Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PROGRAMMER EN PYTHON

Volume six

Parties 32 à 38

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

En réponse aux requêtes des lecteurs, nous avons réuni le contenu de certains articles consacrés à la programmation en python.

Pour l'instant, il s'agit d'une réédition directe de la série Débuter Python, parties 32 à 38, des numéros 60 à 67 ; en effet, nous avons permis à l'incomparable professeur en Python Greg Walters de sauter le numéro 66 parce qu'il a été très sage.

Veuillez considérer l'origine de la publication ; les versions actuelles du matériel et des logiciels peuvent différer de ceux que nous présentons, ainsi vérifiez bien votre matériel et la version de vos logiciels avant d'émuler les tutoriels de cette édition spéciale. Vous pouvez installer des versions de logiciels plus récentes ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Nos coordonnées

SiteWeb :
http://www.fullcirclemagazine.org/

Forums :
http://ubuntuforums.org/forumdisplay.php?f=270

IRC : #fullcirclemagazine on chat.freenode.net

Équipe éditoriale :
Rédacteur en chef : Ronnie Tucker (pseudo : RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster : Rob Kerfia (pseudo : admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcast : Robin Catling (pseudo : RobinCatling)
podcast@fullcirclemagazine.org
Dir. comm. : Robert Clipsham (pseudo : mrmonday) -
mrmonday@fullcirclemagazine.org

TUTORIEL

Débuter Python - Partie 32

Greg Walters

Je dois l'avouer, j'aime ma tablette Android. Alors que je l'utilise tous les jours, elle n'est pas encore un remplaçant pour mon ordinateur de bureau. Et je dois aussi admettre que ce pourquoi je l'utilise principalement est à peu près ce pourquoi tout le monde utilise la sienne : naviguer sur le Web, écouter de la musique, regarder des vidéos, jouer à des jeux et ainsi de suite. J'essaie de la justifier en ayant des applications qui ont un rapport avec des listes de courses et de tâches à faire, la recherche d'essence pas chère, des choses amusantes pour mon petit-fils, etc. C'est vraiment un jouet pour moi en ce moment. Pourquoi utiliser une tablette tactile amusante pour faire votre liste de courses ? Avouons-le… Ce sont les regards d'envie que les gens me jettent dans le magasin quand ils me voient pousser le caddy dans l'allée et tapoter ma tablette pour barrer les éléments de la liste. Ah, le facteur geek RÈGNE ! Bien sûr, je peux utiliser le dos d'une vieille enveloppe pour faire ma liste. Mais ce ne serait ni cool ni geeky, hein, n'est-ce pas ?

Comme 99 % des hommes geeks mariés dans le monde, je suis marié à une femme non-geek. Une merveilleuse femme aimante, bien sûr, mais une non-geek qui, lorsque je commence à baver sur le dernier gadget, soupire et dit quelque chose comme : « Eh bien, si tu penses VRAIMENT que nous avons besoin de ça… ». Puis elle m'envoie le même regard que je lui envoie quand elle caresse affectueusement la 50e paire de chaussures du magasin.

En toute honnêteté, il n'a pas été difficile de ramener la première tablette à la maison. Je l'ai achetée pour ma femme alors qu'elle suivait un traitement de chimiothérapie. Elle a essayé d'utiliser un ordinateur portable pendant quelque temps, mais la chaleur et le poids sur ses genoux étaient insupportables au bout d'un certain temps. Les e-books sur un ordinateur portable n'étaient pas sa tasse de thé, alors quand elle a essayé de lire, elle a dû jongler entre le livre, l'ordinateur portable et le lecteur mp3, tout en étant attachée à un transat avec des tubes dans le bras la remplissant de vilains produits chimiques. Lorsque je lui ai offert la tablette, ce fut le meilleur des mondes. Elle pouvait lire un livre électronique, écouter de la musique, regarder la télé, naviguer sur le Web, vérifier ses mails, mettre à jour son blog sur le cancer, suivre ses amis sur facebook et jouer à des jeux, le tout sur un appareil qui était léger et sympa. Si elle était fatiguée, elle pouvait tout simplement la glisser sur le côté entre elle et le transat (ou le lit quand elle était à la maison pour essayer de reprendre des forces). Bien mieux qu'un ordinateur portable encombrant et un livre, un lecteur mp3, une télécommande et plus encore.

Alors qu'elle se faisait remplir de produits chimiques nocifs, je réquisitionnais une table et une chaise dans le coin de la salle de traitement, à proximité d'une prise de courant, et j'essayais de travailler sur mon vieux portable de six ans. Entre les projets, je faisais des recherches sur la programmation Android. J'ai découvert que la plupart des programmes pour Android se faisait en Java. J'étais presque résigné à ré-apprendre le Java lorsque je suis tombé sur quelques outils qui permettent de programmer en Python pour le système d'exploitation Android. Un de ces outils est appelé « SL4A ». SL4A veut dire Scripting Layer for Android (Couche de Script pour Android). C'est ce sur quoi nous allons nous concentrer dans les deux prochains articles. Dans celui-ci, en fait, nous allons nous concentrer sur la mise en place de SL4A sous Android.

De nombreuses pages web montrent comment charger SL4A dans l'émulateur Android pour ordinateur de bureau. Nous allons essayer de faire cela une autre fois, mais pour l'instant nous allons jouer avec l'appareil Android lui-même. Pour installer SL4A sur votre appareil Android, allez à http://code.google.com/p/android-scripting/ ; vous y trouverez le fichier d'installation pour SL4A. Ne soyez pas totalement perdu ici. Il y a un code QR sur

lequel vous appuyer pour télécharger l'APK. Assurez-vous d'avoir activé l'option « Unknown Sources (Sources inconnues) » dans les paramètres Application. C'est un téléchargement rapide. Une fois que vous l'avez téléchargé et installé, trouvez l'icône et appuyez dessus. Ce que vous verrez est un écran noir, plutôt décevant, disant « Scripts…No matches found » (Scripts… Aucune correspondance trouvée). Ce n'est pas grave. Cliquez sur le bouton menu et sélectionnez View (Afficher). Vous verrez un menu. Sélectionnez Interpreters (interprètes). Ensuite, sélectionnez de nouveau le menu, puis sélectionnez Add (Ajouter). Dans le menu suivant, sélectionnez Python 2.6.2. Cela devrait vous demander de lancer un navigateur pour télécharger Python pour Android. Une fois qu'il est installé, sélectionnez Open (Ouvrir). Vous obtiendrez un menu à l'écran avec les choix Install, Import Modules, Browse Modules (parcourir les modules), et Uninstall Modules. Sélectionnez l'option Install. Maintenant Python va se télécharger et s'installer avec d'autres modules supplémentaires. De plus, vous aurez des exemples de scripts. Enfin, appuyez sur le bouton de retour et vous verrez Python 2.6.2 installé dans l'écran des interprètes. Appuyez de nouveau sur le bouton de retour et vous verrez une liste de quelques exemples de scripts python.

> **Ce que vous verrez est un écran noir, plutôt décevant. Ce n'est pas grave.**

C'est tout ce que nous allons faire cette fois-ci. Tout ce que je voulais faire, c'était vous mettre en appétit. Explorez Python sous Android. Vous pouvez également visiter http://developer.android.com/sdk/index.html pour obtenir le SDK Android (Soft-ware Development Kit) pour votre bureau. Il comprend un émulateur Android afin que vous puissiez jouer avec tout de suite. La mise en place du SDK est vraiment très facile sous Linux et vous ne devriez pas avoir trop de mal.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Comment inclure des accents à partir du clavier

par Barry Smith

Si votre système Linux est en français, allemand ou espagnol et, par conséquent, exige des accents, ou si, de temps en temps, vous avez besoin d'utiliser des accents qui ne figurent pas dans les mots anglais, de nombreux utilisateurs ne savent pas qu'il existe un moyen très facile pour le faire à partir du clavier. Ce qui suit ne s'applique qu'à un clavier du Royaume-Uni.

Accent aigu
Appuyez sur Alt Gr + ; (point-virgule). Levez la main puis appuyez sur la voyelle souhaitée é.

Accent circonflexe
Appuyez sur Alt Gr + ' (apostrophe). Levez la main puis appuyez sur la voyelle souhaitée î.

Accent grave
Appuyez sur Alt Gr + # (dièse). Levez la main puis appuyez sur la voyelle souhaitée è.

Trema
Appuyez sur Alt Gr + [. Levez la main puis appuyez sur u ü.

ñ - Appuyez sur Alt Gr +]. Levez la main puis appuyez sur n ñ.

œ - Appuyez sur Maj + Alt Gr. Levez la main puis appuyez sur o puis appuyez sur e œ. Le œ n'apparaîtra pas avant que le e ne soit tapé.

Pour obtenir ¿ et ¡ (point d'exclamation et d'interrogation inversés) dont je me sers tout le temps en espagnol avant les questions et les exclamations, appuyez sur Alt Gr + Maj, en gardant les deux touches enfoncées, puis appuyez sur _ (souligné) pour ¿ ou tapez ! (point d'exclamation) pour ¡.

