Université de la Rochelle - Licence 3 Informatique
Programmation Évènementielle

TP 1 - Une application complète en PyQt

© B. Besserer, R. Péteri, S. Bourbia

Année universitaire 2025-2026

Avant-propos

Chaque TP comportera des points de validation, représentés par des marques dans les marges, comme ici →. ☑
Pour valider chaque point, vous devrez appeler l'enseignant pour lui montrer votre code et faire une démonstration de la fonctionnalité demandée.
Les difficultés de mise en oeuvre dues à une configuration imparfaite ou incomplète de votre environnement de développement ne seront pas excusés, le TEA étant destiné préalablement à configurer correctement votre environnement de travail (PyCharm ou Visual Studio Code).

L'utilisation d'un assistant IA (Copilot) ou de ChatGPT est très fortement déconseillée (voir cours). Nous testerons votre bonne compréhension du code produit lors des points de validation.

1 Un premier programme console (durée estimé : 30min)

Récupérez sous Moodle le fichier Syracuse_bug.py, qui doit calcule la suite de Syracuse (ou suite de Collatz) pour le nombre N saisi. **Ce fichier comporte des erreurs et bugs**. Vous devez le corriger...
Les termes de la suite de Syracuse suivent la règle suivante : On part d'un nombre entier plus grand que zéro ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1.

- Corriger le cas échéant l'indentation du code (utilisez les fonctionnalités de l'éditeur).
- Corriger l'erreur de runtime si lors de l'exécution pour n=5, vous vous égarez dans une boucle infinie. Si cela arrive il faut choisir arrêter le débogage dans le menu déboguer (Maj+F5). Pour trouver l'erreur, vous pouvez mettre un point d'arrêt et exécuter le code en mode pas-à-pas (F10).
- Prouver l'hypothèse selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint toujours la valeur 1. Bon, c'est une plaisanterie ... Aucun mathématicien n'a encore pu prouver cela !

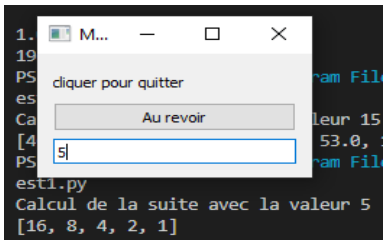
Lorsque les valeurs s'affichent correctement, **ré-écrivez le calcul de la suite sous forme de fonction** : `GenerateSyracuseSequence(n)` tel que les valeurs soient enregistrés dans une liste en tant que **valeurs entières**, et cette liste sera la valeur renvoyée par la fonction.
En effectuant `values=GenerateSyracuseSequence(n)` suivi de `print(values)`, les termes de la suite doivent s'afficher dans la console (il ne faut plus laisser de `print()` dans la fonction...)

2 Apparition des premières fenêtres.... (durée estimée : 1h)

Utilisez maintenant le fichier TP1.py et exécutez ce script [1]
Modifier ce code en y ajoutant votre fonction `GenerateSyracuseSequence(n)` tel que cette fenêtre serve de **fenêtre de saisie** pour la valeur initiale du calcul de la suite de Syracuse :

1. Attention, il est possible que la fenêtre graphique apparaisse à l'arrière plan, derrière des fenêtres déjà ouvertes...

- Ajouter un champ de texte à la fenêtre (de type QLineEdit)
- Mettre du code à la fin, **qui ne sera bien sur exécuté que lorsqu'on quittera l'interface graphique**, et qui va récupérer la valeur qui aura été saisi dans le widget QLineEdit en utilisant la méthode `text()`
- Ré-affichez cette valeur dans la console, lancez le calcul des termes de la suite et affichez le résultat dans la console (avec des `print()`)



