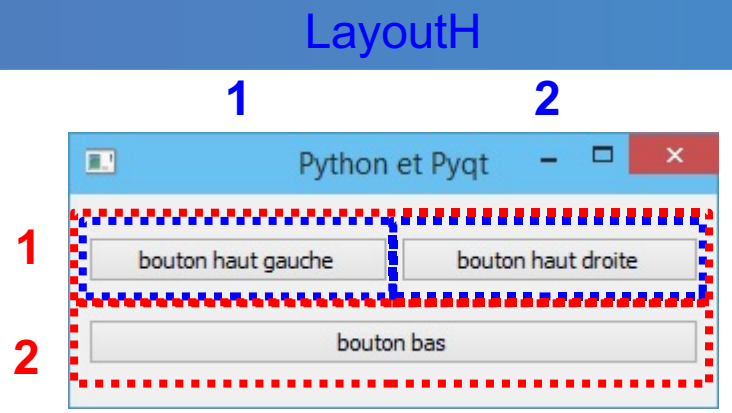
```
class MonWidget(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        label = QLabel('Mon Titre', self)
        label.setAlignment(Qt.AlignHCenter)
        label.setGeometry(10, 10, 200, 20)
        image = QLabel(self)
        image.setPixmap(QPixmap('logo.jpg'))
        image.setGeometry(10, 30, 200, 100)
        bouton = QPushButton('OK', self)
        bouton.setGeometry(10, 150, 200, 20)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')
        central_widget = MonWidget(self)
        self.setCentralWidget(central_widget)
        self.resize(250, 250)
```

Placement avec Layout

```
class MonWidget(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        layout = QVBoxLayout()
        label = QLabel('Mon Titre')
        label.setAlignment(Qt.AlignHCenter)
        image = QLabel()
        image.setPixmap(QPixmap('logo.jpg'))
        bouton = QPushButton('OK')
        layout.addWidget(label)
        layout.addWidget(image)
        layout.addWidget(bouton)
        self.setLayout(layout)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Pyqt')
        central_widget = MonWidget(self)
        self.setCentralWidget(central_widget)
```



Imbrication des Layouts

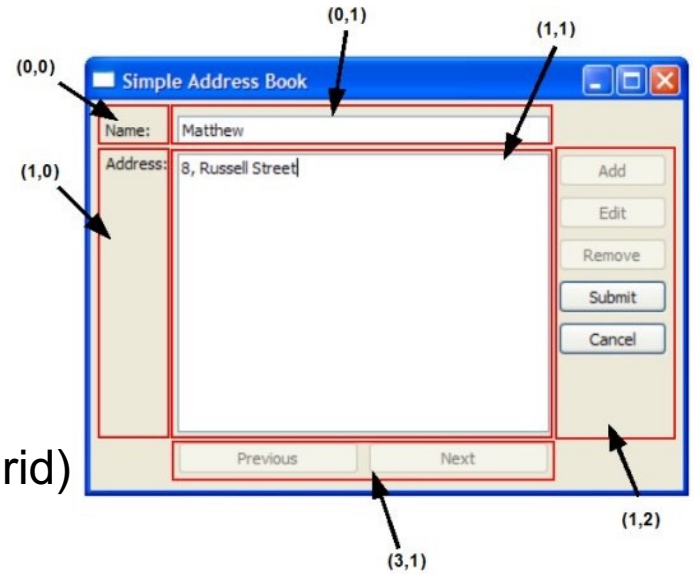



```
class MonWidget(QWidget):
    def__init__(self, parent):
        super().__init__(parent)
        bouton1 = QPushButton('bouton haut gauche')
        bouton2 = QPushButton('bouton haut droite')
        layoutH = QHBoxLayout()
        layoutH.addWidget(bouton1)
        layoutH.addWidget(bouton2)
        bouton3 = QPushButton('bouton bas')
        layoutV = QVBoxLayout()
        layoutV.addLayout(layoutH)
        layoutV.addWidget(bouton3)
        self.setLayout(layoutV)
```

