Petite présentation de l’outil pyUnit

PyUnit est un framework de test unitaire de la famille Xunit pour le langage de programmation Python. Il a été écrit par Steve Purcell et fait partir de la bibliothèque standard de python depuis la version 2.1 du 23 mars 2001 sous le nom d’ «unittest».

Configuration requise

PyUnit est conçu pour fonctionner sur n'importe quel machin ayant une version de python supérieur a la version 1.5.2.

Installation

Les classes nécessaires pour écrire les tests se trouvent dans le module 'unittest' de python. Ce module fait partie de la bibliothèque Python standard pour Python 2.1 et versions ultérieures. Aucune installation n’est donc nécessaire au préalable.

#### Une introduction à TestCases

Les éléments de base des tests unitaires sont des «cas de test» - des scénarios uniques qui doivent être configurés et vérifiés pour leur exactitude. Dans PyUnit, les cas de test sont représentés par la classe TestCase dans le module unittest. Pour créer nos propres cas de test, nous devons écrire des sous-classes (heritage) de la classe TestCase. Une instance d'une classe TestCase est un objet qui peut exécuter complètement une seule méthode de test, avec un code facultatif de configuration et de rangement. Le code de test d'une instance de TestCase doit être entièrement autonome, de sorte qu'il puisse être exécuté de manière isolée ou en combinaison arbitraire avec un nombre quelconque d'autres cas de test.

#### Créer un cas de test simple

La sous-classe de cas de test la plus simple remplacera simplement la methode runTest afin d'exécuter un code de test spécifique:

import unittest

class DefaultWidgetSizeTestCase (unittest.TestCase):

def runTest (self):

widget = Widget ("Le widget")

assert widget.size () == (50,50), 'taille par défaut incorrecte'

Notez que pour tester quelque chose, nous utilisons simplement l'instruction intégrée 'assert' de Python. Si l'assertion échoue lors de l'exécution du scénario de test, un AssertionError sera déclenché et le cadre de test identifiera le scénario de test comme un «échec». D'autres exceptions qui ne résultent pas de vérifications explicites «d'assert» sont identifiées par le cadre de test comme des «erreurs».

#### Classes TestCase avec plusieurs méthodes de test

Souvent, de nombreux petits cas de test utilisent le même appareil. Dans ce cas, nous finirions par sous-classer(heriter) SimpleWidgetTestCase dans de nombreuses petites classes à méthode unique telles que DefaultWidgetSizeTestCase. Cela prend du temps et est décourageant, donc PyUnit fournit un mécanisme plus simple:

import unittest

class WidgetTestCase (unittest.TestCase):

def setUp (self):

self.widget = Widget ("Le widget")

def tearDown (self):

self.widget.dispose ()

self.widget = None

def testDefaultSize (self) :

assert self.widget.size () == (50,50), 'taille par défaut incorrecte'

def testResize (self):

self.widget.resize (100,150)

assert self.widget.size () == (100,150), \

'mauvaise taille après redimensionnement'

Ici, nous n'avons pas fourni de methode runTest, mais avons plutôt fourni deux méthodes de test différentes. Les instances de classe exécuteront désormais chacune l'une des test méthodes, self.widget créées et détruites séparément pour chaque instance. Lors de la création d'une instance, nous devons spécifier la méthode de test à exécuter. Nous faisons cela en passant le nom de la méthode dans le constructeur:

defaultSizeTestCase = WidgetTestCase ("testDefaultSize")

resizeTestCase = WidgetTestCase ("testResize")

#### Exécution de tests de manière interactive

Bien sûr, tout l'intérêt d'écrire ces tests est que nous puissions les exécuter et savoir si notre logiciel fonctionne. Le framework de test utilise les classes 'TestRunner' pour fournir un environnement dans lequel nos tests peuvent s'exécuter. Le TestRunner le plus courant est TextTestRunner, qui peut exécuter des tests et rapporter les résultats sous forme textuelle:

runner = unittest.TextTestRunner ()

runner.run (widgetTestSuite)

Par défaut, TextTestRunner imprime sa sortie vers sys.stderr, mais cela peut être modifié en passant un autre objet semblable à un fichier à son constructeur.

Une telle utilisation de TextTestRunner est un moyen idéal d'exécuter nos tests de manière interactive à partir d'une session d'interpréteur Python.

#### Exécution de tests à partir de la ligne de commande

Le module unittest contient une fonction appelée main, qui peut être utilisée pour transformer facilement un module de test en un script qui exécutera les tests qu'il contient. La fonction main utilise la classe unittest.TestLoader pour rechercher et charger automatiquement les cas de test dans le module actuel.

Par conséquent, si nous nommons nos méthodes de test en utilisant la convention test\* décrite précédemment, nous pouvons placer le code suivant au bas de notre module de test:

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main ()

#### Le testeur de l'interface graphique

Il existe une interface graphique que nous pouvez utiliser pour exécuter nos tests. Il est écrit à l'aide de Tkinter, la boîte à outils de fenêtrage livrée avec Python sur la plupart des plates-formes. Il ressemble à l'interface graphique de JUnit.

Pour utiliser le lanceur de test GUI, exécutez simplement:

% python unittestgui.py