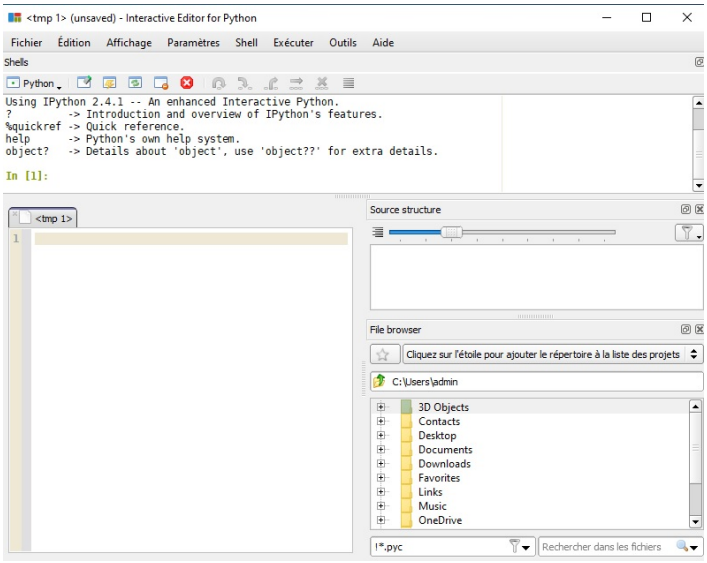
Fiche 2 - Environnement PYTHON

I. L'environnement de travail

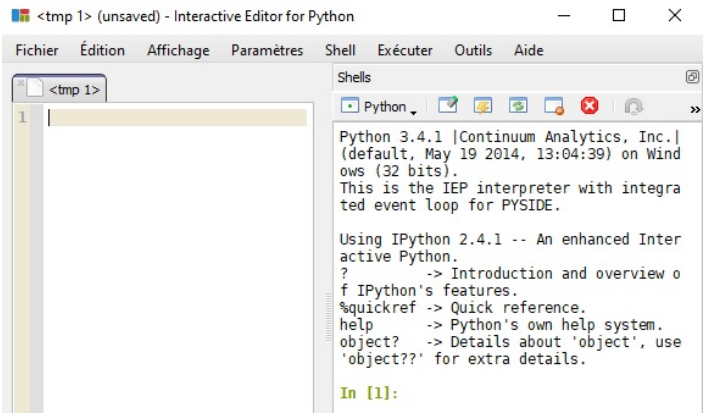
L'environnement se décompose en plusieurs fenêtres de position variable. Préalablement, il convient de choisir la langue et de fermer et redémarrer le logiciel.



Nous conserverons uniquement deux fenêtres :

⇒ à gauche, un **éditeur** pour écrire des fichiers de scripts.

⇒ à droite, le **Shells** où les scripts sont exécuté : ceux de l'éditeur et ceux que l'on peut taper directement.



II. Editer un fichier de scripts

Nous écrirons les scripts dans des fichiers, un par exercice. Cela permet de sauvegarder le travail, de le modifier à moindre coût.

Pour exécuter un script, le menu Exucuter propose plusieurs possibilité (Ctrl+E pour tout le fichier), le clic droit de la souris permet d'exécuter uniquement la partie du script pré-sélectionnée.

III. Commentaires dans un script

Afin de rendre lisible un programme, il est recommandé d'ajouter des commentaires.

Ainsi un lecteur extérieur ou lors d'une lecture ultérieure du programme, il est plus facile de le comprendre, de suivre son déroulement, de saisir sa finalité.

Il s'agit de :

- décrire l'objectif d'une feuille de calcul,
- donner une spécification dans la définition d'une fonction,
- préciser le rôle d'une variable.

➜ entre des triples guillemets ou apostrophes :

```
''' Cette methode permet
de faire des retours a la ligne.
C'est pratique !'''
```

➜ sur une ligne, après le symbole # :

```
p=1e-5  # p est la precision voulue
nb=200  # nb est le nombre d'iterations
n,m=4,5 # (n,m) est la taille de la matrice
```

Remarque : Dans une feuille de script, il est parfois pratique de n'exécuter qu'une partie du script. Deux solutions :

- sélectionner le reste, cliquer droit, et valider *Commenter / Décommenter*
- sélectionner la partie voulue, cliquer droit et valider *Exécuter la sélection*

IV. Les bibliothèques

Parfois appelée *modules*, *extensions* ou encore *packages*, les *bibliothèques* sont des banques de programmes/fonctions regroupées par thèmes.

IV.1. Quelques exemples

:

- numpy (pour "**num**eric **py**thon") : création et manipulation de tableaux (`array()`, `zeros()`, `shape()`, `dot`), algèbre linéaire (`matrix_rank()`, ...) et générateurs aléatoires (`rand()`, `randint()`, ...).

Quelques sous-modules du module numpy

Notation	Explication
`random`	simulations aléatoires
`linalg`	algèbre linéaire
`polynomial`	polynôme

- scipy (pour "**sci**entific **py**thon") : fonctions d'analyse (`odeint()` (équation différentielle), ...)

- `matplotlib` : outils graphiques (`plot()`, ...)

➜ Quelques autres bibliothèques :

Module	Description
`time`	Fonctions relatives au temps
`random`	Générations aléatoires de nombres
`Tkinter`	Création de fenêtre, gestion de la souris
`io`	Manipulation des fichiers
`math`	Fonctions mathématiques
`os`	Communication avec le système d'exploitation

Remarque : Les modules `numpy`, `scipy`, `matplotlib` sont ceux que nous utiliserons principalement.