Si vous voulez une de ces lettres en capitales, il suffit d'appuyer sur Maj avant de taper de la lettre.

Débuter Python - Partie 33

Greg Walters

Ce mois-ci, nous allons mettre en place le SDK Android sur notre bureau Linux. Nous allons aussi créer un périphérique Android virtuel, installer SL4A et Python dessus, et faire un test rapide.

S'il vous plaît faites attention, ce n'est pas quelque chose qu'il faut faire sur une machine qui a moins de 1 Go de RAM. L'émulateur consomme une énorme quantité de mémoire. Je l'ai essayé sur un ordinateur portable fonctionnant sous Ubuntu avec seulement 512 Mo de RAM. Il fonctionne, mais il est VRAIMENT lent.

Voici une liste rapide de ce que nous allons faire. Nous allons y aller étape par étape dans une minute :

- Installer le JDK6 Java.
- Installer le pack de démarrage SDK Android.
- Créer et configurer les AVD.
- Tester AVD et installer SL4A et Python.

En réalité, nous devrions également installer Eclipse et le plugin Android ADT pour Eclipse, mais, puisque nous n'utiliserons pas Eclipse dans cette série d'articles, nous pouvons éviter cela. Si vous souhaitez les inclure, allez voir sur http://developer.android.com/sdk/installing.html pour voir toutes les étapes dans l'ordre suggéré. Nous allons maintenant commencer.

ÉTAPE 1 - Java JDK 6

D'après tout ce que j'ai lu et essayé, il faut vraiment utiliser la version de Sun. OpenJDK n'est pas censé fonctionner. Vous pouvez trouver des informations à ce sujet sur le web, mais voici les étapes que j'ai suivies. Dans un terminal, tapez les commandes suivantes :

```
sudo add-apt-repository
ppa:ferramroberto/java

sudo apt-get update

sudo apt-get install sun-
java6-jdk
```

Une fois que tout ceci est fait, vous devrez modifier votre fichier .bashrc pour régler « JAVA_HOME » pour que tout fonctionne correctement. J'ai utilisé gedit pour ajouter la ligne suivante à la fin du fichier :

```
export
JAVA_HOME="/usr/lib/jvm/java-
6-sun-1.6.0.06"
```

Enregistrez le fichier et passez à l'étape 2.

ÉTAPE 2 - Pack de démarrage Android SDK

Maintenant, la partie « marrante » commence. Rendez-vous sur http://developer.android.com/sdk/index.html. C'est là que se trouve le SDK. Téléchargez la dernière version pour Linux qui, au moment d'écrire ces lignes, est android-sdk_r18-linux.tgz. À l'aide du Gestionnaire d'archives, décompressez-la dans un dossier approprié. Je l'ai mise dans mon répertoire personnel. Tout fonctionne directement à partir de ce dossier, vous n'avez donc vraiment pas besoin d'installer quoi que ce soit. Ainsi, le chemin pour moi est /home/greg/android-sdk-linux. Allez dans ce dossier, puis allez dans le dossier des outils (« tools »). Vous y trouverez un fichier nommé « android ». C'est lui qui lance réellement le SDK. J'ai créé un lanceur sur mon bureau pour en faciliter l'accès.

À présent, la partie ennuyeuse. Exécutez le fichier android ; le gestionnaire de SDK Android va démarrer. Il va mettre à jour les plateformes qui sont disponibles. Je vous préviens maintenant que ce processus prendra un certain temps, alors ne vous embêtez pas si vous n'avez pas beaucoup de temps pour y faire face. Par souci de concision, je vous conseille de n'avoir qu'une plateforme pour commencer. Je vous suggère de commencer par Android 2.1, puisqu'en général si vous développez pour une ancienne plateforme, il ne devrait y avoir aucun problème d'exécution sur une nouvelle plateforme. Vous devez également récupérer l'ensemble des outils. Il suffit de cocher la case à côté de ces deux éléments, puis de cliquer sur le bouton d'installation. Une fois que vous avez obtenu la plateforme de votre choix et l'ensemble d'outils, vous êtes presque prêt à créer votre première machine virtuelle.

ÉTAPE 3 - Créer et configurer votre première AVD

Retournez dans le Gestionnaire de SDK Android, sélectionnez Outils (« Tools ») dans le menu principal, puis sélectionnez « Gérer les AVD ». Cela va ouvrir une nouvelle fenêtre. Puisque c'est la première fois, il n'y aura pas encore de périphérique virtuel configuré. Cliquez sur le bouton « Nouveau ». Cela ouvre une autre fenêtre où nous définissons les propriétés du périphérique virtuel Android. Voici les étapes que vous devrez suivre

pour mettre en place un dispositif émulateur Android simple :

• Définissez le nom de l'appareil. Ceci est important si vous avez plus d'un appareil.
• Réglez le niveau de plateforme cible.
• Définissez la taille de la carte SD (voir ci-dessous).
• Réglez la résolution.
• Créez le périphérique.

Par exemple, dans la zone de texte Nom, tapez « Test1 ». Pour la cible, sélectionnez Android 2.1 - API de niveau 7. Dans la boîte pour « Carte SD : », entrez 512 et assurez-vous que la liste affiche « Mio ». Dans « Skin », réglez la résolution à 800×600. (Vous pouvez jouer avec les autres paramètres de tailles.) Enfin, cliquez sur le bouton « Créer AVD ». Vous verrez alors un message disant que l'AVD a été créée.

ÉTAPE 4 - Test de l'AVD et installation de SL4A et Python

Maintenant, enfin, nous pouvons nous amuser un peu. Mettez en surbrillance l'AVD que vous venez de créer et cliquez sur le bouton Démarrer. Dans la boîte de dialogue qui apparaît, cliquez simplement sur le bouton « Lancer ». Vous devez alors attendre quelques minutes pour que le périphérique virtuel soit créé dans la mémoire et que la plateforme Android soit chargée et démarrée. (Nous reparlerons de l'accélération de ce processus dans un prochain article.)

Une fois que l'AVD a démarré et que vous avez l'écran d'accueil, vous allez installer SL4A. En utilisant le navigateur ou la boîte de recherche Google Web sur l'écran d'accueil, recherchez « sl4a ». Allez à la page des téléchargements et vous finirez par trouver la page web pour les téléchargements http://code.google.com/p/android-scripting/downloads/list.

Faites défiler la page jusqu'à ce que vous obteniez le lien sl4a_r5. Ouvrez le lien et tapez sur le lien « sl4a_r5.apk ». Remarquez que j'ai dit « tapez » plutôt que « cliquez ». Commencez à penser à votre doigt qui appuie sur l'écran plutôt que de cliquer avec la souris. Cela facilitera votre transition vers la programmation. Vous verrez le début de téléchargement. Vous pourriez avoir à tirer vers le bas la barre de notification en haut pour obtenir le fichier téléchargé. Tapez sur le fichier, puis sur le bouton d'installation.

Une fois le fichier téléchargé, vous verrez la possibilité d'ouvrir l'application téléchargée ou de taper sur « Terminé » pour quitter le programme d'installation. Ici, il faut taper sur « Ouvrir ».

Maintenant SL4A va démarrer. Vous verrez probablement une boîte de dialogue vous demandant si vous acceptez un suivi de l'utilisation. C'est à vous de décider si vous voulez accepter ou refuser. Avant d'aller plus loin, vous devriez connaître quelques raccourcis clavier qui vous aideront à vous déplacer. Comme nous n'avons pas un « vrai » appareil Android, les boutons Retour, Accueil et Menu ne sont pas disponibles. Vous en aurez besoin pour naviguer. Voici quelques raccourcis importants :

Retour - Échap
Accueil - Début
Menu - F2

Maintenant, nous voulons télécharger et installer Python dans SL4A. Pour faire cela, appuyez d'abord sur Menu (F2). Sélectionnez « Affichage » dans le menu. Maintenant, sélectionnez « Interprètes ». On dirait que rien ne se passe, mais appuyez sur Menu à nouveau (F2), puis sélectionnez « Ajouter » dans le menu contextuel. Maintenant, faites défiler vers le bas et sélectionnez « Python 2.6.2 ». Ceci va télécharger le paquet de base Python pour Android. Installez le paquet puis ouvrez-le. Vous verrez quatre options. Installer, importer des modules, parcourir les modules et désinstaller un module. Tapez sur Installer. Cela va démarrer le téléchargement et l'installation de tous les morceaux de la dernière version de Python pour Android. Cela peut prendre quelques minutes.