2.1 Transformation du programme déclaratif / procédural en POO

Mais pour l'instant il n'est pas possible de calculer la suite pour deux valeurs différentes sans relancer le programme ! On veut maintenant exécuter le code de calcul lorsque l'utilisateur effectue une action sur un widget (par ex. un clic sur un bouton). Il est alors plus commode d'abandonner l'écriture séquentielle et déclarative de code et de passer à une écriture orienté objet.
Les deux code ci-dessous sont équivalent :

```
if __name__ == '__main__':
  app = QApplication(sys.argv)

  w = QWidget()
  w.resize(500, 300)
  w.move(300,300)
  title = "DUPONT  "
  w.setWindowTitle(title)

  label = QLabel("cliquer pour quitter")
  bu = QPushButton("Go...")

  bu.clicked.connect(QApplication.instance().quit)

  layout = QVBoxLayout(w)
  layout.addWidget(label)
  layout.addWidget(bu)
  w.setLayout(layout)

  w.show()

  app.exec_()
```

```
class MyMainWindow(QWidget):

  def __init__(self, parent=None):
    QWidget.__init__(self, parent=parent)
    self.initUI()

  def initUI(self):
    self.setGeometry(300,300,500,300)
    title = "DUPONT  "
    self.setWindowTitle(title)

    label = QLabel("cliquer pour quitter")
    self.bu = QPushButton("Go...")
    self.bu.clicked.connect(QApplication.instance().quit)

    layout = QVBoxLayout(self)
    layout.addWidget(label)
    layout.addWidget(self.bu)
    self.setLayout(layout)

if __name__ == '__main__':
  app = QApplication(sys.argv)
  w = MyMainWindow()
  w.show()
  app.exec_()
```

Vous devez donc transformer le code que vous avez écrit jusqu'à présent (uniquement la partie gèrant l'interface graphique) dans sa version objet.

Lorsque tout est OK dans la version objet, alors ajoutez une méthode `DisplayMessage()` qui va écrire quelque chose dans un widget texte (il faut créer le widget : un champ de texte multiligne - voir méthode `setPlainText()` `avec QTextEdit  ou .setWordWrap(...) avec un QLabel`)

Cela s'écrit comme une méthode de la classe :

```
    def DisplayMessage(self):
        self.te.setText("Hello")    # te est l'instance de l'objet texte multiligne
```

Et l'action sur le bouton bu sera donc lié à cette méthode : `self.bu.clicked.connect(self.DisplayMessage)`

Testez le fonctionnement, puis modifier le code afin que ce soit le calcul de la Suite de Syracuse qui soit lancé lorsqu'on clique sur le bouton.

☑ 1 `Pour valider: montrez votre code qui fonctionne sous sa forme procédurale`

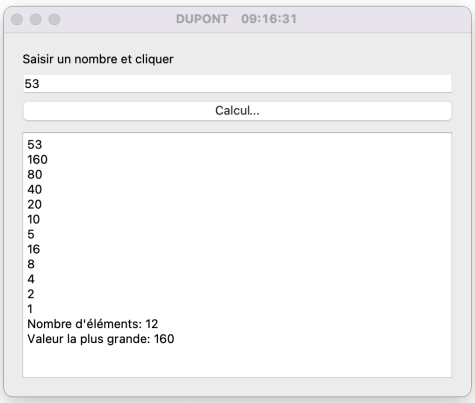
2.2 La fenêtre principale de votre application

On va complément abandonner le mode "console" pour toutes les saisies ; tout s'effectue dans une fenêtre principale. Complétez votre programme (essentiellement le constructeur) pour que le fenêtre principale comporte :
- Un label
- Un champ de saisie de texte sur une ligne
- Un bouton
- Un champ de texte multiligne.

Vérifiez que tout cela apparaît, puis intégrer le calcul de la suite de Syracuse, qui doit se déclencher lors de l'appui sur le bouton en utilisant la valeur qui sera saisie dans le champ de texte "1 ligne" et la liste des valeurs de la suite

de Syracuse doit apparaître dans le dernier champ de texte (multiligne).
Affichez à la suite le nombre d'éléments de la liste obtenue lors du calcul (facile : `len(values)`), et encore la valeur maximum rencontrée (pas besoins d'écrire une boucle, c'est tout aussi simple `max(values)`)
Assurez vous que vous pouvez effectuer un calcul plusieurs fois de suite. Testez votre interface en saisissant des valeurs aberrantes comme des nombres négatifs, des lettres, un champ de saisie vide, etc... En cherchant bien, quelques attributs des widgets ou méthodes du framework Qt permettent de sécuriser votre interface tout en limitant l'écriture de code (chercher du coté de `QLineEdit::setValidator()` par exemple).
Enfin, modifiez votre code afin de bien respecter le format du texte affiché dans la barre de titre : Votre nom en majuscule et l'heure.

```
import socket
from datetime import datetime
...
title = "DUPONT  " + datetime.now().strftime("  %H:%M:%S")
self.setWindowTitle(title)
```



`Pour valider: reproduisez l'affichage ci-contre.` ☑ 2

3 Affichage graphique de la suite de Syracuse (durée estimée : 1h30)

En connaissant le nombre de termes dans la suite et la valeur la plus élevée, on peut effectuer une mise à l'échelle et effectuer un tracé graphique de la suite de Syracuse. Il faut donc créer un nouveau widget en dessous de la zone de texte d'affichage. Ce nouveau widget sera de type QLabel, de taille fixe (pour l'instant) et avec une bordure pour la visualiser. Vous créerez aussi une zone graphique en mémoire (non affichée) de même taille, de type QPixmap.