Types de Layout

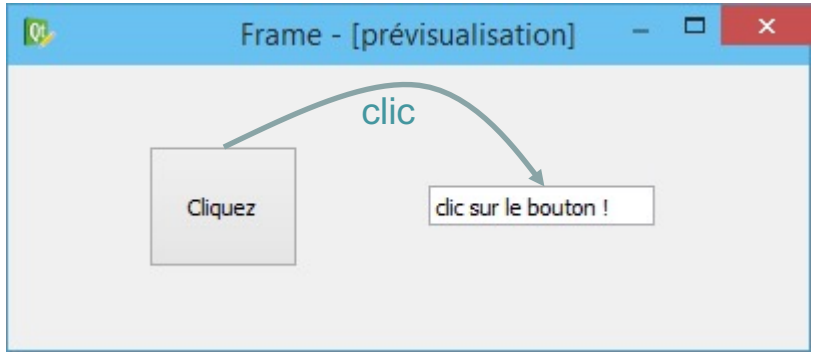
- QHBoxLayout
- QVBoxLayout
- QGridLayout
- QFormLayout

(grid)



(form)

Slot et Signal




```
class MonWidget(QWidget):

    def __init__(self):
        super().__init__()

        self.bouton = QPushButton('Cliquez', self)
        self.texte = QLineEdit(self)

        self.bouton.clicked.connect(self.on_bouton)
    …

    def on_bouton(self):
        self.texte.setText('clic sur le bouton !')
```

signal

slot (ou ‘callback’)

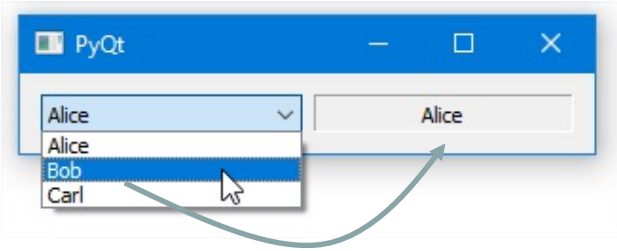
Slot et Signal

- Un signal peut être accompagné d'une information émise
  Ex: valueChanged(**int**) pour QSpinBox, QDial
  Ex: currentTextChanged(**str**) pour QCombBox
- Dans ce cas, la fonction callback doit avoir des arguments correspondant à ce qu'envoie le signal, pour réceptionner l'information émise :

```
class MonWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.label = QLabel()
        combobox = QComboBox()
        combobox.addItems(["Alice", "Bob", "Carl"])
        combobox.currentTextChanged.connect(self.afficher_nom)

    def afficher_nom(self, nom ):
        self.label.setText(nom)
```

émet un str

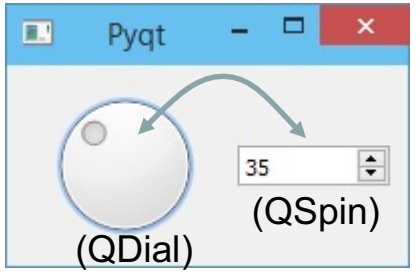
reçoit un str




```
dial = QDial(self)
spin = QSpinBox(self)
spin.valueChanged.connect(dial.setValue)
dial.valueChanged.connect(spin.setValue)
```