Une fois que tout est terminé, appuyez sur Retour (touche Échap) jusqu'à ce que vous arriviez à l'écran des interprètes SL4A. Maintenant tout est chargé pour que nous puissions jouer en Python sur Android. Tapez sur Python 2.6.2 et vous vous trouverez dans la ligne de commande standard de Python. C'est exactement comme la ligne de commande sur votre bureau. Saisissez les trois lignes suivantes, une à la fois, dans la ligne de commande. Assurez-vous d'attendre l'invite « > » à chaque fois.

```
import android

droid = android.Android()

droid.makeToast("Bonjour
depuis Python pour Android")
```

Après avoir tapé la dernière ligne et appuyé sur Entrée, vous verrez une fenêtre aux coins arrondis centrée en bas de la ligne de commande, qui dit : « Bonjour depuis Python pour Android ». C'est ce que fait la commande droid.makeToast.

Vous avez écrit votre premier script Python pour Android. Chouette, hein ?

Maintenant, nous allons créer un raccourci sur l'écran d'accueil d'Android. Tapez sur la touche Accueil (bouton Début). Si vous avez choisi la plateforme 2.1, vous devriez voir une barre de défilement à l'extrême droite de l'écran. Si vous avez choisi une autre plateforme, il se pourrait que ce soit un carré ou un rectangle composé de petits carrés. De toutes les façons, cela vous amène à l'écran des Applis. Tapez dessus et trouvez l'icône SL4A. Maintenant effectuez un « taper long » (clic long), qui créera un raccourci sur l'écran d'accueil. Déplacez le raccourci où vous le souhaitez.

Ensuite nous allons créer notre premier script sauvegardé. Retournez dans SL4A. Vous devriez voir les exemples de scripts fournis avec Python pour Android. Tapez sur le bouton Menu et sélectionnez « Ajouter ». Sélectionnez « Python 2.6.2 » dans la liste. Vous verrez l'éditeur de script. Au sommet se trouve la boîte de nom de fichier avec « .py » déjà rempli. En dessous se trouve la fenêtre de l'éditeur qui contient déjà les deux premières lignes de notre programme saisies pour nous. (Je les ai inclues ci-dessous en italique pour que vous puissiez le vérifier. Nous avons également utilisé ces deux lignes dans notre premier exemple.)

```
import android

droid = android.Android()
```

Maintenant saisissez les deux lignes suivantes dans le script python :

```
uname =
droid.dialogGetInput("Quel
est votre nom ?")

droid.makeToast("Bonjour
%s depuis Python pour
Android") % uname.result
```

La première ligne nouvelle crée une boîte de dialogue (droid.dialogGetInput()) qui demande son nom à l'utilisateur. La réponse est retournée à notre programme dans uname.result. Nous avons déjà utilisé la fonction droid.makeToast().

Nommez le fichier andtest1.py, puis tapez sur Terminé puis sur « Enregistrer et Exécuter ». Si tout s'est bien passé, vous devriez voir une boîte de dialogue vous demandant votre nom. Après l'avoir saisi, vous devriez voir l'alerte en bas de l'écran qui affiche : « Bonjour Votre nom depuis Python pour Android ».

C'est tout pour cette fois-ci. Pour l'instant, il y a une tonne de documentation gratuite sur SL4A sur le web. Vous pouvez jouer un peu tout seul jusqu'à la prochaine fois. Je vous suggère de commencer par http://code.google.com/p/android-scripting/wiki/Tutorials.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Débuter Python - Partie 34

Greg Walters

Cette fois-ci, nous allons en terminer avec l'utilisation de SL4A. Nous allons faire un programme plus important, puis l'envoyer à la machine virtuelle via ABD.

Occupons-nous d'abord de notre code. Pour cela, nous allons simplement essayer quelques-uns des « widgets » qui sont à notre disposition lorsqu'on utilise SL4A. Démarrez sur votre bureau à l'aide de votre éditeur de texte favori.

```
import android

droid = android.Android()
uname = droid.dialogGetInput("Hello","What's your name?")
droid.makeToast("Hello %s from python on Android!" % uname.result)
```

```
droid.dialogCreateAlert(uname.result,"Would you like to play a game?")
droid.dialogSetPositiveButtonText('Yes')
droid.dialogSetNegativeButtonText('No')
droid.dialogShow()
while True: #wait for events for up to 10 seconds...
      response = droid.eventWait(10000).result
      if response == None:
      break
      if response["name"] == "dialog":
      break
droid.dialogDismiss()
```

Entrez le code que vous voyez en haut à droite et sauvegardez-le (mais n'essayez pas de l'exécuter) en tant que « atest.py ».

La première ligne importe la bibliothèque android. Nous en créons une instance dans la deuxième ligne. La ligne 3 crée et affiche une boîte de dialogue avec le titre « Bonjour », l'invite « Quel est votre nom ? », une zone de texte pour que l'utilisateur saisisse son nom, et deux boutons, « OK » et « Annuler ». Une fois que l'utilisateur appuie sur « OK », la réponse est renvoyée dans la variable nomutilisateur. La dernière ligne (pour l'instant) affiche alors : « Bonjour {nom} depuis Python sur Android ! ». Ce n'est pas nouveau, nous l'avons fait la dernière fois. Maintenant, nous allons ajouter plus de code (ci-dessus).

Enregistrez votre code sous « atest1.py ». Nous allons envoyer cela à notre machine virtuelle après avoir discuté de ce qu'il fait.

Jetez un coup d'œil aux quatre premières lignes que nous venons de saisir. Nous créons une boîte de dialogue d'alerte demandant « Voulez-vous jouer à un jeu ? » Dans le cas d'une boîte de dialogue d'alerte, il n'y a pas de zone de texte pour saisir quoi que ce soit. Les deux lignes suivantes permettent de créer deux boutons, l'un avec le texte « Oui », qui est un bouton « positif », et l'autre avec le texte « Non », un bouton « négatif ». Les boutons positif et négatif se réfèrent à la réponse retournée - positive ou négative. La ligne suivante affiche alors la boîte de dialogue. Les sept lignes suivantes attendent une réponse de l'utilisateur.

Nous créons une simple boucle (while True:), puis attendons une réponse pendant 10 secondes en utilisant la fonction droid.eventWait(valeur). La réponse (soit positive soit négative) sera retournée dans - vous l'aurez deviné - la variable reponse. Si cette variable contient « dialog », nous sortons de la boucle et retournons la réponse. Si rien ne se passe avant que le délai soit dépassé, nous sortons simplement de la boucle. L'information réelle retournée dans la variable réponse ressemble à ceci (en supposant qu'on ait choisi le bouton positif, soit « Oui ») :

```
{u'data': {u'which':
u'positive'}, u'name':
u'dialog', u'time':
1339021661398000.0}
```

Vous pouvez voir que la valeur est passée dans le tableau « data », le texte « dialog » dans le tableau « name » et il y a une valeur « time » dont nous ne nous soucions pas ici.

Enfin, nous fermons la boîte de dialogue.

Avant que nous puissions envoyer notre code à la machine virtuelle, il faut la faire démarrer. Exécutez votre émulateur Android. Une fois qu'il a démarré, vous remarquerez que la barre de titre contient quatre chiffres au début du titre. Il s'agit du port sur lequel la machine est à l'écoute. Dans mon cas (et probablement dans le vôtre) il s'agit de 5554.

Maintenant, nous allons envoyer le code vers notre machine virtuelle. Ouvrez une fenêtre de terminal et allez dans le dossier où vous avez enregistré le code. En supposant que vous avez configuré votre chemin par défaut pour y inclure le SDK, saisissez :

```
adb devices
```

Ceci demande à adb de montrer tous les périphériques qui sont connectés. Cela peut inclure non seulement l'émulateur Android, mais aussi les smartphones, tablettes ou autres appareils Android. Vous devriez voir quelque chose comme ceci :

```
Liste des périphériques
connectés
emulator-5554    device
```

Maintenant que nous sommes sûrs que notre appareil est connecté, nous voulons envoyer l'application sur le périphérique. La syntaxe est la suivante :

```
adb push fichier_source
chemin_et_fichier_destination
```

Donc, dans mon cas, ce serait :

```
adb push atest1.py
/sdcard/sl4a/scripts/atest1.py
```

Si tout fonctionne correctement, vous obtiendrez un message plutôt décevant semblable à ceci :

```
11 Ko/s (570 octets en 0.046s)
```

Maintenant, sur l'émulateur Android, démarrez SL4A. Vous devriez voir tous les scripts python et, parmi eux, vous devriez voir atest1.py. Tapez (cliquez) sur « atest1.py » et vous verrez une boîte de dialogue avec 6 icônes. De gauche à droite, on trouve « Exécuter dans une fenêtre de dialogue », « Exécuter en dehors d'une fenêtre », « Éditer », « Enregistrer », « Supprimer », et « Ouvrir dans un éditeur externe ». Pour le moment, tapez (cliquez) sur l'icône la plus à gauche « Exécuter dans une fenêtre de dialogue » afin que vous puissiez voir ce qui se passe.

Vous verrez la première boîte de dialogue vous demandant votre nom. Entrez quelque chose dans la boîte et tapez (cliquez) sur le bouton « Ok ». Vous verrez le message bonjour. Ensuite, vous verrez la boîte de dialogue d'alerte. Tapez (cliquez) sur l'un des boutons pour fermer la boîte de dialogue. Nous ne regardons pas les réponses pour l'instant alors peu importe celle que vous choisissez. Maintenant, nous allons ajouter un peu plus de code (en haut à droite).

