Chapitre 1.

Quelques rappels sur le langage Python

Plan du chapitre

I. L'environnement de développement Pyzo

I.1 Installation

On développe dans cet ouvrage l'utilisation du logiciel Pyzo. Celui-ci peut être installé en suivant les instructions détaillées sur le site :

https ://pyzo.org/start.html

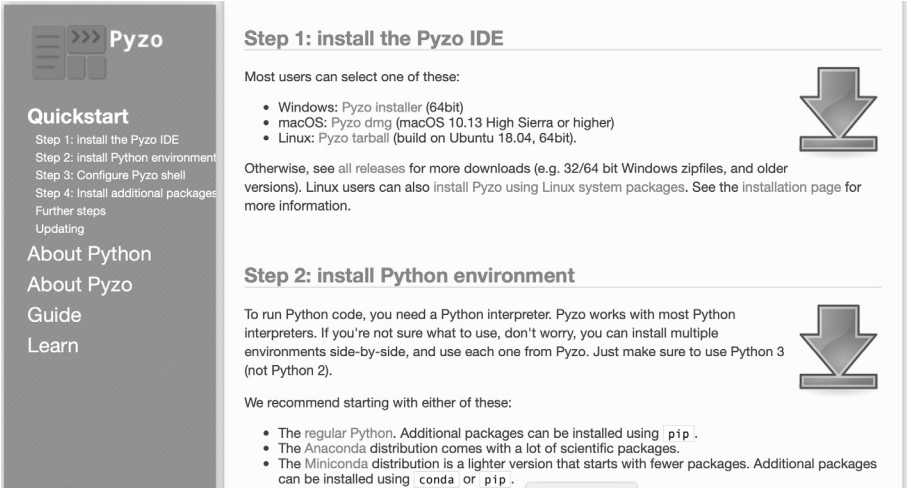

Fig. 01 : extrait du site *pyzo.org*

On recommande l'installation d'**Anaconda** pour l'utilisation des bibliothèques scientifiques mais surtout pas de **Miniconda** qui est trop limité.

I.2 Découverte de l'environnement de développement Pyzo

Pyzo va permettre de saisir des *programmes* (ou *scripts*) plus ou moins courts et d'en visualiser les résultats.

Tout d'abord, il faut lancer Pyzo. Les fenêtres les plus utiles pour la suite seront :

- le **shell** (pour la saisie de *scripts* très courts et la *visualisation* de résultats),
- l'**éditeur** (pour l'écriture de programmes plus ou moins longs que l'on peut sauvegarder)
- le **gestionnaire de fichiers** (pour visualiser les *fichiers* de travail).

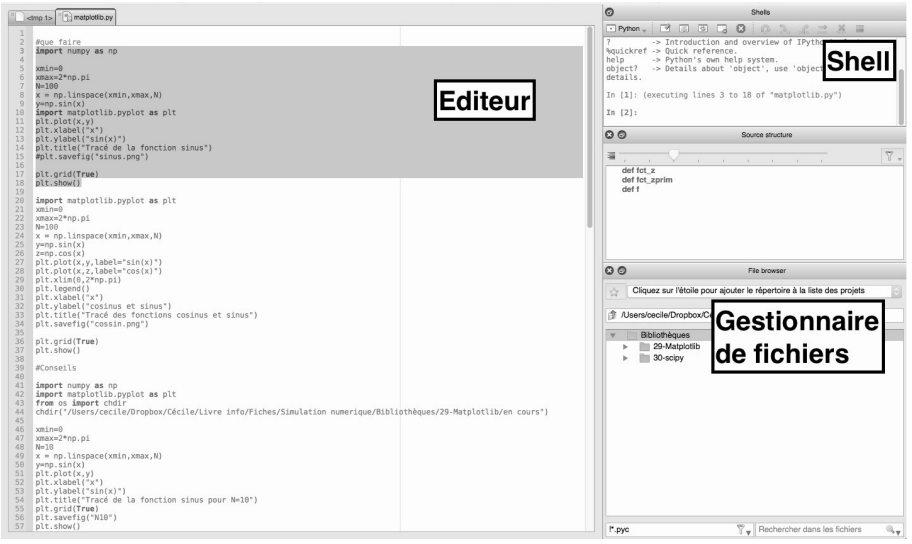

Fig. 02 : Les principales fenêtres de l'interface de Pyzo

II. Calculs simples

II.1 Calculs dans le shell

Voici quelques premiers calculs simples que l'on peut effectuer sur une interface Pyzo vierge. Le shell de *Pyzo 4* invite à saisir des instructions avec `>>>`. On peut alors taper :

```
1+1
```

Puis taper sur la touche du clavier Enter. Le shell renvoie le résultat (voir *Fig. 03*) :

```
2
```