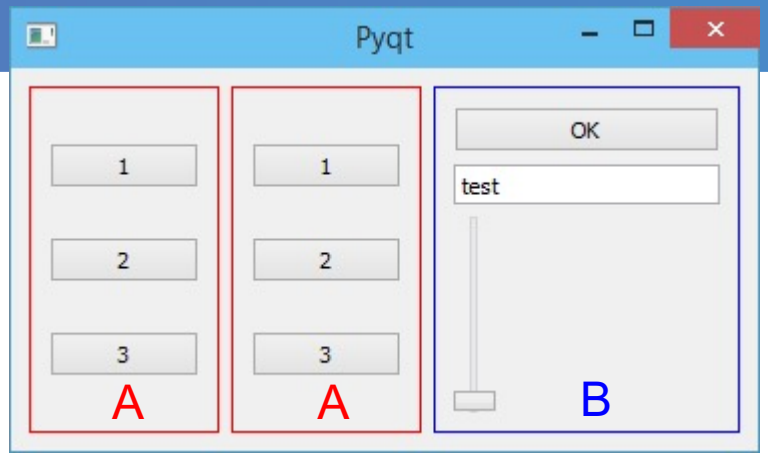
émet un int

reçoit un int



Fenêtre plus complexe

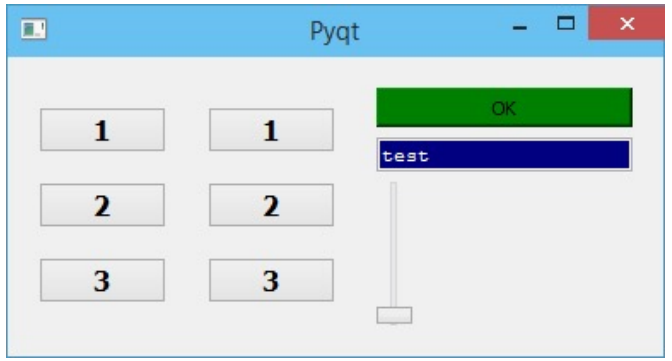
```
class MonWidgetA(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('1'))
        layout.addWidget(QPushButton('2'))
        layout.addWidget(QPushButton('3'))
        self.setLayout(layout)

class MonWidgetB(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('OK'))
        layout.addWidget(QLineEdit('test'))
        layout.addWidget(QSlider())
        self.setLayout(layout)
```




```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        widget1 = MonWidgetA(self)
        widget2 = MonWidgetA(self)
        widget3 = MonWidgetB(self)
        layout = QHBoxLayout()
        layout.addWidget(widget1)
        layout.addWidget(widget2)
        layout.addWidget(widget3)
        widget = QWidget()
        widget.setLayout(layout)
        self.setCentralWidget(widget)
```

Feuilles de style

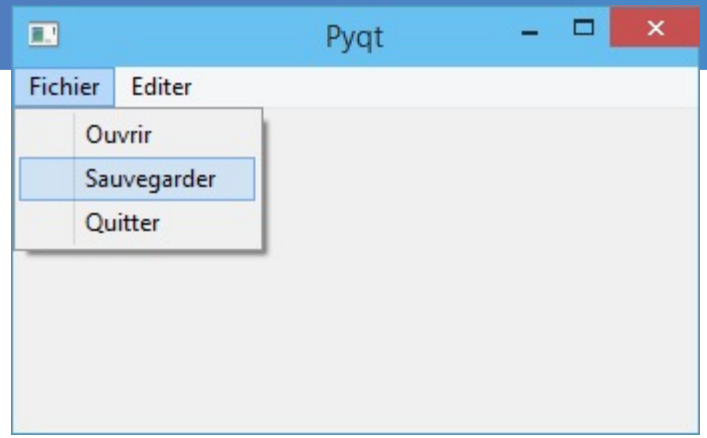
```
class MonWidgetA(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        self.setStyleSheet('QPushButton { font-weight: bold; font-size: 16px; }')
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('1'))
        layout.addWidget(QPushButton('2'))
        layout.addWidget(QPushButton('3'))
        self.setLayout(layout)
```




```
class MonWidgetB(QWidget):
    def __init__(self, parent):
        super().__init__(parent)
        self.setStyleSheet(' \
            QLineEdit { background-color: rgb(0,0,128); color: white; font-family: courier; }
            \ QPushButton { background-color: rgb(0,128,0); }')
        layout = QVBoxLayout()
        layout.addWidget(QPushButton('OK'))
        layout.addWidget(QLineEdit('test'))
        layout.addWidget(QSlider())
        self.setLayout(layout)
```

