1

Présentation de PyQt

PyQt est la contraction de deux mots : d'un côté, Python (le langage de programmation utilisé) réputé fort simple d'apprentissage ; de l'autre, Qt, un cadriciel extrêmement complet (principalement pour des interfaces graphiques), mais écrit en C++. PyQt sert de couche de liaison entre ces deux mondes et apporte Qt à l'environnement Python.

Qt est une bibliothèque multiplateforme, reconnue avant tout pour ses fonctionnalités d'aide à la conception d'interfaces graphiques. Cependant, Qt peut faire beaucoup plus : cette bibliothèque vient avec des modules pour l'accès aux bases de données SQL, un navigateur web complet réutilisable, un système d'aide, des fonctionnalités multimédia. Depuis quelque temps, elle propose de nouvelles fonctionnalités plus intégrées et de plus haut niveau, comme l'accès à des outils de cartographie et de localisation, à la communication sans fil (NFC, Bluetooth), à des graphiques et de la visualisation de données, etc. Également, son environnement est très riche, avec de nombreuses autres bibliothèques d'extension disponibles (voir la Section 1.2, Environnement de PyQt).

Son principal point fort est de s'adapter aux nouvelles utilisations de l'informatique : il est l'une des très rares bibliothèques d'interfaces graphiques généralistes à s'être implantée dans le domaine des applications mobiles (il peut s'exécuter sur Android, iOS et Windows Phone) et à proposer une hybridation avec des applications web. Un autre point à souligner est Qt Quick, une technologie déclarative de développement d'interfaces : au lieu d'écrire du code pour effectuer le lien entre deux parties d'une interface graphique (un bouton et le texte affiché, par exemple), il suffit de *déclarer* qu'il existe une relation entre les deux ; ce paradigme est détaillé dans Développement d'une application avec Qt Quick.

Note › *Qt dispose également d'une série d'autres modules, nettement plus utiles en C++ qu'en Python, comme des chaînes de caractères évoluées, avec des expressions régulières, ou encore l'accès au réseau et à Internet. Bon nombre de ces fonctionnalités sont disponibles de longue date en Python, mais elles ne sont arrivées que très récemment dans l'environnement standard C++.*

1.1. PyQt et les autres bibliothèques de développement d'interfaces graphiques

PyQt n'est pas la seule manière de réaliser des interfaces graphiques. En réalité, l'environnement Python ne manque pas de choix : notamment, Tkinter (construit par-dessus Tk), qui a l'avantage d'être livré par défaut avec Python ; cependant, il possède peu de composants graphiques de base (boutons, zones de texte, etc.). En outre, vous devrez recourir à des extensions pour disposer de composants graphiques très utilisés, comme des boîtes de dialogue (pour afficher une information à l'utilisateur ou lui demander de sélectionner un fichier), ainsi que des fonctionnalités comme le glisser-déposer. Beaucoup de développeurs lui reprochent une mauvaise intégration avec l'environnement de bureau, un souci qui a toutefois été corrigé avec les années.

L'autre grand concurrent est wxPython, qui à nouveau correspond à une couche de liaison vers une bibliothèque C++ (wxWidgets). Comme PyQt, il doit être installé séparément de Python et est livré avec quantité de composants graphiques (à la différence de Tkinter). Cependant, le développement de la branche actuelle est très lent (pas de nouvelle version depuis 2014) et wxPython n'est toujours pas compatible avec Python 3 (contrairement aux deux autres ; pourtant, Python 3.0 est sorti en 2008). Le projet Phoenix a un développement actif et remédie à ces problèmes, mais il n'est pas encore utilisable par le grand public.

Alors, quelle bibliothèque utiliser ? Toutes trois sont très matures, elles existent déjà depuis un certain temps et sont disponibles gratuitement ; chacune a sa propre communauté, qui propose son aide sur les forums et listes de diffusion. Tkinter est souvent dite pour être la moins intuitive des trois, PyQt est la seule à proposer une approche déclarative (détaillée dans Développement d'une application avec Qt Quick).

Ces aspects techniques ne sont pas les seuls importants, l'environnement de chaque bibliothèque importe au moins autant. PyQt dispose d'une documentation de qualité et celle de Qt reste exploitable en Python. Aussi bien wxPython que PyQt disposent d'un éditeur visuel d'interfaces (respectivement, wxGlade et Qt Designer).

Globalement, PyQt dispose d'un bon nombre d'avantages par rapport à ses concurrents, tant au niveau technique pur que dans l'environnement : après les avoir essayés, beaucoup de personnes (dont les auteurs) préfèrent généralement travailler avec PyQt plutôt que wxPython ou Tkinter.