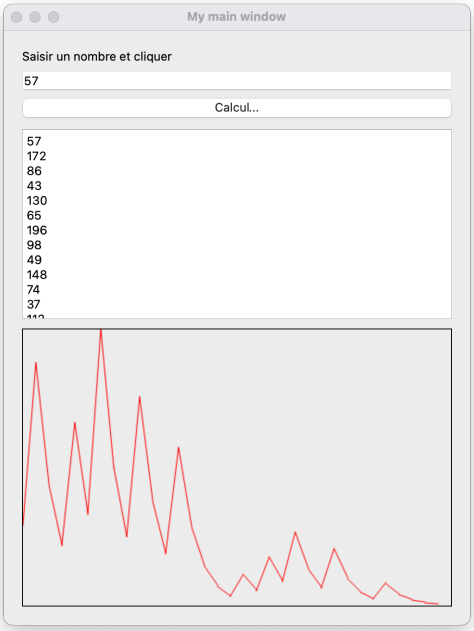
```
self.lb = QLabel()
self.lb.setFixedWidth(300)
self.lb.setFixedHeight(300)
self.lb.setFrameShape(QFrame.Panel)    # pour dessiner un contour autour de la zone
self.px = QPixmap(300,300)
```

La taille en x et y du QLabel étant connu, ainsi que le nombre de terme de la suite et la hauteur maximale, determier un pas en x (nommé step) et un coefficient en y pour le tracé (tel que la plus elevée des valeurs corresponde à la hauteur de la zone graphique, simple règle de 3).
Faire le tracé. Celui-ci devra avoir lieu immédiatement après le calcul de la suite. Le code de base est le suivant :

```
self.px.fill(QColor(200, 200, 200))    # RAZ de la zone graphique, avec une couleur RGB de votre choix
painter = QPainter(self.px)            # récupération de l'objet QPainter de la pixmap
for i in .... :                        # parcours dans les termes de la suite
     h = .....                         # hauteur du point ou du trait... calcul de la mise à l'échelle
     painter.drawPoint(x,y)            # tracé d'un point
     painter.drawLine(x1,y1,x2,Y2)     # ou bien tracé d'un trait
self.lb.setPixmap(self.px)             # la zone graphique est affecté au widget label
```

Pour valider:

- Vérifiez avec la valeur **57** que vos obtenez bien la copie d'écran suivante. ☑ 3
- Assurez-vous que la mise à l'échelle fonctionne (votre tracé doit prendre toute la hauteur et la largeur du `QLabel`). ☑ 4



3.1 Message box pour quitter

Affichez une boite de dialogue comme ci-dessus affichant un message du type “souhaitez vous vraiment quitter ?” avec des boutons oui/non lorsque l’utilisateur clique sur la case de fermeture de la fenêtre principale.

- Il faut implémenter du code en réaction à l’événement closeEvent. On va donc surcharger la méthode `closeEvent(self, event)` : de la fenêtre principale. Écrire cette surcharge, qui devra créer une boite de dialogue et traiter la réponse de celle-ci.
- A savoir : La plupart des boites de dialogue standards sont prédéfinies dans Qt. La classe se nomme QMessageBox, et une méthode de cette classe nommée `QMessageBox.question()` permet d’afficher cette boite de dialogue en récupérant la réponse de l’utilisateur. Évidemment cette méthode nécessite plusieurs paramètres (texte, nombres de boutons, bouton par defaut, ...), à vous de trouver.
- Récupérez la réponse de la QMessageBox et faites un test : si l’utilisateur décide que quitter, transmettre l’événement vers la méthode de la classe de base avec `event.accept()`, sinon `event.ignore()`

☑ 5 Appelez l’intervenant pour validez, en lui montrant la boite de dialogue pour quitter.