PROGRAMMATION D'ORPHY GTS EN PYTHON

1 - INTRODUCTION – LE LANGAGE PYTHON

1.1 - Python est facile à apprendre et à utiliser

Il possède une syntaxe claire et facile à lire. Avec un bon tutoriel, n'importe qui peut écrire un programme plus ou moins important en fonction de son niveau initial en programmation. Comme Python est un langage interprété, le développement est facilité par l'absence d'une phase de compilation/liens. Ainsi l'apprentissage et la modification en est grandement simplifié.

1.2 - Python est un langage de programmation orienté objets

Dans Python tout est objet: modules, classes (et non seulement leurs instances), fonctions, type pré-défini (entier, réel). Tout dans Python peut être utilisé comme objet, et être adapté à une utilisation plus spécifique.

1.3 - Python est un langage libre et portable

Depuis la version v2.1.1, Python est disponible sous licence GPL. De plus on trouve des portages de Python sur quasiment tous les systèmes d'exploitation : Linux, Unix commerciaux, Windows pour ne citer que les principaux.

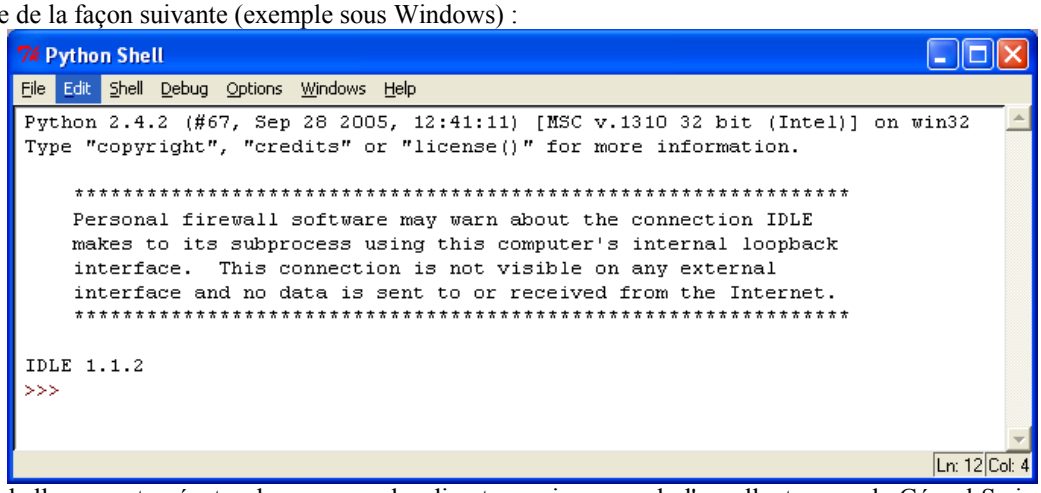
1.4 - Outils pour développer en Python

L'interpréteur Python

L'interpréteur Python est l'arme ultime lors de la programmation en Python. Il permet d'exécuter du code Python directement, comme si on lançait des fonctions directement sur une calculatrice par exemple. Il fonctionne comme un shell traditionnel (par exemple bash, qui est l'interpréteur de commandes par défaut sous Linux et de nombreux Unix).

L'interpréteur Python standard n'est néanmoins pas très évolué : il n'est pas très agréable à utiliser, notamment pour lister les fonctions fournies par un module ou un objet.

Elle se présente de la façon suivante (exemple sous Windows) :



A l'aide de ce shell, on peut exécuter des commandes directes, voir pour cela l'excellent cours de Gérard Swinnen consultable sur http://www.ulg.ac.be/cifen/inforef/swi/python.htm.
Evidemment, on peut développer des programmes beaucoup plus complexes mais l'intérêt ici est de pouvoir récupérer des données provenant d'Orphy-Gts de façon simple.

Éditeur de texte

N'importe quel éditeur de texte peut faire l'affaire, mais un éditeur de texte avec une bonne coloration syntaxique et une gestion de l'indentation est un plus indéniable.

1.5 – Où trouver les fichiers d'installations

Sur http://www.python.org/.