Je suis sûr que vous comprendrez que ce morceau de code vérifie simplement la réponse et, si c'est « Aucune » à cause du temps d'attente, nous affichons simplement « Trop tard ». Et si c'est effectivement quelque chose que nous voulons, alors nous assignons les données à la variable rdialog. Maintenant, ajoutez le bout de code suivant (ci-dessous) :

```
if response==None:
    print "Timed out."
else:
    rdialog=response["data"]
```

Ce morceau de code regardera les données transmises par l'événement d'appui sur un bouton. Nous vérifions si la réponse a une valeur « which » et, si c'est le cas, c'est bien un appui légitime sur un bouton. Nous vérifions ensuite si le résultat est la réponse « positive » (bouton « OK »). Si c'est le cas, nous créons une autre boîte de dialogue d'alerte, mais cette fois nous allons ajouter une liste d'éléments parmi lesquels l'utilisateur

```
    if rdialog.has_key("which"):
    result=rdialog["which"]
    if result=="positive":
        droid.dialogCreateAlert("Play a Game","Select a game to play")
        droid.dialogSetItems(['Checkers','Chess','Hangman','Thermal
Nuclear War']) # 0,1,2,3
        droid.dialogShow()
        resp = droid.dialogGetResponse()
```

choisira. Dans ce cas, nous proposons à l'utilisateur de choisir parmi une liste comprenant Dames, Échecs, Pendu et Guerre nucléaire et nous attribuons les valeurs 0 à 3 pour chaque élément. (Est-ce que cela commence à paraître familier ? Oui, ça vient d'un film.) Nous affichons ensuite la boîte de dialogue et attendons une réponse. La partie de la réponse qui nous intéresse est sous la forme d'un tableau. En supposant que l'utilisateur tape (clique) sur Échecs, la réponse résultante nous revient comme ça :

```
Result(id=12,
result={u'item':1},
error=None)
```

Dans ce cas, nous sommes vraiment intéressés par la partie « result » des données renvoyées. La sélection est le n° 1 et est contenue dans la clé « item ». Voici la partie suivante du code (en haut à droite) :

```
if resp.result.has_key("item"):
    sel = resp.result['item']
    if sel == 0:
        droid.makeToast("Enjoy your checkers game")
    elif sel == 1:
        droid.makeToast("I like Chess")
    elif sel == 2:
        droid.makeToast("Want to 'hang around' for a while?")
    else:
        droid.makeToast("The only way to win is not to play...")
```

Ici, nous vérifions pour voir si la réponse contient la clé « item » et, si c'est le cas, nous l'affectons à la variable « sel ». Ensuite nous utilisons une boucle if/elif/else pour vérifier les valeurs et traiter celle qui est sélectionnée. Nous utilisons la fonction droid.makeToast pour afficher notre réponse. Bien sûr, vous pourriez ajouter votre propre code ici. Maintenant, voici la dernière partie du code (au milieu à droite).

Comme vous pouvez le voir, nous répondons simplement aux autres appuis de boutons ici.

Sauvegardez, envoyez, puis exécutez le programme.

Comme vous pouvez le voir, SL4A vous donne la possibilité de faire des applications « graphiques », mais pas complètement. Cela ne devrait toutefois pas vous empêcher d'aller de

```
        elif result=="negative":
            droid.makeToast("Sorry.  See you later.")
    elif rdialog.has_key("canceled"):
        print "Sorry you can't make up your mind."
    else:
        print "unknown response=",response
print "Done"
```

l'avant et de commencer à écrire vos propres programmes pour Android. Ne vous attendez pas à les mettre sur le « marché ». La plupart des gens veulent vraiment des applications complètement graphiques. Nous verrons cela la prochaine fois. Pour plus d'informations sur l'utilisation de SL4A, il suffit de faire une recherche sur Internet et vous trouverez beaucoup de tutoriels et d'autres informations.

Au fait, vous pouvez envoyer votre code directement sur votre smartphone ou votre tablette de la même manière.

Comme d'habitude, le code a été mis en place sur pastebin à http://pastebin.com/BHJWqjCf

Rendez-vous la prochaine fois.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

Débuter Python - Partie 35

Greg Walters

Cette fois-ci, nous allons dévier un peu de notre exploration de la programmation Android et regarder un nouvel environnement pour la programmation d'interfaces graphiques appelé Kivy. Vous pouvez aller sur http://kivy.org pour télécharger et installer le paquet, avant d'aller plus loin dans cet article. Les instructions d'installation pour Ubuntu sont ici : http://kivy.org/docs/installation/installation-ubuntu.html.

Tout d'abord, Kivy est une bibliothèque Open Source qui gère les écrans tactiles. Si ce n'est pas encore assez chouette, elle est aussi multiplateforme, ce qui signifie qu'elle fonctionne sous Linux, Windows, Mac OSX, iOS et Android. Maintenant vous comprenez pourquoi nous en parlons. Mais rappelez-vous, la plupart du temps, tout ce que vous codez à l'aide de Kivy peut fonctionner sur n'importe laquelle des plateformes ci-dessus sans recodage.

Avant d'aller trop loin, permettez-moi de faire quelques déclarations. Kivy est TRÈS puissant. Kivy vous procure un nouvel ensemble d'outils pour programmer des interfaces graphiques. Tout cela étant dit, Kivy est également assez compliqué à gérer. Vous êtes limité aux widgets qu'ils ont fournis. En outre, il n'existe pas de designer graphique pour Kivy et vous devez donc faire beaucoup de planification préalable avant d'essayer de faire des trucs compliqués. Rappelez-vous aussi que Kivy est constamment en développement et que les choses peuvent changer rapidement. Jusqu'à présent, je n'ai jamais eu de morceaux de mon code qui aient été cassés par une nouvelle version de Kivy, mais c'est toujours une possibilité.

Plutôt que de nous précipiter et de créer notre propre code ce mois-ci, nous allons examiner quelques-uns des exemples livrés avec Kivy et, le mois prochain, nous préparerons le nôtre.

Une fois que vous avez décompressé Kivy dans son propre dossier, utilisez un terminal et allez dans ce dossier. Le mien est dans /home/greg/Kivy-1.3.0. Maintenant, allez dans le dossier des exemples, puis dans le dossier widgets. Regardons l'exemple accordion_1.py.

Il est très simple, mais montre un widget très chouette. Ci-dessous, voici leur code.

Comme vous pouvez le voir, les trois premières lignes sont des déclarations d'importation. Tout widget que vous utilisez doit être importé et vous devez toujours importer App de kivy.app.

Les huit lignes suivantes sont la classe principale de l'application. La classe est définie, puis une routine appelée build est créée. Vous devrez presque toujours avoir une routine de construction (build) quelque part dans vos programmes Kivy. Ensuite nous définissons un objet racine du widget Accordéon. Ensuite, nous créons cinq éléments AccordionItems et définissons leurs titres. Nous ajoutons ensuite dix étiquettes avec le texte « contenu très grand ». Nous ajoutons ensuite chaque étiquette dans le widget racine (l'accordéon) et, enfin, nous retournons l'objet racine. C'est en substance

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App

class AccordionApp(App):
     def build(self):
          root = Accordion()
          for x in xrange(5):
               item = AccordionItem(title='Title %d' % x)
               item.add_widget(Label(text='Very big content\n' * 10))
               root.add_widget(item)
          return root

if __name__ == '__main__':
     AccordionApp().run()
```

ce qu'affiche l'objet racine dans la fenêtre que crée Kivy. En dernier lieu, nous avons la déclaration « if_name_ » puis exécutons l'application.

Allez-y, exécutez cet exemple pour voir ce qu'il fait.

Dans quelques instants, vous verrez une fenêtre s'ouvrir avec cinq barres verticales à l'intérieur. En cliquant sur une barre, on provoque son ouverture ce qui révèle les dix étiquettes. Bien sûr, chaque barre a le même texte sur les dix étiquettes, mais vous êtes capable de changer cela.

Le widget Accordéon peut être utilisé pour toutes sortes de choses, mais la chose qui me vient toujours à l'esprit est un écran de configuration… chaque barre étant un ensemble distinct de configurations.

Ensuite, nous allons prendre l'exemple textalign.py. Il n'est pas aussi « sexy » que le précédent, mais c'est un bon exemple qui vous fournit des informations importantes pour la suite.

Avant de regarder le code, exécutez le programme.

Vous devriez voir une étiquette en haut de la fenêtre, un ensemble de neuf cases rouges avec du texte dans une grille 3×3 et quatre boutons le long du bas de la fenêtre. Lorsque vous cliquez (appuyez) sur chacun des boutons, l'alignement du texte dans les cases rouges changera. La raison principale pour laquelle il faut examiner cet exemple attentivement est qu'il vous montre comment utiliser et contrôler certains des widgets importants, ainsi que la façon de changer l'alignement de vos widgets, ce qui n'est pas totalement intuitif.

Le code de cet exemple se trouve ci-dessus à droite. Je vais le décortiquer. D'abord le code d'importation (en haut à droite).

Ci-dessous se trouve quelque chose de spécial. Ils ont créé une classe qui ne contient aucun code. Je vais en parler dans quelques minutes :

```
class BoundedLabel(Label):

  pass
```

Ensuite, une classe appelée « Selector » (ci-dessous) est créée :

```
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.gridlayout import GridLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.properties import ObjectProperty
```

```
class Selector(FloatLayout):

app = ObjectProperty(None)
```

Maintenant, la classe Application est créée.

C'est ici que la routine de sélection est créée. Un widget GridLayout est créé (appelée grid ou grille), qui dispose de 3 lignes et 3 colonnes. Cette grille va contenir les neuf cases rouges.