En ce qui concerne les licences, Tkinter est distribué sous la même licence que Python, wxPython sous la licence wxWidgets (similaire à la LGPL) et PyQt sous la licence GPL ou une licence commerciale (payante). En pratique, pour distribuer une application uti-

lisant l'une de ces bibliothèques, seul PyQt peut poser problème : il est nécessaire de la distribuer sous la licence GPL ou de payer pour une licence commerciale.

Note *› PySide est une autre couche de liaison Python pour Qt, la différence principale étant la licence : PySide est disponible sous la LGPL, très peu restrictive. Cependant, son développement a été longtemps à l'arrêt : il reprend maintenant sous l'égide du Qt Project.*

1.2. Environnement de PyQt

L'un des grands avantages de Qt et de PyQt, c'est qu'ils sont bien entourés. L'environnement de PyQt comporte notamment quelques outils incontournables qui vous simplifieront la vie dans un grand nombre de tâches ; bon nombre proviennent d'ailleurs de Qt en C++ et ont été légèrement adaptés pour PyQt.

Outils

Qt Designer est prévu pour la conception *graphique* d'interfaces, c'est-à-dire que vous les concevez à l'aide de la souris et de glisser-déposer d'éléments pour former la structure de la fenêtre. Une fois composée, la fenêtre peut être transformée en code (à l'aide de l'utilitaire `pyuic`).

Pour les traductions, Qt Linguist peut être utilisé par les traducteurs, en combinaison avec d'autres outils pour extraire le texte et le transmettre (`pylupdate`) et pour générer la version des traductions utilisable par PyQt (`lrelease`).

`pyqtdeploy` est un utilitaire qui facilite le déploiement d'applications PyQt, en prenant en compte ses spécificités (notamment la dépendance à des bibliothèques non écrites en Python, qui sont une source de problèmes avec d'autres outils de déploiement).

Tous ces outils sont généralement bien intégrés dans des environnements de développement intégrés (EDI), le principal pour PyQt étant eric. Qt Creator est l'environnement de référence pour Qt et s'ouvre petit à petit à Python ; il jouit également d'une très bonne intégration avec ces outils ; il est d'ailleurs le seul EDI gratuit à intégrer Qt Quick.

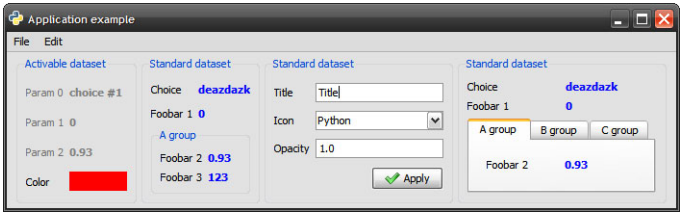
Bibliothèques

Outre ces quelques outils, livrés en standard avec toute distribution de PyQt, l'environnement contient un grand nombre de bibliothèques implémentant des fonctionnalités moins courantes, mais néanmoins utiles selon les cas.

Beaucoup d'applications exploitent une base de données (notamment les exemples développés dans la suite de l'ouvrage), principalement relationnelles (SQL). Camelot per-

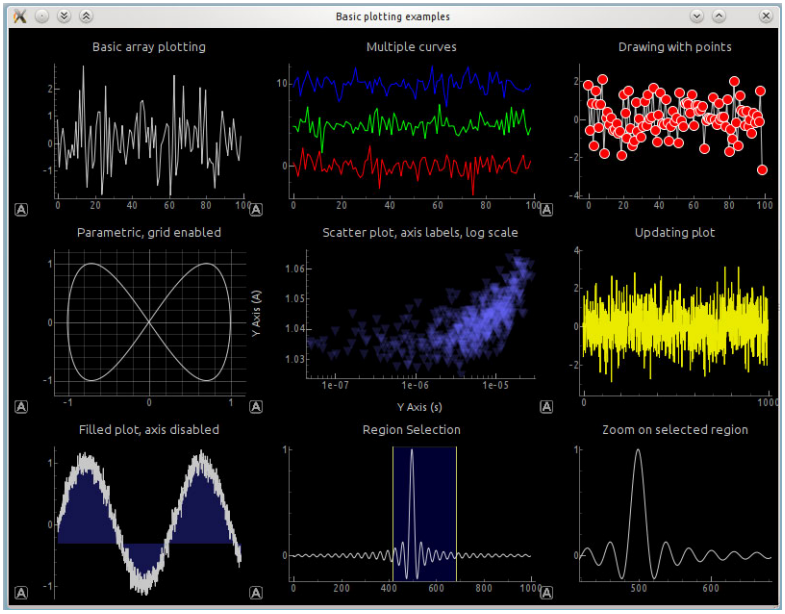
met d'utiliser une interface orientée objet pour ces dernières (à travers SQLAlchemy). GuiData remplit un objectif similaire ; il est mis à jour plus fréquemment, mais se limite à la partie d'édition des données.