1.5.1 - Installation sous Windows

Il vous faudra installer les fichiers (dans l'ordre) : python-2.4.2.msi puis pywin32-207.win32-py2.4.exe et enfin pyserial-2.2.win32.exe (ce sont les versions que j'ai installé).

1.5.2 – Installation sous Linux

En général, sous Linux, Python est installé en « standard » sinon il faut télécharger python-2.4.2.tgz et pyserial-2.2.zip. L'installation se fait « comme d'habitude sous Linux ». Avec Mandriva Linux (ce que j'utilise), on peut trouver les fichiers RPM pour l'installation.

1.5.3 – Installation sous Mac

Il faut télécharger macpython sur http://www.python.org/.
Je n'ai pas testé macpython et le module pySerial sur Mac.

2- LES CARACTERISTIQUES D'ORPHY-GTS

Pour pouvoir programmer, il faut connaître les caractéristiques d'Orphy-Gts.
Voilà les caractéristiques essentielles :
Orphy GTS est une centrale autonome de commande et de mesures. Elle dispose en interne :

- d'un microprocesseur
- de mémoire vive (8 ko) et morte (32 ko)
- de ports d'entrées/sorties tout ou rien
- de 2 compteurs
- d'un convertisseur analogique/numérique multiplexé (8 entrées) et d'un échantillonneur-bloqueur
- d'un convertisseur numérique/analogique
- d'une liaison série asynchrone

La liaison série se trouve sur la face arrière. La communication se fait de façon asynchrone à 9600 bits/s, 8 bits de données, 1 bit de stop et pas de contrôle de parité.
Les LED en façade remplissent un rôle important : elles sont le témoin de la bonne communication entre l'ordinateur et l'interface.
La liaison série est le support de la communication entre un ordinateur et Orphy. 2 modes de communication sont possibles : le mode rapide et le mode texte.
Le mode rapide permet une commande ou une interrogation depuis un ordinateur avec un échange d'informations de taille minimale entre les deux équipements. Ce mode permet une interrogation avec un minimum de temps consommé dans la communication.
Les commandes issues de l'ordinateur ont une taille fixe de 1 octet.
Orphy selon le cas répondra 0, 1, 2 ou 255 octets. Mais attention : Orphy ne répond que lors d'une interrogation !
Voici un récapitulatif des codes des commandes correspondants :

Codes			**Fonctions**	**Nombre d'octets reçus**	**Equivalence mode texte**
Hexadécimal	*Décimal*	*Caractère équivalent*			
$00 ... $05	0 ... 5		Mise à 0 d'une sortie binaire. Code = 0 + n° de la sortie binaire	0	"XRBIT x\n"
$20 ... $25	32 ... 37	' ' ... '%'	Mise à 1 d'une sortie binaire. Code = 32 + n° de la sortie binaire	0	"XSBIT x\n"
$30 ... $37	48 ... 55	'0' ... '7'	Acquisition de 255 valeurs à intervalles de temps réguliers d'environ 51µs sir l'entrée analogique choisie. Code = 48 + n° de l'entrée analogique	255	
$40 ... $47	64 ... 71	'@' ... 'G'	Lit une entrée analogique. Code = 64 + n° de l'entrée analogique	1	"XEA x\n"
$60 ... $65	96 ... 101	'`' ... 'e'	Lit une entrée binaire. Code = 96 + n° de l'entrée binaire	1	"XEBIT x\n"
$70 ... $7F	112 ... 127	'p' ...	Commande la sortie analogique. Code = 112 → Usa = 0V Code = 127 → Usa = 4,70V Code = 112 + (Usa / 0,313V) pour 0V ≤ Usa ≤ 4,70V	0	"XSA n\n"
$80 ... $BF	128 ... 191	'Ç' ... '+'	Commande les 6 sorties binaires en bloc. Code = 128 + code des sorties binaires	0	"XSBLOC b\n"
$C0	192	'+'	Lit les 6 entrées binaires en bloc. Code = 192	1	"XEBLOC\n"
$D0 ... $D1	208 ... 209	'ð' ... 'Ð'	Compte les changements d'états de l'entrée front choisie et remet à zéro le compteur interne. Code = 208 + n° de l'entrée front	2	"XCPT x\n"