```
for valign in ('bottom',
'middle', 'top'):

for halign in ('left',
'center', 'right'):
```

Ici nous avons deux boucles, l'une extérieure et l'autre intérieure.

```
label = BoundedLabel(text='V:
%s\nH: %s' % (valign,
halign),

size_hint=(None, None),

halign=halign, valign=valign)
```

Dans le code ci-dessus, une instance du widget BoundedLabel est créée, une fois pour chacune des neuf cases rouges. Vous voudrez peut-être m'arrêter là et dire : « Mais attendez ! Il n'y a pas de widget BoundedLabel. Il a juste une déclaration pass. » Eh bien, oui et non. Nous créons une instance d'un widget personnalisé. Comme je le disais un peu plus haut, nous en parlerons plus en détail dans un instant.

Dans le bloc de code (en haut à droite de la page suivante), nous examinons la variable « case », qui est passée à la routine de sélection.

Ici, on retire la grille pour effacer l'écran.

```
class TextAlignApp(App):

     def select(self, case):

     grid = GridLayout(rows=3, cols=3, spacing=10, size_hint=(None, None),
                  pos_hint={'center_x': .5, 'center_y': .5})
```

```
if self.grid:

self.root.remove_widget(self.
grid)
```

La méthode bind ici définit la taille et la grille est ajoutée à l'objet racine.

```
grid.bind(minimum_size=grid.s
etter('size'))

self.grid = grid

self.root.add_widget(grid)
```

Rappelez-vous, dans le dernier exemple j'ai dit que vous devrez presque toujours utiliser une routine de construction. Voici celle de cet exemple. L'objet racine est créé avec un widget FloatLayout. Ensuite (au milieu à droite) nous appelons la classe Selector pour créer un objet de sélection, qui est ensuite ajouté à l'objet racine et on initialise l'affichage en appelant self.select(0).

Enfin, l'application est autorisée à fonctionner.

```
TextAlignApp().run()
```

Maintenant, avant d'aller plus loin, nous devons éclaircir quelques petites choses. Tout d'abord, si vous regardez dans le dossier qui contient le fichier .py, vous remarquerez un autre fichier appelé textalign.kv. Ceci est un fichier spécial utilisé par Kivy pour vous permettre de créer vos propres widgets et règles. Lorsque votre application Kivy démarre, elle cherche dans le même répertoire le fichier d'aide .kv. S'il est présent, elle le charge en premier. Voici le code dans le fichier .kv.

Cette première ligne indique la version minimum de Kivy qui doit être utilisée pour exécuter cette application.

```
#:kivy 1.0
```

Ensuite le widget BoundedLabel est créé. Chacune des cases rouges dans l'application est un BoundedLabel.

Color définit la couleur d'arrière-plan de la boîte à rouge (rgb: 1,0,0). Le widget Rectangle crée un rectangle (vous l'aurez deviné). Lorsque nous appelons le widget BoundedLabel dans le code de l'application, nous passons une étiquette en tant que parent. La taille et la position (ici dans le fichier .kv) sont réglées à la taille et la position de l'étiquette.

Ici (à droite sur la page suivante) le widget de sélection est créé. Ce sont les quatre boutons qui apparaissent au bas de la fenêtre ainsi que l'étiquette sur le haut de la fenêtre.

```
if case == 0:
                    label.text_size = (None, None)
          elif case == 1:
                    label.text_size = (label.width, None)
          elif case == 2:
          label.text_size = (None, label.height)
          else:
          label.text_size = label.size
                    grid.add_widget(label)
```

```
     def build(self):
     self.root = FloatLayout()
          self.selector = Selector(app=self)
     self.root.add_widget(self.selector)
     self.grid = None
          self.select(0)
     return self.root
```

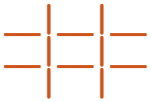
```
<BoundedLabel>:
     canvas.before:
          Color:
               rgb: 1, 0, 0
     Rectangle:
          pos: self.pos
               size: self.size
```

Remarquez que l'étiquette qui constitue le titre en haut de la fenêtre a une position (pos_hint) en haut, a une hauteur de 50 pixels et une taille de police de 16. Chacun des boutons a un texte aligné au centre. La déclaration on_release est une déclaration qui ressemble à « bind », qui fait que, lorsque le bouton est relâché, il appelle (dans ce cas) root.app.select avec la valeur « case ».

J'espère que tout commence à devenir plus clair maintenant. Vous pouvez voir pourquoi Kivy est si puissant.

Examinons un instant deux widgets dont je n'ai pas parlé ci-dessus pendant la discussion du code de l'application, GridLayout et FloatLayout.

GridLayout est un widget parent qui utilise une description en ligne et colonne qui permet de placer les widgets dans chaque cellule. Dans ce cas, il s'agit d'une grille 3×3 (comme un plateau de Tic-Tac-Toe).



Lorsque vous voulez placer un widget dans un GridLayout, vous utilisez la méthode add_widget. Il y a cependant un problème. Vous ne pouvez pas spécifier la cellule de la grille dans laquelle chaque commande sera placée autrement que par l'ordre dans lequel vous les ajoutez. En outre, chaque widget est ajouté de gauche à droite et de haut en bas. Vous ne pouvez pas avoir une cellule vide. Bien sûr, vous pouvez tricher. À vous de comprendre comment.

Le widget FloatLayout semble être juste un conteneur parent pour d'autres widgets enfants.

J'ai passé sous silence quelques points pour l'instant. Cette fois, mon intention était tout simplement de vous enthousiasmer par les possibilités qu'offre Kivy. Dans les prochains articles, nous allons continuer à explorer ce que Kivy peut nous apporter, comment utiliser divers widgets et comment créer un APK pour publier nos applications sur Android.

En atendant, explorez davantage les exemples de Kivy et, surtout, allez sur la page de documentation de Kivy : http://kivy.org/docs/.

```
<Selector>:
      Label:
      pos_hint: {'top': 1}
            size_hint_y: None
      height: 50
            font_size: 16
      text: 'Demonstration of text valign and halign'
      BoxLayout:
            size_hint_y: None
      height: 50
            ToggleButton:
                  halign: 'center'
            group: 'case'
                  text: 'label.text_size =\n(None, None)'
                  on_release: root.app.select(0)
                  state: 'down'
      ToggleButton:
                  halign: 'center'
            group: 'case'
                  text: 'label.text_size =\n(label.width, None)'
                  on_release: root.app.select(1)
      ToggleButton:
                  halign: 'center'
                  group: 'case'
            text: 'label.text_size =\n(None, label.height)'
                  on_release: root.app.select(2)
      ToggleButton:
                  halign: 'center'
                  group: 'case'
            text: 'label.text_size =\n(label.width, label.height)'
                        on_release: root.app.select(3)
```

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

TUTORIEL

Débuter Python - Partie 36

Greg Walters

Avant de commencer, je tiens à souligner que cet article marque les trois ans de la série d'articles sur la programmation Python pour débutants. Je tiens à remercier Ronnie et l'ensemble des personnes du magazine Full Circle pour leur soutien et surtout vous, les lecteurs. Je n'ai jamais pensé que cela continuerait aussi longtemps.

Je tiens aussi à saisir l'occasion de faire une brève remarque au sujet de quelques commentaires flottant dans l'air, suggérant que, après trois ans, le mot « débuter » est peut être déplacé dans le titre de cette série. Après tout, au bout de trois ans, en êtes-vous toujours à débuter ? Eh bien, à certains niveaux, je suis d'accord. Cependant, je reçois encore des commentaires de lecteurs disant qu'ils viennent de découvrir la série et le magazine Full Circle, et qu'ils sont maintenant en train de lire toute la série à partir du début. Ces gens sont donc bien des débutants. Quoi qu'il en soit, à partir de la partie 37, nous enlèverons « débuter » du titre de la série.

Maintenant place au contenu de cet article… la suite sur Kivy.

Imaginez que vous jouez de la guitare. Pas de la « air guitare » (en faisant semblant), mais avec une guitare réelle. Cependant, vous n'êtes pas un très bon joueur de guitare et quelques accords vous posent problème. Par exemple, vous connaissez les accord standards de Do, Mi, Sol, Fa, mais quelques accords – comme Fa# mineur ou Do# mineur – bien que faisables, sont difficiles à faire pendant un morceau rapide. Que faites-vous, surtout si le concert est dans seulement quelques semaines et que vous devez être au point AUJOURD'HUI ? Vous pouvez contourner ce problème en utilisant le capo (cette drôle de pince que vous voyez parfois sur le manche de la guitare). Cela relève la tonalité de la guitare et vous utilisez alors des accords différents pour être comme le reste du groupe. C'est ce qu'on appelle la transposition. Parfois, vous pouvez transposer à la volée dans votre tête. Parfois, il est plus facile de s'asseoir et de travailler sur le papier ; par exemple, si l'accord est Fa# mineur et que vous placez le capo sur la frette 2, vous pouvez simplement jouer un Mi mineur. Mais cela prend du temps. Fabriquons une application qui vous permettra tout simplement de faire défiler la position sur les frettes pour trouver les accords les plus faciles à jouer.

Notre application va être assez simple. Une étiquette de titre, un bouton avec la gamme de base comme texte, une vue défilante « scrollview » (un widget parent merveilleux qui contient d'autres commandes et vous permet de « lancer » ce qu'il contient pour le faire défiler) contenant un certain nombre de boutons qui ont des gammes repositionnées comme texte et un bouton de sortie. Cela ressemblera À PEU PRÈS au texte ci-dessous.

Commencez avec un nouveau fichier Python nommé main.py. Ce nom sera important si/quand vous décidez de créer une application Android avec Kivy. Maintenant, nous allons ajouter nos instructions d'importation qui sont indiquées en haut à droite de la page suivante.

Remarquez la deuxième ligne, « kivy.require('1.0.8') ». Cela vous permet de vous assurer que vous pouvez utiliser les fonctionnalités les plus récentes et les meilleures fournies par Kivy. Notez également que nous incluons une instruction système pour quitter (ligne 3). Plus tard nous aurons un bouton pour quitter.

Voici le début de notre classe appelée « Transpose ».