⇨ CSS3

Menu déroulant

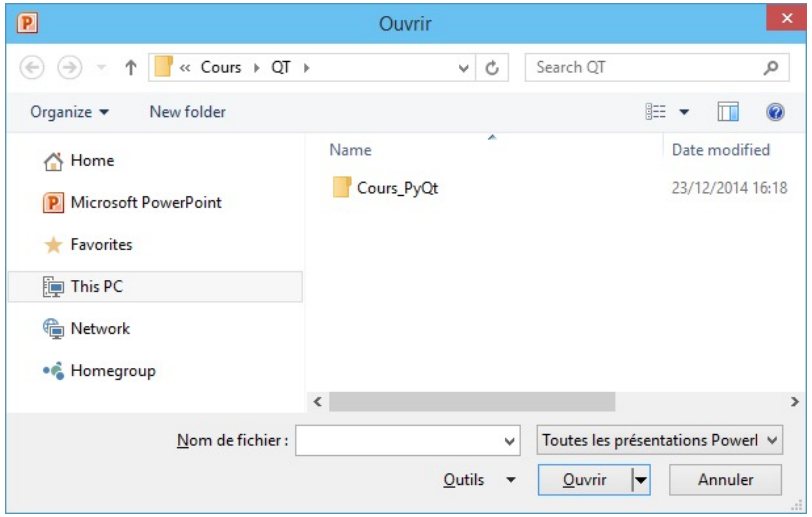


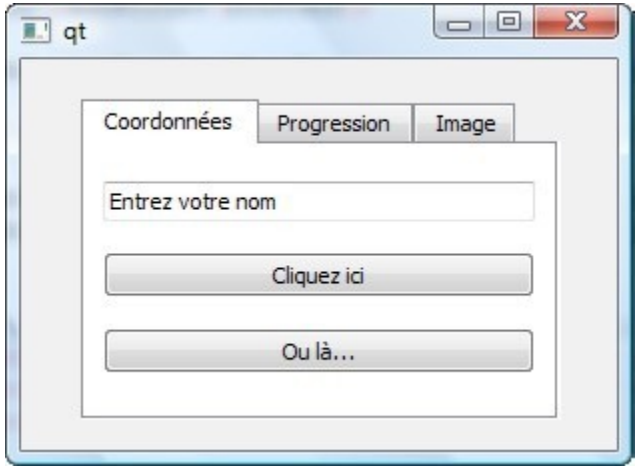

```
class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()

        menu1 = self.menuBar().addMenu('Fichier')

        action1 = QAction('Ouvrir', self)
        action1.triggered.connect(self.on_ouvrir)
        menu1.addAction(action1)

        action2 = QAction('Sauvegarder', self)
        action2.triggered.connect(self.on_sauver)
        menu1.addAction(action2)

        action3 = QAction('Quitter', self)
        action3.triggered.connect(self.on_quitter)
        menu1.addAction(action3)

        menu2 = self.menuBar().addMenu('Editer')
```

méthode de classe à définir

Quelques autres widgets de Qt

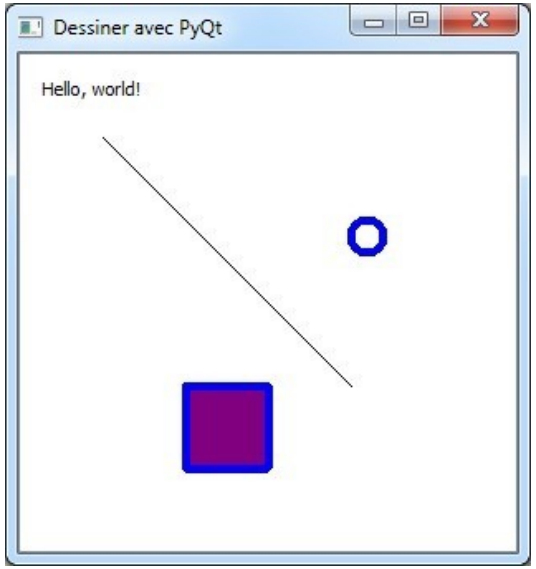
- Boites de dialogues usuelles
  - QMessageBox
  - QInputDialog
  - QFileDialog
- Widgets conteneurs
  - QGroupBox
  - QTabWidget
  - QStackedLayout
- Toolbar et Statusbar
  - QToolBar
  - QStatusBar
- Widgets complexes
  - QListView
  - QTreeView
  - QTableView





Dessiner

```
class MaScene(QGraphicsScene):
    """cette classe décrit la scène"""
    def __init__(self, parent):
        super().__init__(parent)
        self.setSceneRect(0, 0, 300, 300)
        texte = self.addText("Hello, world!")
        texte.setPos(10, 10)
        self.addLine(50, 50, 200, 200)
        stylo = QPen(Qt.blue, 5, Qt.SolidLine)
        self.addEllipse(200, 100, 20, 20, stylo)
        brosse = QBrush(QColor(128, 0, 128), Qt.SolidPattern)
        self.addRect(100, 200, 50, 50, stylo, brosse)

class MaVueGraphique(QGraphicsView):
    """cette classe fait le rendu (= dessin) de la scène"""
    def __init__(self, parent):
        super().__init__(parent)
        scene = MaScene(self)
        self.setScene(scene)

class MaFenetre(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Dessiner avec PyQt")
        vue = MaVueGraphique(self)
        self.setCentralWidget(vue)
```



Clavier et Souris

- Toute classe dérivée de QWidget peut redéfinir des fonctions héritées pour réagir aux événements clavier et souris, par exemple :
- keyPressEvent()
  - appelée lorsque qu'une touche du clavier est pressée.

```
def keyPressEvent(self, keyevent):
    if keyevent.key() == Qt.Key_Q:
        ...
```