Notes :

- Les \n dans les chaînes de caractères représentent le code du "Retour Chariot" (code ASCII 13).
- Dans le cas d'une réponse d'un octet, la valeur pourra varier de :
- 0 à 255 pour la lecture d'une entrée analogique,
- 0 à 1 pour la lecture d'une entrée binaire,
- 0 à 63 pour la lecture en bloc de toutes les entrées binaires.
- Pour les entrées front, la réponse est sur 16 bits (nombre de fronts compris entre 0 et 65535). Le premier octet représente le poids fort, le second le poids faible de la donnée.

3 - RECUPERER DES DONNEES PROVENANT D'ORPHY GTS

3.1 – Le premier programme

Utiliser le bloc notes ou tout autre traitement de texte pour taper le texte ci-dessous :

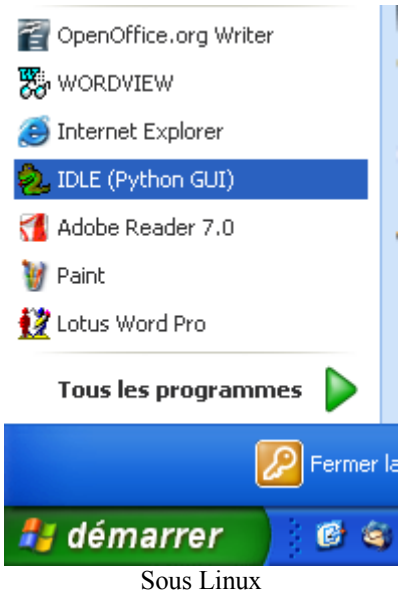
Remarques :

Sous Windows, le port COM1 se notera se.port=0, le port COM2 se notera se.port=1.
Sous Linux, le port COM1 se notera se.port=/dev/ttyS0, le port COM2 se notera se.port=/dev/ttyS1
Le texte derrière les caractères # est considéré comme du commentaire par Python.

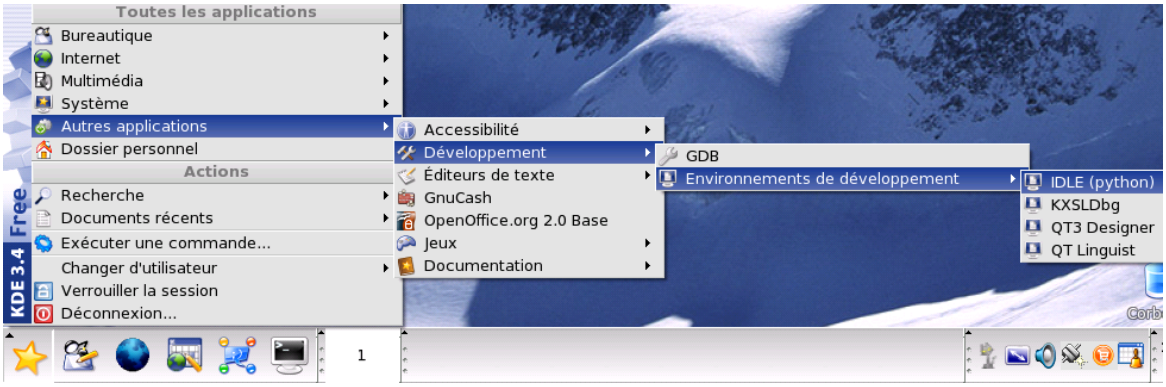
```
# -*- coding:Latin-1 -*-
import serial                #importe le module serial pour communiquer avec la voie série
se = serial.Serial()         #définition de la variable se communiquant avec le port série
se.baudrate = 9600           #sélection de la vitesse de transmission à 9600 bauds
se.bytesize=8                #Transmission des informations sur 8 bits
se.parities=0                #sans parité
se.stopbits=1                #bit de stop 1
se.xonxoff=0                 #pas de contrôle logiciel
se.rtscts=0                  #pas de contrôle RTS/CTS
se.timeout=1                 #timeout (important)
se.port = 0                  #le port 0 est ici COM1, COM2 serait le port 1
print se.portstr             #affiche le nom du port ouvert
se.open()                    #ouvre la voie série sélectionnée
print se                     #Affiche l'état complet de la voie série
y=se.write("@")              #requête sur la voie série @ pour EA0, A pour EA1, B pour EA2, C pour EA3... G pour EA7
x=se.read()                  #lecture de la voie série
print x                      #affichage du caractère de lecture
t=ord(x)                     #conversion de x en numérique
print t                      #affichage en nombre (0 à 255)
se.close()                   #fermeture de la voie série
```