```
                    Transposer Ver 0.1
-------------------------------------------------------------

 1|  C  C#/Db  D  D#/Eb  E  F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C
 2|  C  C#/Db  D  D#/Eb  E  F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db  D
     D#/Eb  E  F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db D
-------------------------------------------------------------
```

```
class Transpose(App):
    def exit(instance):
        sys.exit()
```

Maintenant, travaillons sur notre routine « build » (au milieu à droite). Elle est nécessaire pour toutes les applications Kivy.

Cela semble confus. Malheureusement, l'éditeur ne garde pas toujours les espaces correctement, même avec une police à espacement fixe. L'idée est que la chaîne texte1 est une simple gamme commençant par la note « Do ». Chacune doit être centrée dans 5 espaces. Comme le texte affiché en bas à droite.

La chaîne texte2 devrait être la même chose, mais répétée. Nous allons utiliser un décalage dans la chaîne texte2 pour remplir le texte du bouton à l'intérieur du widget scrollview.

Nous créons maintenant l'objet racine (qui est notre fenêtre principale) contenant un widget GridLayout. Si vous vous souvenez, il y a TRÈS longtemps, quand nous faisions d'autres développements d'interfaces pour Glade, il y avait un widget grille (« grid view »). Eh bien, le GridLayout ici est à peu près la même chose. Dans notre cas, nous avons une grille qui contient une colonne et trois lignes. Dans chaque cellule de la grille, on peut mettre d'autres widgets. Rappelez-vous, nous ne pouvons pas choisir où va chaque widget autrement que par l'ordre dans lequel on les ajoute.

```
racine =
GridLayout(orientation='verti
cal', spacing=10,
cols=1,rows=3)
```

Dans ce cas, la représentation est la suivante :

```
-----------------------------
(0)   étiquette titre
-----------------------------
(1)   bouton principal
-----------------------------
(2)   scrollview
-----------------------------
```

La vue scrollview contient plusieurs éléments ; dans notre cas ce sont des boutons. Ensuite, on crée l'étiquette qui sera tout en haut de notre application.

```
etiquette =
Label(text='Transposer Ver 0.1',
    font_size=20,
    size_hint=(None,None),
    size=(480,20),
    padding=(10,10))
```

```
import kivy
kivy.require('1.0.8')
from sys import exit
from kivy.app import App
from kivy.core.window import Window
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.gridlayout import GridLayout
```

Les propriétés qui sont définies devraient être assez explicites. Les seules qui pourraient vous poser problème sont celles de « padding » (remplissage) et de « size_hint ». Le remplissage est le nombre de pixels autour de l'élément dans un repère x,y. On lit dans la documentation Kivy que « size_hint » (pour X, mais c'est identique pour Y) est défini comme suit :

X size hint. Représente la quantité d'espace que le widget doit utiliser selon la direction de l'axe X, par rapport à la largeur de son parent. Seuls Layout et Window utilisent cette propriété. La valeur est indiquée en pourcentage sous forme d'un nombre compris entre 0 et 1, où 1 signifie la taille totale de son parent et 0,5 représente 50 %.

Dans notre cas, size_hint est défini à « none » (aucun), qui vaut par défaut

```
    def build(self):
        #-------------------------------------------------------------
        text1 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C"
        text2 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db  D
D#/Eb  E    F  F#/Gb  G  G#/Ab  A  A#/Bb  B    C  C#/Db"
        #-------------------------------------------------------------
```

```
|    |    |    |    |    |    |    |    |    |    |    |    |
123456789012345678901234567890123456789012345678901234567890123456
  C  C#/Db  E    F  F#/Gb  G  G#/Ab  A  A#/Bb  B    C
```

100 % ou 1. Ce sera plus important (et compliqué) plus tard.

Maintenant, nous définissons notre bouton « principal » (en haut à droite). Il s'agit d'une référence statique pour la gamme.

Encore une fois, tout devrait être assez clair.

Maintenant, nous ajoutons les widgets à l'objet racine, qui est le widget GridLayout. L'étiquette va dans la première cellule, le bouton (btn1) va dans la seconde.

```
#----------------------------
racine.add_widget(etiquette)
racine.add_widget(btn1)
#----------------------------
```

Arrive maintenant du code plus difficile à comprendre. Nous créons un autre objet GridLayout et l'appelons « s ». Nous le lions ensuite à la hauteur du widget suivant qui, dans ce cas, se trouve être la ScrollView, PAS les boutons.

```
s = GridLayout(cols=1,
spacing = 10, size_hint_y =
None)
s.bind(minimum_height=s.sette
r('height'))
```

Maintenant (au milieu à droite), nous créons 20 boutons, remplissons la propriété « texte », puis les ajoutons au GridLayout.

Maintenant nous créons le ScrollView, définissons sa taille, et l'ajoutons au Grid-Layout racine.

```
sv =
ScrollView(size_hint=(None,
None), size=(600,400))

sv.center =
Window.center

racine.add_widget(sv)
```

Enfin, nous ajoutons le GridLayout, qui contient tous nos boutons dans le ScrollView, et retournons l'objet racine à l'application.

```
sv.add_widget(s)

return racine
```

Enfin, nous avons notre routine « if_name_ ». Remarquez que nous nous réservons la possibilité d'utiliser l'application comme une application android.

```
if __name__ in
('__main__','__android__'):

    Transpose().run()
```

Maintenant, vous vous demandez peut-être pourquoi j'ai utilisé des boutons au lieu d'étiquettes pour tous nos objets textuels. C'est parce que les étiquettes dans Kivy n'ont aucune sorte de bordure visible par défaut. Nous jouerons avec cela dans le prochain épisode. Nous allons également ajouter un bouton pour quitter et d'autres petites choses.

