Projet Robotique - Navigation
Formation ingénieur par apprentissage - 3ème Année
Travaux Dirigés (TD)

Introduction au langage Python

L'objectif de ce document est de fournir une base de référence au langage de programmation python. Il se focalise sur les éléments nécessaires à la mise en place de concepts mathématiques et la validation expérimental des méthodes. Ce document est certainement non exhaustif.

1 Hello World !

Pour commencer, il faut créer un dossier de travail et un fichier d'extension ".py" que nous éditerons.

Pour créer le fichier, utilisez un éditeur de texte de votre préférence.

Dirigez-vous vers le fichier à l'aide d'un explorateur de fichiers et éditez-le.

```
#premier script en python
text = "Hello World!"
print(text)
```

Fermez le fichier en sauvegardant les modifications et en vous assurant que le fichier porte une extension ".py".

Afin d'exécuter le script, faites appel à un terminal (CMD.EXE), et exécutez la commande suivante dans une fenêtre terminal :

```
python hello_world.py
```

En sortie de la console, vous devez observer le message "Hello World !".

2 Les variables et les opérateurs

L'exemple illustré ci-dessous implante différents types de variables numériques.

Quelques éléments à retenir concernant les variables et les opérateurs :

- La déclaration et l'initialisation de variables en python est obligatoire.
- La sélection du type associé à la variable est implicite (automatique).
- Une division entière ("floor-division") est faite si les opérants sont entiers. Si l'un des opérants est float la valeur réel est rendue.

```
#les variables:::::::::::::::::::::::::
a = 1 #une variable de type entier
b = 2 #une duxieme variable de type entier
c = 3.2 #une variable de type reel (float)
text = "Chaine de caracteres" #une variable de type chaine de caracteres
bool_var1 = None #une variable booleenne
bool_var2 = True
print("Variables :",a,b,c,text,"\n")
#operateurs:::::::::::::::::::::::::::::
addition = a+b
soustraction = b-a
produit = a*b
division =a/b
division2 = a/c
puissance = b**b
```

```
print("Operateurs :\n")
print("Addition :",addition,"\n")
print("Soustraction :",soustraction,"\n")
print("Produit :",produit,"\n")
print("Division (floor division) :",division,"\n")
print("Division2 :",division2,"\n")
print("puisssance X^Y:",puissance,"\n")
```

Des structures de données telles que les vecteurs et les matrices peuvent être manipulées au travers de la bibliothèque **Numpy**. Afin d'ajouter les fonctionnalités de cette bibliothèque il est nécessaire d'inclure en tête du script :

```
import numpy as np
```

Cette commande permettra d'instancier les fonctions de la bibliothèque au travers de la variable dénommé np.

La création d'un vecteur :

```
vecteur_int = np.array([1,2,3,4]) #vecteur d'entiers
vecteur_float = np.array([1.6,2.8,3.3,4.1]) #vecteur de reels
```

La création d'une matrice :

```
matrice = np.array([[1,2,],[3,4]]) #création d'une matrice
```

La création d'une matrice initialisée à zéro, à un et une matrice identité :

```
matrice_zeros = np.zeros((3,4)) #matrice nulle 3x4
matrice_ones = np.ones((2,5)) #matrice unitaire 2x5
identite = np.eye(3) #matrice identite
```

Les attributs concernant le nombre d'éléments et la forme d'un vecteur ou d'une matrice peut être obtenue de la manière suivante :

```
vecteur_float.size  #renvoir le nombre d'elements
matrice_zeros.shape #renvoie le nombre lignes,n,  et colonnes,m, (n,m)
```

L'indexation de vecteurs et de matrices commencent par l'indice 0. Les valeurs peuvent être accédées comme l'illustre l'exemple ci-dessous :

```
vecteur_int[0] #accede à la valuer du premier element
identite[1,2] #accede a la valeur de l'element (ligne,colonne)
```

Pour aller plus loin, les opérations vectorielles et matricielles peuvent être consultés sur le site `https://docs.scipy.org/doc/numpy-dev/user/quickstart.html`.