Enregistrer ce programme avec l'extension .py serie.py par exemple puis lancer l'interpréteur Python.

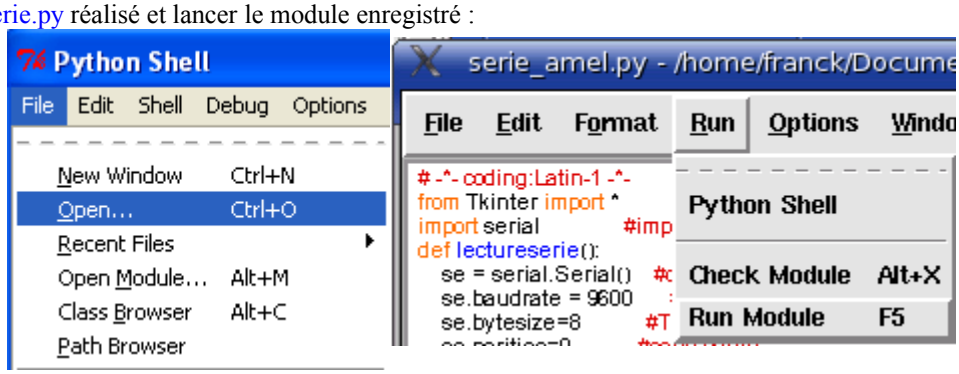
Sous Windows



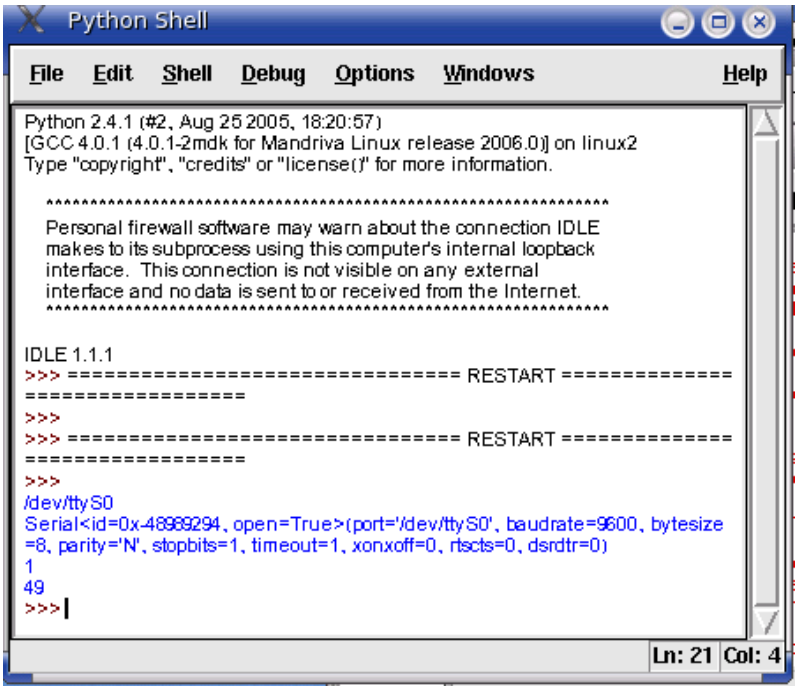
Sous Linux



Vérifier que l'interface Orphy GTS est connectée et sous tension.
Ouvrir le fichier serie.py réalisé et lancer le module enregistré :