- mousePressEvent()
  - appelée lorsqu'un bouton de la souris est cliqué.

```
def mousePressEvent(self, mouseevent):
    self.x = mouseevent.scenePos().x()
    self.y = mouseevent.scenePos().y()
```

Timer

- Un *timer* permet de déclencher l'appel d'une fonction à intervalles de temps réguliers.
- La classe Qt pour créer un timer est **QTimer**

```
def afficher():
    print("bonjour")

app = QApplication(sys.argv)
timer = QTimer()
timer.timeout.connect(afficher)
# répétition toutes les 1000 millisecondes
timer.start(1000)
app.exec()
```

```
class MonTimer(QTimer):
    def __init__(self):
        super().__init__()
        self.timeout.connect(self.ontimer)

    def ontimer(self):
        print("bonjour")

app = QApplication(sys.argv)
timer = MonTimer()
# répétition toutes les 1000 millisecondes
timer.start(1000)
app.exec()
```

`timer.stop()` permet d'arrêter le timer

Exercice

- 1re partie
  - Créer une classe (scène) affichant une image de carte.

    ```
    addPixmap(QPixmap("carte.png"))
    ```
  - La classe contient une fonction permettant de repositionner cette carte à l'endroit où on clique.
  - La classe contient une fonction permettant de créer une 2e carte quand on appuie sur la touche 'c'.
  - La classe contient une fonction permettant de déplacer la 2e carte quand on appuie sur les flèches du clavier.

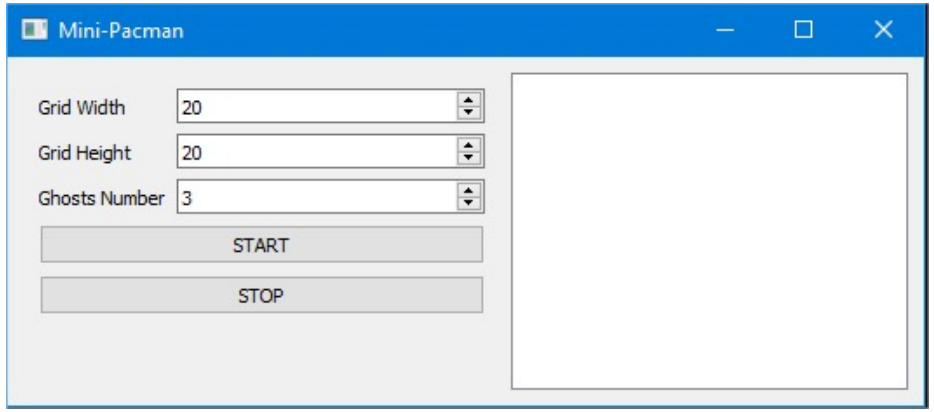
2e partie

- La classe contient un timer permettant d'animer la première carte, en modifiant sa position de quelques pixels à intervalles de temps réguliers.
- La classe contient une fonction permettant de mettre l'animation en pause quand on appuie sur la touche 'p'.

Exercice : mini Pacman

- Partir des modules model.py, controler.py et view.py fournis sur learnpython.ovh

- Module view.py:
  - coder le constructeur de la classe PacmanParams de manière à obtenir l'interface ci-dessous
    - Utiliser QSpinBox pour saisir les dimensions du plateau de jeu et le nombre de fantômes
    - Utiliser QFormLayout pour titrer et disposer les 3 options
    - connecter les méthodes on_start et on_stop aux boutons start et stop. Ces méthodes ne font rien pour le moment.