Figure 1.1 : *Exemple d'interface réalisée avec GuiData*



Côté graphiques, PyQtGraph et GuiQwt proposent des fonctionnalités similaires pour la visualisation de données bi- et tridimensionnelles ou encore d'images avec éléments interactifs. Ces composants sont principalement prévus dans des applications scientifiques, notamment dans les domaines de l'ingénierie ou du médical.

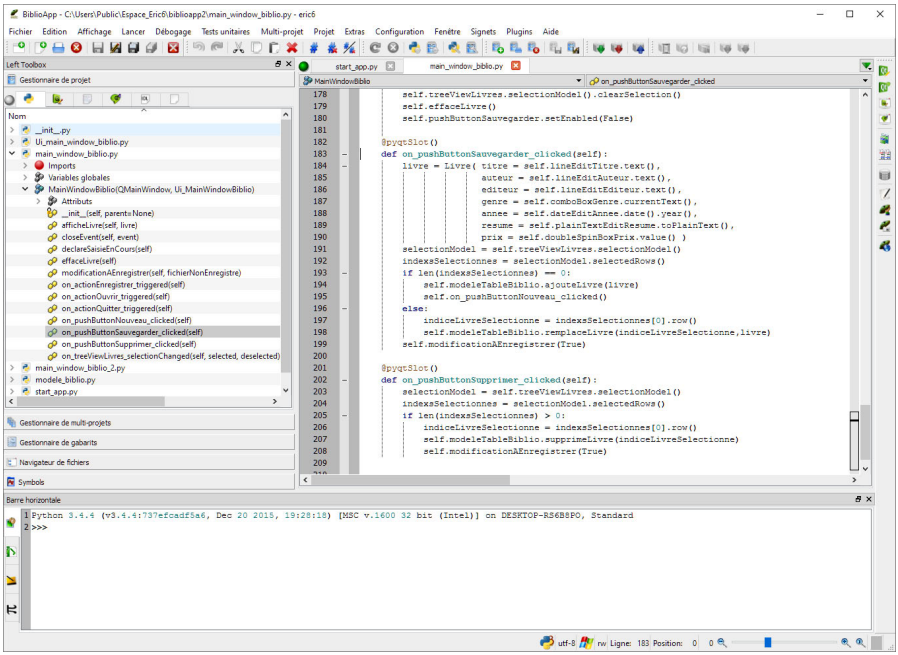
Figure 1.2 : *Quelques possibilités de PyQtGraph en 2D*



Note *› Depuis PyQt 5.7, des outils similaires sont fournis gratuitement avec PyQt, même s'ils doivent être téléchargés séparément. PyQtCharts affiche des graphiques bidimensionnels (comme PyQtGraph et GuiQwt). PyQtDataVisualization en est la déclinaison tridimensionnelle.*

QScintilla propose un composant d'édition de texte principalement prévu pour le code informatique, avec notamment la coloration syntaxique ou des facilités de présentation d'erreurs ou d'intégration d'autocomplétions. Il est notamment utilisé pour développer eric6.

Figure 1.3 : *Utilisation de QScintilla dans eric6*



Pour les plateformes mobiles (iOS et Android) ainsi que macOS, PyQtPurchasing propose d'effectuer des achats depuis l'intérieur d'une application tout en s'intégrant avec l'outil d'achat d'applications de la plateforme (Mac App Store, App Store, Android Market).