Remarque : si le programme indique une erreur, vérifier la voie série utilisée et/ou « faire » un RAZ sur l'interface.
Résultat obtenu :



Commentaire des résultats obtenus

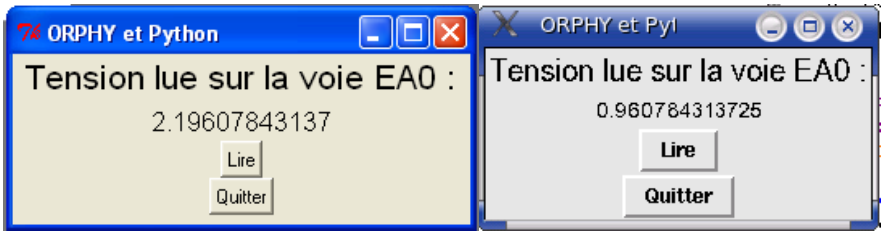
Première ligne	voie série utilisée
Deuxième et troisième ligne	état de la voie série utilisée
Quatrième ligne	le caractère reçu est 1
Cinquième ligne	valeur convertie en 49 en utilisant le codage ANSI Latin.

Inconvénient :
Le résultat obtenu est affiché dans une « console ».

3.2 - Amélioration du progamme

On peut afficher également le résultat dans une fenêtre graphique. Pour cela, il faut utiliser le module graphique de Python.
Voici des captures d'écran obtenues ci-dessous avec le module graphique :

Sous Windows Sous Linux



3.3 – Compilation du programme

Pour compiler un programme Python en exécutable
Sous Windows : il est nécessaire d'installer le module py2exe pour créer un exécutable qui intégrera le programme et les dll correspondantes.
Sous Linux : il est nécessaire d'installer le module Freeze qui réalise des exécutables pour les systèmes Unix.

3 - RECUPERER DES DONNEES PROVENANT D'ORPHY GTS 2

On utilisera la même syntaxe pour les voies analogiques.

La différence est qu'Orphy GTS 2 fonctionne sur 16 bits donc à l'interrogation d'Orphy on reçoit alors 2 caractères et non plus un seul. Il faut convertir ces caractères sur 16 bits (nombre décimal compris entre 0 et 65535)

Le programme ci-dessous fonctionne pour les voies analogiques d'Orphy GTS2. Les entrées différentielles n'ont encore pas été testées.

```
# -*- coding:Latin-1 -*-
import serial                   #importe le module serial pour communiquer avec la voie série
se = serial.Serial()            #définition de la variable se communiquant avec le port série
se.baudrate = 9600              #sélection de la vitesse de transmission à 9600 bauds
se.bytesize=8                   #Transmission des informations sur 8 bits
se.parities=0                   #sans parité
se.stopbits=1                   #bit de stop 1
se.xonxoff=0                    #pas de contrôle logiciel
se.rtscts=0                     #pas de contrôle RTS/CTS
se.timeout=1                    #timeout (important)
se.port = 0                     #le port 0 est ici COM1, COM2 serait le port 1
print se.portstr                #affiche le nom du port ouvert
se.open()                       #ouvre la voie série sélectionnée
print se                        #Affiche l'état complet de la voie série
y=se.write("@")                 #requête sur la voie série @ pour EA0, A pour EA1, B pour EA2, ... G pour EA7
x=se.read()                     #lecture de la voie série octet de poids faible
y=se.read()                     #lecture de la voie série octet de poids fort
print x, y                      #affichage des caractères ASCII de lecture
t=1*ord(x)+256*ord(y)           #calcul du nombre codé sur 16 bits
if t > 32767 :
   U=((t)/65535.0)*20-20        #conversion en tension suivant le cas
else :
   U=((t)/65535.0)*20
print U
se.close()                      #fermeture de la voie série
```

Résultat obtenu :

```
>>>
COM1
Serial<id=0xb42630, open=True>(port='COM1', baudrate=9600, bytesize=8, parity='N'
, stopbits=1, timeout=1, xonxoff=0, rtscts=0, dsrdtr=0)
J ;
4.63202868696
>>>
```