```
    btn1 = Button(text = "  " + text1,size=(680,40),
           size_hint=(None, None),
           halign='left',
           font_name='data/fonts/DroidSansMono.ttf',
           padding=(20,20))
```

```
for i in range(0,19):
    if i <= 12:
        if i < 10:
            t1 = " " + str(i) + "| "
        else:
            t1 = str(i) + "| "
    else:
        t1 = ''
        text2 = ''
    btn = Button(text = t1 + text2[(i*5):(i*5)+65],
          size=(680, 40),
          size_hint=(None,None),
          halign='left',
          font_name='data/fonts/DroidSansMono.ttf')
    s.add_widget(btn)
#--------------------------------------------------------------
```

Le code source peut être trouvé sur Pastebin :

http://pastebin.com/8jTJSmLR

Jusqu'à la prochaine fois, amusez-vous et je vous remercie de m'avoir suivi pendant trois ans !

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

TUTORIEL

Programmer en Python - Partie 37

Greg Walters

Ce mois-ci, nous allons terminer le programme de transposition que nous avons écrit dans Kivy. J'espère que vous avez enregistré le code de la dernière fois, parce que nous allons le compléter. Sinon, récupérez-le sur le FCM n° 64.

Commençons par récapituler ce que nous avons fait le mois dernier. Nous avons créé une application qui permet à un guitariste de transposer rapidement d'une clé à une autre. Le but ultime est de pouvoir exécuter cette application non seulement sous Linux ou Windows, mais également sur un appareil Android. Je l'emporte sur ma tablette quand je vais répéter avec mon groupe. Je me préparais à conditionner notre projet pour Android, mais certaines choses ayant changé dans la méthode pour le faire, nous verrons cela le mois prochain.

L'application, telle que nous l'avons laissée la dernière fois, ressemblait à ce qui est ci-dessous à gauche.

Lorsque nous aurons terminé, elle devrait ressembler à l'écran ci-dessous à droite.

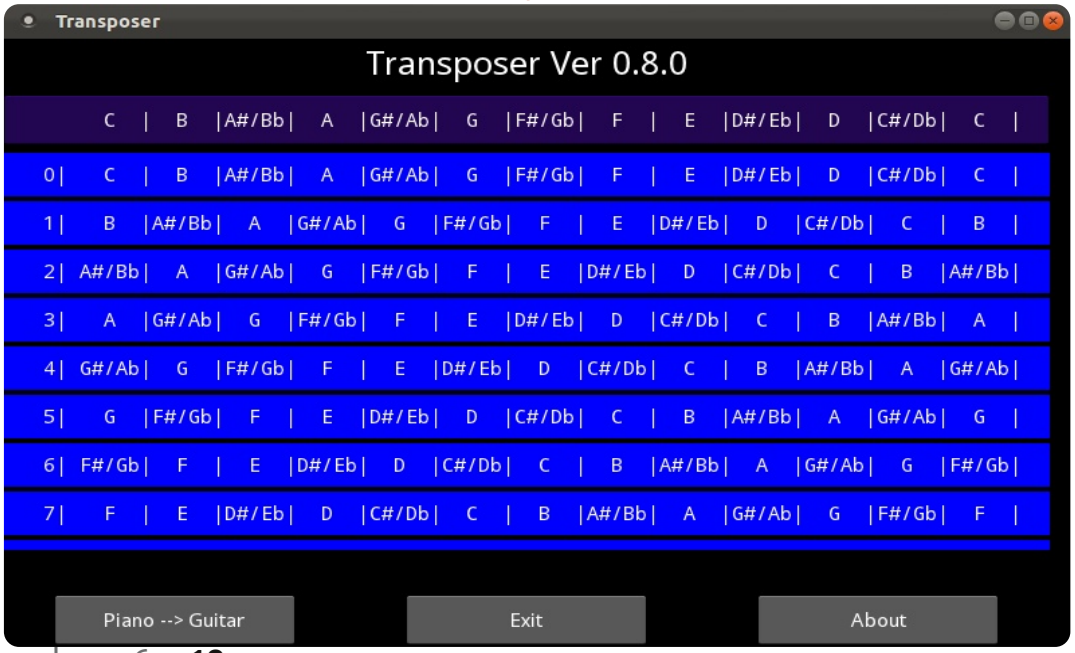
La première chose que vous remarquerez est qu'il y a des étiquettes bleues à la place de celles qui étaient grises et tristes. La suivante est qu'il y a trois boutons. Enfin, les étiquettes qui défilent sont plus proches de la largeur totale de la fenêtre. À part ça, c'est à peu près la même chose (visuellement). L'un des boutons est un bouton « à propos » qui affichera quelques informations simples, pour vous montrer comment faire un pop-up simple. Un autre bouton sert à quitter. Le troisième bouton va remplacer l'étiquette de texte pour faciliter la transposition de piano à guitare ou vice-versa.

```
#:kivy 1.0
#:import kivy kivy

<BoundedLabel>:
      canvas.before:
      Color:
            rgb: 0, 0, 1
      Rectangle:
            pos: self.pos
            size: self.size
```

Nous allons commencer par créer un fichier .kv (ci-dessus). C'est ce qui va nous donner les étiquettes colorées. C'est un fichier très simple.

Les deux premières lignes sont né-



cessaires. Elles indiquent simplement quelle version de Kivy est requise. Ensuite, nous créons un nouveau type d'étiquette appelé « BoundedLabel ». La couleur est réglée avec des valeurs RVB (entre 0 et 1, ce dernier représentant 100 %), et comme vous pouvez le voir, la valeur de bleu est fixée à 100 pour cent. Nous allons également créer un rectangle qui est l'étiquette réelle. Enregistrez ce fichier sous le nom « transpose.kv ». Vous devez prendre le même nom que la classe qui va l'utiliser.

Maintenant que ceci est terminé, ajoutez les lignes suivantes juste avant la classe transpose dans le fichier source de la dernière fois :

```
class BoundedLabel(Label):

    pass
```

Pour que cela fonctionne, il suffit d'une définition. Avant d'aller plus loin, ajoutez la ligne suivante à la section des importations :

```
from kivy.uix.popup import
Popup
```

Cela nous permettra de créer le popup plus tard. Maintenant, dans la classe Transpose, juste à l'intérieur de la routine build, placez le code ci-dessus à droite.

```
def ChargeEtiquettes(w):
    if w == 0:
            tex0 = self.texte1
            tex1 = self.texte2
        else:
            tex0 = self.texte3
            tex1 = self.texte4
        for i in range(0,22):
            if i <= 12:
               if i < 10:
                  t1 = " " + str(i) + "| "
               else:
                  t1 = str(i) + "| "
                  t = tex1
            else:
                t1 = ''
                t = ''
            l = BoundedLabel(text=t1+t[(i*6):(i*6)+78], size=(780, 35),
                size_hint=(None, None),halign='left',
                font_name='data/fonts/DroidSansMono.ttf')
            s.add_widget(l)
```

La routine ChargeEtiquettes nous donnera les étiquettes de couleur (BoundedLabel) et la capacité d'échange. Vous avez pratiquement tout vu la dernière fois. Nous passons une valeur au paramètre « w » pour déterminer quel texte est affiché. La ligne l=BoundedLabel est à peu près la même que la dernière fois, sauf que, cette fois, nous utilisons un widget BoundedLabel au lieu d'un widget Bouton. Les « ChargeEtiquettes » seront principalement appelés depuis la routine suivante, Echange. Placez ce code (à droite) en dessous de ChargeEtiquettes.

```
def Echange(instance):
    if self.quelsens == 0:
        self.quelsens = 1
        btnSens.text = "Guitare --> Piano"
        btn1.text = "    " + self.texte3
        s.clear_widgets()
        ChargeEtiquettes(1)
    else:
        self.quelsens = 0
        btnSens.text = "Piano --> Guitare"
        btn1.text = "    " + self.texte1
        s.clear_widgets()
        ChargeEtiquettes(0)
```

```
self.quelsens=0
self.texte1 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |"
self.texte2 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Ab|  D  |C#/Db|  C  |"
self.texte3 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |"
self.texte4 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|"
```

Vous pouvez voir que cette routine est assez explicite. Nous utilisons une variable (self.quelsens) pour déterminer « dans quel sens » les étiquettes s'affichent… de Guitare vers Piano ou de Piano vers Guitare.

Assurez-vous de sauvegarder votre travail dès à présent, car nous allons faire beaucoup de changements à partir de maintenant.

Remplacez les lignes définissant texte1 et texte2 par les lignes ci-dessus (tableau en haut de page).

Nous réglons self.quelsens à 0 qui sera notre valeur par défaut pour la procédure d'échange. Ensuite, nous définissons quatre chaînes au lieu des deux que nous avions la dernière fois. Vous remarquerez peut-être que les chaînes texte3 et texte4 sont en fait texte1 et texte2 à l'envers.

Maintenant, nous allons adapter la définition de la ligne racine. Changez-la de :

```
root =
GridLayout(orientation='verti
cal', spacing=10,
cols=1,rows=3)
```

à :

```
root =
GridLayout(orientation='verti
cal', spacing=6, cols=1,
rows=4,
row_default_height=40)
```

Nous avons changé l'espacement de 10 à 6 et réglé la hauteur de ligne par défaut à 40 pixels. Changez le texte de l'étiquette (ligne suivante) en « text='Transposer Ver 0.8.0' ». Pour le reste, rien n'a changé sur cette ligne.

Maintenant changez la définition du bouton de :

```
btn1 = Button(text = "   " +
text1,size=(680,40),

  size_hint=(None,None),

  halign='left',

font_name='data/fonts/DroidSa
nsMono.ttf',

  padding=(20,20))
```

à :