1.3. Choix d'une interface

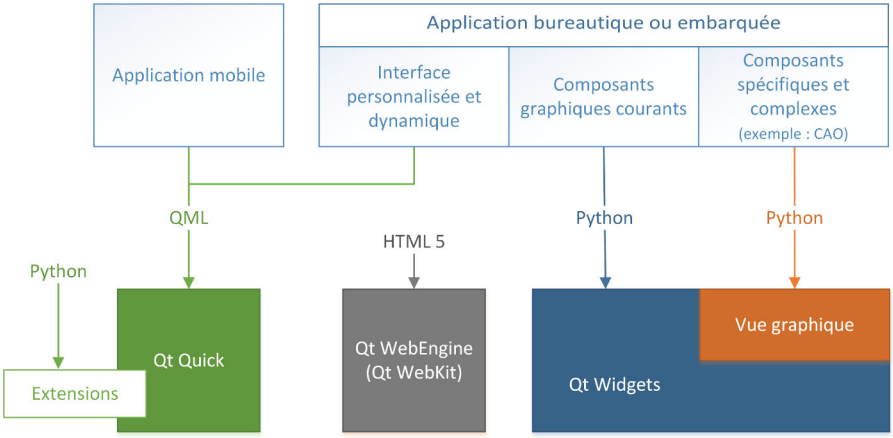
Tout comme PyQt par rapport à ses concurrents, à l'intérieur même de la bibliothèque, vous devrez effectuer un choix pour la manière d'écrire votre application. Comment s'y retrouver entre Qt Widgets, Qt Quick, Graphics View ? Ces différentes approches ont peut-être l'air de s'opposer, mais elles peuvent se compléter mutuellement pour profiter des avantages de chacune dans une même application.

Outre une affaire de goût, ces méthodologies se distinguent sur le type d'applications visé. De manière générale :

- pour développer une interface graphique classique sur un ordinateur avec clavier et souris, **Qt Widgets** est bien adapté. Cette possibilité, qui utilise uniquement le langage Python, est détaillée dans Développement d'une application avec des widgets. Si cette interface nécessite par ailleurs des graphiques complexes présentant beaucoup d'interactions avec l'utilisateur (vue cartographique, CAO, jeux, etc.), l'approche Qt Widgets peut être complétée avec le framework **Graphics View**, qui est couvert dans Affichage 2D interactif avec les vues graphiques ;
- pour développer une interface pour un téléphone portable, une tablette ou un écran tactile en général, le meilleur choix est probablement **Qt Quick**, puisqu'il a été prévu pour ces utilisations, mais il reste parfaitement adapté à bon nombre d'applications plus traditionnelles. Il nécessite cependant l'apprentissage d'un autre langage de programmation, QML. De manière générale, les nouveautés de Qt seront développées pour Qt Quick (bien que généralement disponibles en Python). Cette possibilité est détaillée dans Développement d'une application avec Qt Quick ;
- pour développer une interface avec des technologies web (par exemple, pour utiliser le même code sur un site web ou pour une vraie application), c'est-à-dire une application hybride, votre choix devra se porter sur **Qt WebEngine**. Ces applications doivent exploiter une base avec Qt Widgets ou Qt Quick. Vous trouverez régulièrement des références à Qt WebKit, mais ce dernier n'est plus maintenu officiellement (même si la communauté a remis le projet au goût du jour).

Ces premiers éléments sont lapidaires et permettent d'orienter rapidement un choix vers une technologie qui sera probablement adaptée, bien que la situation varie énormément, notamment selon les habitudes des développeurs.

Figure 1.4 : Choix d'une technologie de création d'interfaces graphiques



Vue graphique ou Qt Quick ?

Lorsque la vue graphique (Graphics View) de Qt est apparue avec Qt 4.2, en 2006, il s'agissait du summum du modernisme pour afficher de grands nombres d'éléments à l'écran... puis Qt Quick est arrivé.

Ce nouveau venu fut, en réalité, une opportunité pour la vue graphique : la première version de Qt Quick l'utilisait pour son implémentation, guidait la danse pour les prochaines fonctionnalités et améliorations de performance. Depuis PyQt 5.0, cependant, Qt Quick n'utilise plus ce cadriciel pour son implémentation, mais bien un graphe de scène et un rendu effectué directement avec OpenGL. Ce n'était pas un aveu de faiblesse de la vue graphique : ce cadre a été très utile pour proposer une version de Qt Quick performante, avec un temps de développement très réduit. Néanmoins, il s'agissait d'une couche d'abstraction pas forcément utile pour l'utilisateur final de Qt Quick et elle ne permettait pas toutes les optimisations nécessaires pour les ambitions que nourrissaient les développeurs envers Qt Quick. Depuis ce changement radical, la vue graphique n'a pas beaucoup évolué, comme le montre l'historique Git.

Alors, que faut-il choisir : vue graphique ou Qt Quick ? Ce dernier n'est pas forcément un remplaçant pour tous les emplois de la vue graphique : au contraire, il lui manque un certain nombre de fonctionnalités, plus ou moins importantes selon les cas d'utilisation.

- La vue graphique distingue clairement la vue (QGraphicsView) de la scène (QGraphicsScene) : une vue n'est qu'une manière de voir une scène (voir la Section 10.2, Scènes, éléments et vues). On peut ainsi disposer d'un grand nombre de vues d'une même scène, avec une très bonne performance.