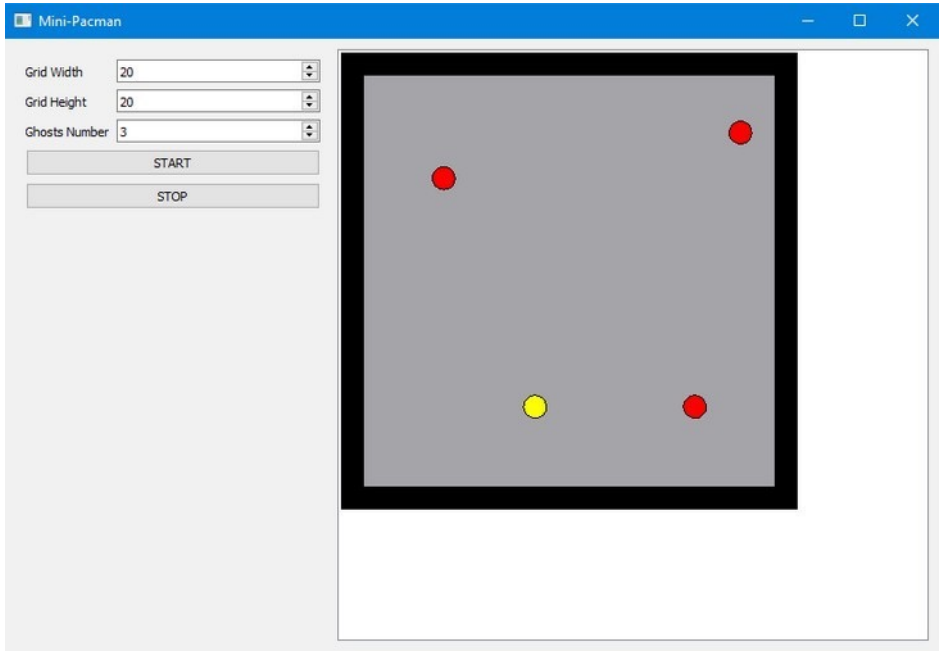


Exercice : mini Pacman

- Module controler.py
  - Coder le constructeur et la méthode start() de PacmanControler
    - Attributs de la classe PacmanControler :
      - self.ghosts : liste d'objets fantômes (Entity)
      - self.pacman : objet pacman (Pacman)
      - self.timer : timer
      - self.width : largeur du plateau
      - self.height : hauteur du plateau
    - Dans le constructeur :
      - Créer l'attribut timer (classe QTimer) et le connecter à la méthode next()
    - Dans start():
      - Créer nb_ghosts objets fantômes et leur donner une position initiale aléatoire
      - Créer un objet pacman et lui donner une position initiale aléatoire
      - Démarrer le timer
    - Dans stop()
      - Arrêter le timer
    - Appeler les méthodes start() et stop() du contrôleur au moment adéquat dans la classe PacmanParams (view.py). Bien prendre en compte les paramètres saisis dans l'interface (largeur, hauteur, nombre de fantômes). Vérifier que ces paramètres parviennent avec la bonne valeur à la méthode start().
    - Vérifier que la méthode next() est bien appelée à intervalle de temps régulier lorsqu'on clique sur le bouton Start (mettre un print dans la méthode next()). Vérifier que cliquer sur le bouton Stop arrête bien le timer.

Exercice : mini Pacman

- Module view.py:
  - Classe PacmanScene :
    - Constructeur :
      - Initialiser la scène à une dimension de 512 x 512 pixels
    - Méthode refresh() :
      - Dessiner le plateau de jeu en allouant 25x25 pixels à chaque case du plateau de jeu
        - » Dessiner le fond en gris et les contours du plateau en noir (rectangles)
        - » Dessiner chaque fantôme (disque de couleur rouge)
        - » Dessiner Pacman (disque de couleur jaune)
- Module controler.py
  - Compléter la méthode next() :
    - Déplacer les fantômes
    - Déplacer Pacman
    - Si un fantôme est "mangé", le supprimer de la liste des fantômes
    - S'il n'y a plus de fantôme, arrêter le timer
    - Provoquer un rafraichissement de tous les "clients" du contrôleur
- Tester
  - Vérifier le bon déplacement des entités
  - Vérifier la disparition des fantômes mangés
  - Vérifier que la fin de partie se passe bien (Pacman s'arrête)
  - Vérifier qu'on peut démarrer une nouvelle partie, en modifiant ou pas les paramètres du jeu