Puis on souhaite calculer 3^2. Il reste à taper :

```
3**2
```

Après avoir enfoncé la touche Enter, ceci donne (voir *Fig. 03*) :

```
9
```

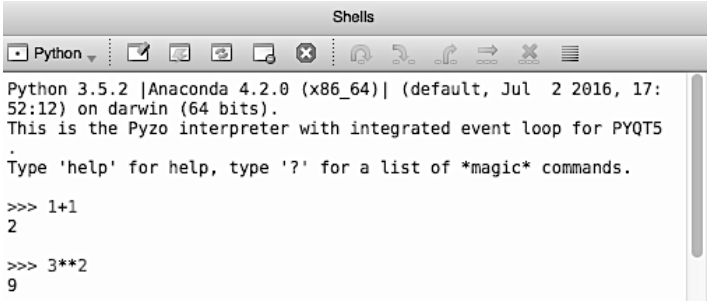

Fig. 03 : Le shell dans *Pyzo 4*

On peut également taper le calcul `23/3` :

```
1 | 23/3
```

Ceci renvoie un réel (appelé *flottant*) :

```
1 | 7.666666666666667
```

II.2 Priorité des opérations

Les parenthèses s'emploient comme lors d'un calcul mathématique, ou sur une calculatrice. La commande `2*3+2` donne `8`, là où la commande `2*(3+2)` donne `10`.

L'opération 2^2 s'écrit `2**2` en python, et le calcul de cette puissance est prioritaire sur les autres opérations.

```
>>> 2*3+2
8

>>> 2*(3+2)
10

>>> 1/2**2
0.25

>>> 1/(2**2)
0.25
```

Fig. 04 : Priorité des opérations

III. Manipuler des variables

III.1 Créer une variable et la modifier

Pour *déclarer* la variable `a`, dans l'éditeur de scripts ou dans le shell, il suffit de lui *affecter* une valeur :

```
a = 0.01
```

Ainsi la valeur 0.01 est *affectée* à la variable `a`. Pour *incrémenter* de 1 la variable `a`, on peut taper :

```
a = a + 1 #1 ajouté à a
#ou encore
a += 1 #1 ajouté à a
```

III.2 Type d'une variable

a) Entiers et flottants

Les nombres en python peuvent être de différents *types*. Pour tester le type d'un nombre, il suffit de taper la commande **type**`(nombre)`. Par exemple :

```
type(2)
```

Ceci renvoie **int** pour *integer* (*entier* en anglais) :

```
<class 'int'>
```

De même :

```
type(2.0)
```

Ceci renvoie **float** ou *flottant* en français (*nombre réel* en anglais) :

```
<class 'float'>
```

b) Conversion de type

Il est possible de forcer le type d'un nombre en tapant par exemple **int**`(nombre)` ou **float**`(nombre)`. Par exemple :

```
float(2)
```

Ceci renvoie un *flottant* :

```
2.0
```

De même :

```
int(2.0)
```

Ceci renvoie un entier (*integer*) :

```
2
```

c) Les booléens

Un booléen est une variable qui peut prendre deux états : soit l'état vrai (`True`), soit l'état faux (`False`). L'intérêt des booléens en informatique est qu'ils permettent de tester si une expression logique, est vraie ou fausse.

Dans un script Python, lorsque l'on essaie de déterminer si une expression logique est vraie ou fausse, le shell renvoie un booléen : le booléen `True` si l'expression est vraie et le booléen `False` si l'expression est fausse.

Si on affecte les valeurs suivantes : `a,b,c = 4,2,1` on peut effectuer les tests suivants :

Commandes	Signification	Résultat
`a==b`	`a` est égal à `b`	`False`
`a>b`	`a` est strictement supérieur à `b`	`True`
`a<b`	`a` est strictement inférieur à `b`	`False`
`a>=b`	`a` est supérieur ou égal à `b`	`True`
`a<=b`	`a` est inférieur ou égal à `b`	`False`
`a!=b`	`a` est différent de `b`	`True`
`a<b` **and** `b>c`	`a` est inférieur à `b` et `b` est supérieur à `c`	`False`
`a<b` **or** `b>c`	`a` est inférieur à `b` ou `b` est supérieur à `c`	`True`

- Donc assez naturellement, `a<b` **and** `b>c`, qui revient à `False` **and** `True`, renvoie `False`.
- De même, `a<b` **or** `b>c`, qui revient à `False` **or** `True`, renvoie `True`.