  Au contraire, avec Qt Quick, cette fonctionnalité (pas si souvent utile, voire inutilisable dans bon nombre d'applications mobiles et embarquées) n'a pas été répliquée : pour y arriver, il faut impérativement créer plusieurs scènes, à garder synchronisées – ou alors utiliser Qt 3D (voir chapitre Affichage 3D avec Qt 3D) et plusieurs caméras. Ce faisant, le code de rendu de Qt Quick a pu être grandement simplifié et optimisé.

- Les QGraphicsItem disposent d'une solution de détection de collisions efficace, totalement absente de Qt Quick. C'est un réel handicap pour réaliser des jeux, vu que ces algorithmes doivent être implémentés pour chaque application – tout en faisant attention à la performance, souvent critique pour des jeux mobiles. L'infrastructure nécessaire est cependant arrivée avec Qt 3D (voir chapitre Affichage 3D avec Qt 3D).

- Qt Quick ne dispose, pour le moment, que d'une seule forme de base : le rectangle. Certes, il est très configurable (avec des bords arrondis, on peut dessiner des cercles, des ellipses), mais le niveau de fonctionnalité est très loin de celui proposé par la vue graphique et ses formes personnalisables à l'infini.

  On peut obtenir un résultat similaire avec le composant Canvas de Qt Quick émulant l'API HTML5 du même nom (c'est-à-dire que le dessin est piloté en JavaScript, avec les problèmes de performance qui l'accompagnent – voir le chapitre Affichage 2D avec Canvas), mais sans réelle intégration à l'environnement. On peut aussi créer ses propres composants Qt Quick en Python, voire créer ses propres nœuds pour le rendu dans le graphe de scène (ce qui apportera une performance maximale, mais avec un complexité d'implémentation très élevée).

- La vue graphique est entièrement pensée pour Python, il est très facile de créer ses propres classes. Au contraire, l'interface Python de Qt Quick n'est pas pensée pour l'extension : un très grand nombre de composants n'a qu'une interface privée. Tout ce travail d'extension est censé être fait en QML (il est d'ailleurs encore plus aisé qu'en Python), mais la performance peut en pâtir.

  En d'autres termes, il n'est pas facile d'y accéder depuis son propre code et, de toute façon, les développeurs de Qt ne garantissent aucunement sa stabilité : tout code qui utilise directement les composants Qt Quick en Python (par

héritage, par exemple) n'a aucune garantie de continuer à fonctionner avec la prochaine version de Qt.

Cette décision a beaucoup de sens : l'utilisateur peut toujours faire ce qui lui plaît en QML (le code ne sera pas complètement cassé par une mise à jour), tout en laissant aux développeurs de Qt la liberté de faire évoluer l'API interne, notamment à cause du jeune âge de Qt Quick.

Quelles conclusions peut-on en tirer ? La vue graphique n'est plus l'objet de grandes attentions, mais est extrêmement stable et performante : elle est d'ailleurs utilisée dans un très grand nombre d'applications. Il n'est pas envisagé, pour le moment, de les retirer de Qt. Néanmoins, la politique actuelle de Qt indique clairement que Qt Quick est l'avenir pour la création d'interfaces graphiques : les nouvelles fonctionnalités de Qt sont d'ailleurs généralement utilisables aussi bien en Python qu'en QML (comme Qt 3D).

Ce livre ne pose pas de choix particulier : pour le moment, les deux approches ont du sens, selon le type d'application à développer. Ainsi, nous présentons tant la vue graphique (Affichage 2D interactif avec les vues graphiques) que QtQuick (Développement d'une application avec Qt Quick). Vous pourrez ainsi vous faire votre propre avis sur chacune de ces deux solutions et voir laquelle est la plus appropriée à vos besoins.

Note *› À titre de comparaison, PyQt propose la classe QPainter pour le dessin à l'écran (tout ce qui n'est pas possible avec les classes de base de Qt, comme des courbes, des graphiques scientifiques, etc.). Elle est déjà relativement ancienne (première moitié des années 2000 !), mais reste là : malgré l'apparition de solutions nettement plus évoluées, plus complexes (les vues graphiques et Qt Quick en particulier), les fonctionnalités de QPainter sont toujours là et elles ne devraient pas disparaître avant longtemps. Les changements sont rares, mais pas inexistants : courant 2017, elle sera adaptée aux technologies de rendu les plus modernes (une dépendance à OpenGL 2 limitée).*