Exercice : mini Pacman

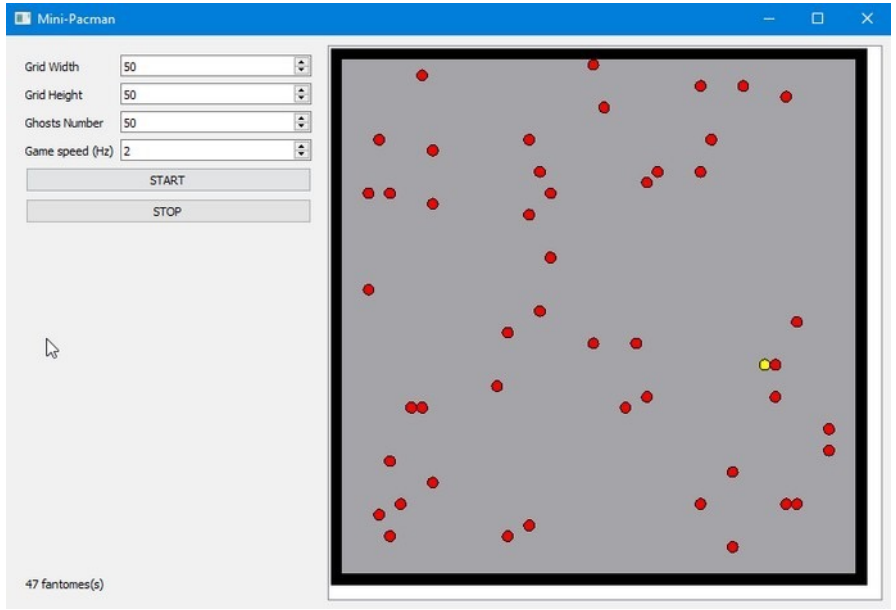
- Module view.py:

  - Téléportation de Pacman à la souris :
    - Ajouter une méthode move_pacman(x, y) au contrôleur qui téléporte instantanément Pacman à la position (x, y). Ne pas permettre de téléporter Pacman en dehors du plateau de jeu ou sur la bordure.
    - Coder la méthode mousePressEvent pour que Pacman soit téléporté sur le curseur de souris lorsqu'on clique sur le plateau de jeu.

  - Mettre le jeu en pause au clavier (touche P)
    - Implémenter la méthode keyPressEvent() (module view.py) pour transmettre la touche du clavier saisie à la méthode process_keypress() du contrôleur
    - Implémenter la méthode process_keypress() (module controler.py) pour activer / désactiver la pause par la touche P. La mise en pause consiste à stopper le timer. Redémarrer le timer pour enlever la pause, sans réinitialiser le jeu.
    - Vérifier le bon comportement si le jeu est relancé (bouton start) pendant la pause.

Exercice : mini Pacman

- Adapter automatiquement la dimension en pixels d'une case de plateau (actuellement fixé à 25x25 pixels) pour que la plateau occupent toujours 512 pixels quel que soit la largeur et la hauteur choisies
- Lorsqu'il n'y a plus de fantômes, afficher "Game Over" par-dessus le plateau de jeu (jusqu'au démarrage d'une nouvelle partie avec start)
- Afficher le nombre de fantômes encore en vie dans la barre des paramètres
- Ajouter un paramètre permettant de modifier la vitesse du jeu pendant une partie (ajouter un widget QSpinBox dans les paramètres, utiliser la méthode setInterval du timer).



Extension de Python en C

Quand est ce que on utilise l'interface de Python en C ?

- Pour étendre les fonctionnalités de Python
- Pour améliorer les performances
- Pour utiliser Python comme langage de collage
- Pour créer des liaisons Python pour une bibliothèque

Exemple 1 C vs Python

```c
#include <stdio.h>

int main(int argc, char **argv){

  int i, j, total;
  double avg;
  total = 10000000;
  for (i = 0; i < 10; i++){
        avg = 0;
        for (j = 0; j < total; j++){
          avg += j;
          }
        avg = avg/total;
        }
  printf("Average is %f\n", avg);
  return 0;
  }
```

Compile and execute gcc -O3 add numbers.c -o add numbers.x time ./add numbers.x

add_numbers.py

```python
total = 10000000

for i in xrange(10):
  avg = 0.0
  for j in xrange(total):
        avg += j
  avg = avg/total

print "Average is {0}".format(avg)
```

Execute the Python script

time ./add numbers.py

Example 1: Avec Numpy

add_numbers_np.c

```
from numpy import mean, arange

total = 10000000

a = arange(total)

for i in xrange(10):
        avg = mean(a)
print "Average is {0}".format(avg)
```


Program	time
Python:	20.17s
C:	0.09s
Numpy:	0.17s

Interaction de Python avec d'autre langages

Python peut interagir avec d'autre langages:
https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages

Pour l'interaction avec du C on peut utiliser

- Python C API
- Ctypes
- Cython
- Boost
- Swig
- pybind11

Ctypes

Ctypes est une bibliothèque de fonctions étrangères pour Python.