3 Structure de contrôle

Les conditions **IF**, **ELSE** et **ELSEIF** sont traités dans l'exemple ci-dessous. Attention, **l'indentation des lignes est un élément clé** qui décrit dans python la dépendance de la ligne à une structure de contrôle.

```
x = int(raw_input("Please enter an integer: "))
if x < 0:
        x = 0
        print 'Negative changed to zero'
elif x == 0:
        print('Zero')
elif x == 1:
        print('Single')
else:
        print('More')
```

4 Boucles

En python, la boucle **FOR**, peut fonctionner suivant une progression arithmétique ou elle peut itérer sur une séquence d'éléments. Ces modalités de fonctionnement sont illustrées dans l'exemple suivant :

```
# iteration d'elements:
words = ['cat', 'window', 'defenestrate']
for w in words:
        print(w,len(w))
# progression arithmetique
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
        print(i, a[i])
```

La boucle **WHILE** s'exécute tant que la condition d'itération soit validée. A savoir, toute variable comportant une valeur non nulle est considérée comme un état logique VRAI. Les opérateurs de comparaison sont : $<$, $>$, $==$,$<=$,$>=$ et $!=$ et ils correspond respectivement : inférieur, supérieur, égal, inférieur ou égal, supérieur ou égale et différent.

```
a, b = 0, 1
while b < 10:
        print(b)
        a, b = b, a+b
```

5 Traitement synchronisé

Pour le traitement synchronisé, il sera nécessaire faire appel à l'utilisation des fonctions de type "Timeout". Cela consiste en définir plusieurs fonctions (traitements) qui seront par la suite exécutées de manière régulière. L'intervalle de temps d'exécution de chaque fonction doit être définie par l'utilisateur en millisecondes.

Pour importer la bibliothèque **GObject** dans le script :

```
import gobject
```

Dans l'exemple suivant, l'intervalle représente le nombre de milliseconds entre deux appels de la fonction. La fonction constitue le nom de la fonction qui sera appelée de manière synchrone. Tout autre argument après le deuxième paramètre sera passé à la fonction comme argument.

```
source_id = gobject.timeout_add(intervalle, fonction, ...)
```

La valeur source_id peut être utilisée pour arrêter l'appel synchronisé de la fonction comme suit :

```
gobject.source_remove(source_id)
```

Autre alternative pour induire l'arrêt du timeout, il suffit que la fonction renvoie une valeur équivalente à FALSE.

Ci-dessous un exemple complet d'un appel synchronisé de deux fonctions :

```
import gobject
import time

def process_a():
        print("[process_a] :",time.time(),"\n")
        return True
def process_b():
        print("[process_b] :",time.time(),"\n")
        return True

if __name__ == '__main__':
        try:
```

```
        loop = gobject.MainLoop()                                          13
        handle = gobject.timeout_add(500, process_a) #every 500 ms         14
        handle2 = gobject.timeout_add (1000, process_b) #every 1000 ms     15
        loop.run() #                                                       16
    except KeyboardInterrupt:                                              17
        print("Fin du programme \n")                                       18
        pass                                                               19
```

6 Affichage d'une courbe

L'affichage de données pourra se faire à l'aide de la bibliothèque **Matplotlib**. Un exemple minimaliste permettant l'affichage de deux vecteurs de données est illustré ci-dessous. Pour commencer, il est nécessaire d'importer la bibliothèque et d'autres dépendances.

```
import matplotlib.pyplot as plot                                           1
import time                                                                2
import warnings                                                            3
import sys                                                                 4
sys.tracebacklimit=0                                                       5
warnings.filterwarnings("ignore",".*GUI is implemented.*")                 6
```

Ensuite, deux vecteurs de données sont déclarés en tant d'instances globales.

```
#plot data                                                                 7
x_data = []                                                                8
y_data = []                                                                9
```

Dans la procédure principale, la fenêtre d'affichage est initialisée, les vecteurs de données sont redimensionnés dynamiquement dans une boucle FOR et les données du plot sont mises à jour. A la finalisation du script, la fenêtre du plot est fermée.

```
if __name__ == '__main__':                                                 10
    plot.autoscale()                                                       11
    plot.show(block=False)                                                 12
    plot_handle, = plot.plot([-1,70],[-1,4000], marker='o', color='red')   13
    try:                                                                   14
        for i in range(60):                                                15
            x_data.append(i)                                               16
            y_data.append(i*i)                                             17
        plot_handle.set_xdata(x_data)                                      18
        plot_handle.set_ydata(y_data)                                      19
        plot.draw()                                                        20
        while True:                                                        21
            plot.pause(0.05)                                               22
    except KeyboardInterrupt:                                              23
        time.sleep(1)                                                      24
        plot.close()                                                       25
        print("Fin du programme \n")                                       26
        pass                                                               27
```