- `sympy` (pour "**sym**bolic **py**thon") : calcul formel, calcul à précision arbitraire (`N()`)
- `pylab` : On y trouve tout ou presque de ce que vous pourriez avoir besoin ...

IV.2. Charger une bibliothèque

Il y a plusieurs façons de procéder, traitons le cas de la fonction `clock()` de la bibliothèque `time` qui donne en seconde la valeur de l'horloge-processeur sous windows :

➜ avec le module en préfixe : cette commande charge toute les fonctions de module `time` :

Charger un module

```
>>> import time
```

Pour appeler la fonction `clock` il convient de taper :

Appel d'une fonction

```
>>> time.clock()
6.775217977240195e-06
```

Attention ! Dans cette configuration, le nom de module doit être spécifié *`module.fonction()`*.

Il est possible de donner un alias au nom du module afin de raccourcir l'instruction :

```
>>> import time as tm
>>> tm.clock()
```

Remarque : Cette démarche est conseillée (par rapport à celle qui suit) car elle permet de bien identifier qu'une telle fonction provient de tel module. En effet, il existe des fonctions portant le même nom dans des modules différents.
Cette rédaction permet d'éviter les confusions.

➜ sans préfixe : la commande suivante charge uniquement la fonction voulue

Charger une fonction

```
>>> from time import clock
>>> clock()
```

La commande suivante charge toutes les fonctions du module

Charger un module

```
>>> from time import *
>>> clock()
```

Remarque : Dans certains modules, les fonctions sont organisées en sous-modules (ou classes).
Remarque : Il est possible de créer son propre module. Un module n'étant qu'une collection de fonctions.

V. Aide

Il y a plusieurs façons d'obtenir de l'aide sur une fonction ou un module :

➜ Avec la fonction `help` : taper dans la console

```
>>> help(clock)
Help on built-in function clock in module time:

clock(...)
    clock() -> floating point number

    Return the CPU time or real time since the
    start of the process or since the first call
    to clock(). This has as much precision as
    the system records.
```

➜ Sur internet, le site www.scipy.org donne des informations plus complètes sur les modules.

VI. Opérations d'entrée/sortie

En sortie : l'affichage des résultats est un attendu naturel du programme.

En entrées : il est intéressant de pouvoir demander à l'utilisateur la valeur des paramètres avant d'exécuter un programme.

Notation	Explication
`x=input('Message')`	Affiche le message spécifié, affecte à la variable x la chaîne de caractères entrée et validée par la touche return
`print(x)`	Affiche la valeur de x où x est une chaîne de caractère ou une variable
`print(x,y,z)`	Affiche x, y, z dans cet ordre.

Table 2.1: Entrées/sorties.

VI.1. La fonction input

La fonction `input` n'interprète pas ce qui est entré, elle le traite comme une chaîne de caractères. Il convient de préciser le type souhaité en utilisant : `int()`, `float()`, ...

```
>>> n=input('Donner n : ')
Donner n : 12
>>> n
'12'
>>> n=int(n)
>>> n
12
```

ou directement en composant les deux fonctions :

```
>>> p=float(input('Donner la precision p = '))
Donner la precision p = 1E-3
>>> p
0.001
```

Le script ci-dessous affiche l'exponentielle d'un nombre x donné par l'utilisateur :

Script exponentielle.py

```
a="L'exponentielle de ";
x=float(input('Calcul de l exponentielle de : '));
print(a+str(x)+' est '+str(exp(x)));
  # autre redaction pour l affichage !
print("L'exponentielle de",x,'est',exp(x));
```

Attention ! Si l'entrée est une expression qui n'est pas assimilable à un type simple (type `int`, `float` ou `complex`), elle doit être évaluée/interprétée.
Utiliser la fonction `eval()` qui prend en entrée une chaîne de caractères.

➜ Une fraction est interprétée comme un réel :

```
>>> a=input('Entrer une fraction : ')
Entrer une fraction : 1/2
>>> a;
'1/2'
>>> float(a)
ValueError: could not convert string to float: '1/2'
>>> eval(a)
0.5
>>> type(eval(a))
<class 'float'>
```

➜ les entier et les réels sont reconnus :

```
>>> b=eval(input('Cas d un entier : '))
Cas d un entier : 12
>>> b;type(b);
12
<class 'int'>
>>> c=eval(input('Cas d un reel : '));c;
Cas d un reel : 3.12
3.12
```

➜ Les fonctions de type `lambda` sont reconnues :

```
>>> f=eval(input('Donner une fonction "lamnbda" : '))
Donner une fonction "lamnbda" : lambda t:t**2-1
>>> type(f);f(2)
<class 'function'>
3
```

VI.2. La fonction print

La fonction `print` possède deux paramètres (voir l'aide de la fonction) :

➢ sep définit ce qu'il y a entre l'affichage de deux éléments. Par défaut, il y a un espace : `sep=' '`.

➢ end définit ce qu'il y a à la fin de l'affichage. Par défaut, il y a un retour à la ligne : `end='\n'`

Paramètres de print

```
>>> print(1,2,3,4);print('fin')
1 2 3 4
fin
>>> print(1,2,3,4,sep='',end='');print('fin')
1234fin
>>> print(1,2,3,4,sep='\n',end='\n\n_FIN_\n');
1
2
3
4

_FIN_
>>> print(1,2,3,4,sep='_',end='__');print('fin')
1_2_3_4__fin
```