Il fournit des types de données compatibles C et permet d'appeler des fonctions dans des DLL ou des bibliothèques partagées.

Interaction de Python avec d'autre langages

ctypes type	C type	Python type
c_bool	_Bool	bool (1)
c_char	char	1-character string
c_wchar	wchar_t	1-character unicode string
c_byte	char	int/long
c_ubyte	unsigned char	int/long
c_short	short	int/long
c_ushort	unsigned short	int/long
c_int	int	int/long
c_uint	unsigned int	int/long
c_long	long	int/long
c_ulong	unsigned long	int/long
c_longlong	__int64 or long long	int/long
c_ulonglong	unsigned __int64 or unsigned long long	int/long
c_float	float	float
c_double	double	float
c_longdouble	long double	float
c_char_p	char * (NUL terminated)	string or None
c_wchar_p	wchar_t * (NUL terminated)	unicode or None
c_void_p	void *	int/long or None

Exemple 2 Librairy

example2/add.c

```
float add_float(float a,float b){return a + b;}

int add_int(int a,int b){ return a + b;}

int add_float_ref(float *a,float *b,float *c){
  *c = *a + *b;
  return 0;
  }
```

example2/arrays.c

```
int add_int_array(int *a,int *b,int *c, int n){
  int i;
  for (i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
    }
  return 0;
  }
float dot_product(float *a, float *b, int n) {
  float res=0;
  int i;
  for (i = 0; i < n; i++) {
    res = res + a[i] * b[i];
    }
  return res;
  }
```

Compile and create the library

$ gcc -fPIC -c add.c
$ gcc -fPIC -c arrays.c
$ gcc -shared add.o arrays.o -o

libmymath.so

```
import ctypes
math= ctypes.CDLL("libmymath.so")
math.add_int(4,5)
```

Exemple 2 Library

Avec :

```
math.add_float(4,5)
math.add_float(4.0,5.0)
```

Une exception sera générer

On doit spécifier le types des paramètres:

```
math.add_float.restype=ctypes.c_float
math.add_float(ctypes.c_float(4.0),ctypes.c_float(5.0))
```

```
math.add_float.restype=ctypes.c_float
math.add_float.argtypes= [ctypes.c_float,ctypes.c_float]
math.add_float(4.0,5.0)
```

Exemple 2 Library

Utilisation des références comme paramètres

```
a=ctypes.c_float(5)
b=ctypes.c_float(5)
res=ctypes.c_float()

math.add_float_ref(ctypes.byref(a),ctypes.byref(b), ctypes.byref(res))
res.value
```

Utilisation des tableaux avec Ctypes

```
a=(ctypes.c_int * 3) (-1, 2, 5)
b=(ctypes.c_int * 3) (-1, 3, 3)
res=(ctypes.c_int * 3) (0, 0, 0)
n=ctypes.c_int(3)

math.add_int_array(a,b, res,n)

res[0], res[1], res[2]
```

Utilisation des tableaux avec numpy

```
import numpy as np
a=np.array([1,2,-5], dtype=ctypes.c_int)
b=np.array([-1,3,3], dtype=ctypes.c_int)
res= np.zeros(3, dtype=ctypes.c_int)
n=ctypes.c_int(3)
intp=ctypes.POINTER(ctypes.c_int)

i=a.ctypes.data_as(intp)
j=b.ctypes.data_as(intp)
k=res.ctypes.data_as(intp)
math.add_int_array(i,j,k,n)

res
```

Structures

example2/rectangle.c

```
typedef struct _rect {
  float height, width;
  } Rectangle;

float area(Rectangle rect){
  return rect.height*rect.width
  }
```

```
gcc -fPIC -c rectangle.c
gcc -shared rectangle.o -o libgeom.so
```

```
from geometry import *

r= Rectangle(3,4)

r.area()
r.width=10
r.area()
```

example2/geometry.py

```
import ctypes as C
clib = C.CDLL('./libgeom.so')
clib.area.argtypes=[C.Structure]
clib.area.restype=C.c_float

class Rectangle(C.Structure):
  _fields_=[
          ("width",C.c_float),
          ("height",C.c_float)
          ]

  def __init__(self,width,height):
    self.width = width
    self.height = height

  def area(self):
    return clib.area(self)
```