IV. Programmes simples dans l'éditeur

IV.1 Un premier programme

On peut commencer par taper dans l'*éditeur* (voir la *Fig. 05* pour le visuel) :

```
1 a = 1
2 b = 2
3 print("Le calcul de a+b donne", a+b)
```

Fig. 05 : Programme dans l'éditeur avant enregistrement

Puis on enregistre le programme créé sous le nom `Calcul.py` avant de l'exécuter :

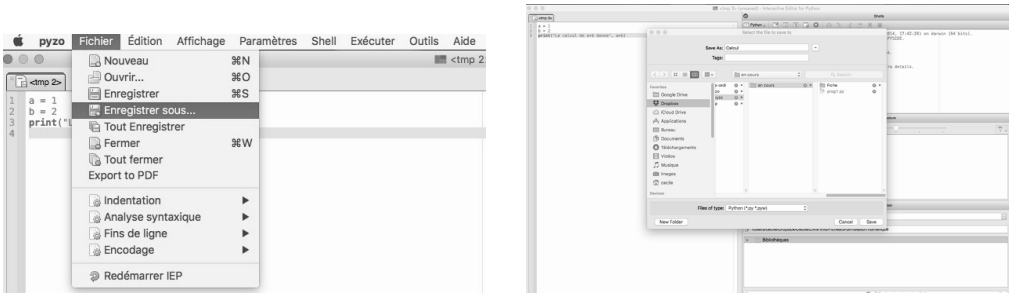

Fig. 06 : Les étapes d'enregistrement du fichier `Calcul.py`

Une fois le fichier enregistré, on va pouvoir exécuter le programme. On mentionne ici deux méthodes assez pratiques : *exécuter le fichier* (*Fig. 07a*) ou *exécuter la sélection* (*Fig. 07b*).

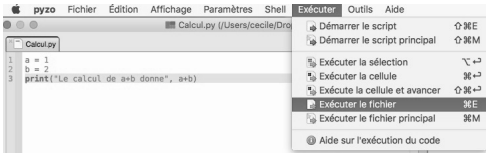

Fig. 07a : Exécuter le fichier

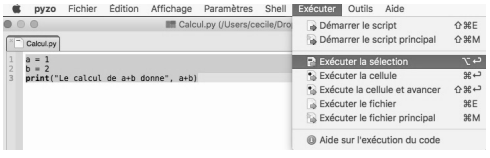

Fig. 07b : Exécuter la sélection

L'exécution du fichier `Calcul.py` donne le résultat suivant dans le *shell* :

```
>>> (executing file "Calcul.py")
Le calcul de a+b donne 3
```

Fig. 08 : Résultat de `Calcul.py`

Une fois le fichier `Calcul.py` enregistré, on peut vérifier sa présence dans le répertoire */Info* par exemple :

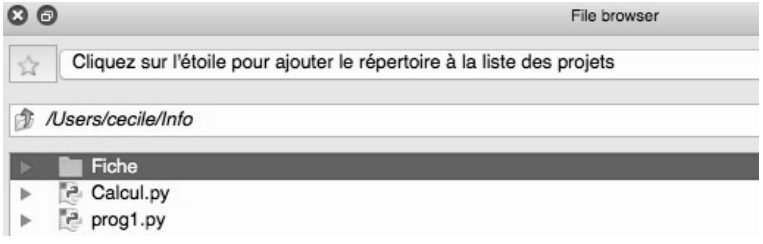

Fig. 09 : Visualisation de `Calcul.py`

IV.2 Affichage d'un résultat

Toutes les lignes tapées dans le *shell* peuvent être écrites dans l'*éditeur* et être *enregistrées*. Pour permettre l'affichage des résultats au fur et à mesure de leur calcul, **il faut utiliser** la commande **print( )**. Dans le cas contraire, seul le dernier résultat s'affiche dans le *shell*.

```
print("2*3 =",2*3)
```

Ceci donne dans le shell :

```
2*3 = 6
```

IV.3 Commentaires dans les programmes

On peut placer des commentaires grâce à l'utilisation des triples guillemets ou du #. Par exemple : `"""Commentaire"""` ou encore `#Commentaire`

Voici un exemple de programme tapé dans l'éditeur :

```
"""Exemples de calculs simples"""
print("2*3 =",2*3)
#Affichage du type
print("le type de 2 est :",type(2))
```

Puis on enregistre le programme créé sous le nom `Calcul2.py` avant de l'exécuter.

Calcul2.py

```
"""Exemples de calculs simples"""
print("2*3 =",2*3)
#Affichage du type
print("le type de 2 est :",type(2))
```

Fig. 010 : Programme `Calcul2` dans l'éditeur

Une fois le fichier enregistré, on peut exécuter le programme. L'exécution du fichier `Calcul2.py` donne le résultat suivant dans le *shell* :

```
2*3 = 6
le type de 2 est : <class 'int'>
```

On vérifie que 2 est de type entier (*integer*).

V. Fonctions mathématiques

V.1 La bibliothèque math

On peut utiliser les fonctions trigonométriques, logarithme, exponentielle etc... Cela peut se faire dans le *shell* directement mais le passage par l'*éditeur* est plus