```
btn1 = Button(text = "      "
+ self.text1,size=(780,20),

  size_hint=(None, None),

  halign='left',

font_name='data/fonts/DroidSa
nsMono.ttf',

  padding=(20,2),

background_color=[0.39,0.07,.
92,1])
```

Remarquez que j'ai changé le format de la première définition pour plus de clarté. Les gros changements sont la taille qui passe de 680,40 à 780,20 et la couleur de fond du bouton. Rappelez-vous, on peut changer la couleur de fond pour les boutons, mais pas pour les étiquettes « standards ».

Ensuite, nous allons définir trois widgets AnchorLayout pour les trois boutons que nous ajouterons plus tard. Je les ai nommés al0 (AnchorLayout0), al1 et al2. Nous ajoutons également le code pour le Popup « à propos » et définissons nos boutons avec les paramètres de liaison (bind). Ceci est illustré à la page suivante, en haut à gauche.

Trouvez la ligne « s = GridLayout » et modifiez l'espacement de 10 à 4. Ensuite, ajoutez la ligne suivante après la ligne s.bind (juste avant la boucle for) :

```
ChargeEtiquettes(0)
```

Ceci appelle la routine ChargeEtiquettes avec notre « quelsens » par défaut qui vaut 0.

Ensuite, commentez la totalité du code de la boucle for. Cela commence par « for i in range(0,19): » et se termine par « s.add_widget(btn) ». Nous n'avons pas besoin de cette routine puisque ChargeEtiquettes le fait pour nous.

Maintenant, enregistrez votre code

```
al0 = AnchorLayout()
al1 = AnchorLayout()
al2 = AnchorLayout()
popup = Popup(title='A propos de Transposer',
        content=Label(text='Ecrit par G.D. Walters'),
        size_hint=(None,None),size=(400,400))
btnSens = Button(text = "Piano --> Guitare",
         size=(180,40),size_hint=(None,None))
btnSens.bind(on_release=Echange)
btnAPropos=Button(text="A propos",
         size=(180,40),size_hint=(None,None))
btnAPropos.bind(on_release=AfficheAPropos)
btnQuitter =Button(text="Quitter",
         size=(180,40),size_hint=(None,None))
btnQuitter.bind(on_release=exit)
```

et essayez de l'exécuter. Vous devriez voir un bouton violet foncé en haut et nos BoundLabels d'un joli bleu. De plus, vous remarquerez que les BoundLabels dans la fenêtre de défilement sont plus rapprochés, ce qui en facilite grandement la lecture.

Nous sommes presque au bout de notre code, mais il nous reste quelques petites choses à faire. Après la ligne « sv = ScrollView », ajoutez la ligne suivante :

```
sv.size = (720, 320)
```

Cela définit la taille du widget ScrollView à 720 sur 320, ce qui le rend plus large à l'intérieur de la fenêtre racine. Maintenant, avant la ligne « return racine », ajoutez le code que vous voyez en haut à droite.

Ici, nous ajoutons les trois boutons aux widgets AnchorLayout, créons un GridLayout pour contenir les AnchorLayouts et enfin ajoutons les AnchorLayouts au GridLayout.

Retournez juste en dessous de la routine « def Echange » et ajoutez ce qui suit :

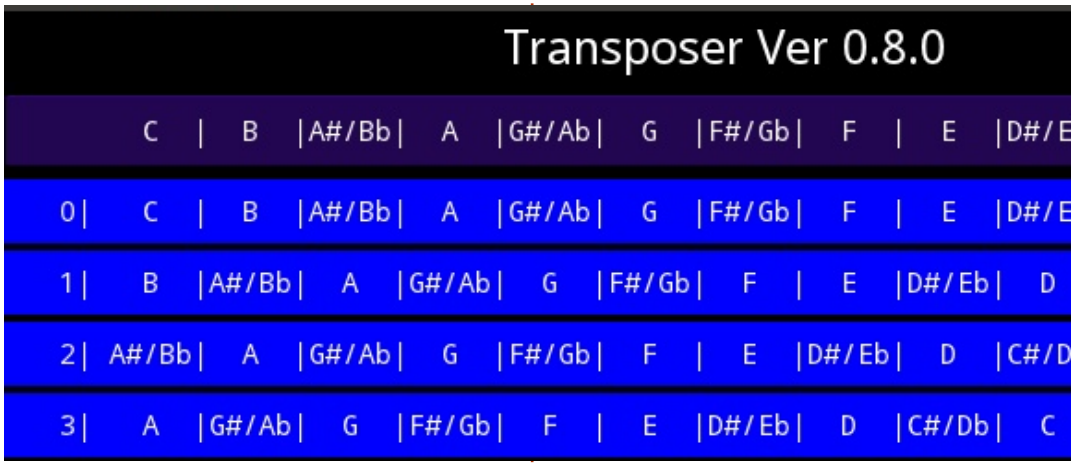
```
def AfficheAPropos(instance):

  popup.open()
```

C'est tout. Enregistrez et exécutez le code. Si vous cliquez sur le bouton « À propos », vous verrez le popup tout simple. Cliquez n'importe où en dehors du popup pour le faire disparaître.

Maintenant, notre code est écrit. Vous pouvez trouver le code complet

```
        al0.add_widget(btnSens)
        al1.add_widget(btnQuitter)
        al2.add_widget(btnAPropos)
        bgl = GridLayout(orientation='vertical',
              spacing=6, cols=3,rows=1,
              row_default_height=40)
        bgl.add_widget(al0)
        bgl.add_widget(al1)
        bgl.add_widget(al2)
```



ici : http://pastebin.com/T0kJ0q5z

Ensuite, nous devons créer notre paquet Android… mais cela devra attendre la prochaine fois.

Si vous voulez vous préparer et essayer d'empaqueter pour Android avant le mois prochain, allez sur http://kivy.org/docs/guide/packaging-android.html pour trouver la documentation à ce sujet. Assurez-vous de suivre attentivement la documentation.

Rendez-vous le mois prochain.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille. Son site web est www.thedesignatedgeek.net.

TUTORIEL

Écrit par Greg Walters

Programmer en Python : Partie 38

Comme je l'ai promis la dernière fois, nous allons reprendre l'application de transposition que nous avons écrite et créer un APK pour l'installer sur votre appareil Android.

Avant de commencer, assurons-nous que tout soit prêt. La première chose dont nous avons besoin est de placer les deux fichiers que nous avons créés la dernière fois dans un dossier auquel vous pouvez facilement accéder. Appelons-le « transposer ». Créez-le dans votre répertoire personnel. Ensuite, copiez les deux fichiers (transpose.kv et transpose.py) dans ce dossier. Maintenant renommez transpose.py en main.py. Cette partie est importante.

Ensuite, nous avons besoin de faire référence aux instructions d'empaquetage de Kivy dans un navigateur Web. Le lien est http://kivy.org/docs/guide/packaging-android.html. Nous allons utiliser ceci pour les prochaines étapes, mais pas exactement comme les gens de Kivy l'ont prévu. Vous devez avoir le SDK android de notre leçon précédente. Idéalement, vous devriez y retourner et récupérer tous les logiciels qui y sont listées, mais pour nos besoins, il vous suffit de suivre ce qui est indiqué ici. Vous aurez besoin de télécharger le logiciel python-for-android. Ouvrez une fenêtre de terminal et tapez ce qui suit :

```
git clone
git://github.com/kivy/python-
for-android
```

Ceci va télécharger et installer le logiciel dont nous avons besoin pour continuer. Maintenant, dans un terminal, allez dans le répertoire du dossier python-for-android/dist/default.

Vous allez y trouver un fichier appelé build.py. C'est lui qui va faire tout le travail pour nous. Et maintenant, voici la magie.

Le programme build.py prend divers arguments sur la ligne de commande et créera l'APK pour vous. Ci-dessus se trouve la syntaxe pour build.py prise directement dans la documentation Kivy.

```
./build.py --dir <path to your app>
           --name "<title>"
           --package <org.of.your.app>
           --version <human version>
           --icon <path to an icon to use>
           --orientation <landscape|portrait>
           --permission <android permission like VIBRATE> (multiple allowed)
           <debug|release> <installd|installr|...>
```

Pour nos besoins, nous allons utiliser la commande suivante (le « \ » est un caractère de continuation de ligne) :

```
./build.py --dir ~/transposer
--package
org.RainyDay.transposer \
 --name "RainyDay Transposer"
--version 1.0.0 debug
```

Regardons les morceaux de la commande :

./build.py - c'est l'application ;
--dir ~/transposer - il s'agit du répertoire contenant le code de l'application ;
--package org.RainyDay.transposer - c'est le nom du paquet ;
--name "RainyDay Transposer" - c'est le nom de l'application qui apparaîtra dans la liste des applications ;
--version 1.0.0 - la version de notre application ;
debug - c'est le niveau de sortie (debug ou release).

Une fois que vous aurez exécuté ceci, en supposant que tout a fonctionné comme prévu, vous devriez avoir un certain nombre de fichiers dans le dossier /bin. Celui que vous cherchez est intitulé « RainyDay-Transposer-1.0.0-debug.apk ». Vous pouvez le copier sur votre appareil Android en utilisant votre gestionnaire de fichiers favori, puis l'installer comme n'importe quelle autre application des divers magasins d'applications.

C'est tout le temps dont je dispose pour ce mois-ci.