TP 1 : Découverte de l'environnement Python et premières manipulations

PCSI, lycée Bertran de Born

1 Informatique, algorithmique et programmation

Dans le cadre du programme d'informatique, nous allons souvent parler d'algorithmique et de programmation. Commençons par définir les termes utilisés.

1. Un algorithme est une suite finie d'instructions permettant de résoudre un problème particulier. Généralement, un algorithme utilise et transforme des entrées et aboutit à des sorties.
2. L'informatique est la science de l'information et notamment son traitement automatisé. L'aspect qui nous intéresse est celui de la programmation c'est à dire de l'écriture des programmes. Un programme informatique peut être vu comme la traduction, dans un langage compréhensible par un ordinateur, d'un algorithme.

Illustrons la différence entre algorithme et programme : l'algorithme d'Euclide permet de calculer le PGCD de deux entiers a et b, en effectuant une succession finie de divisions euclidiennes. Dans ce cas, les entiers a et b sont les entrées et le PGCD est la sortie de l'algorithme. Si l'on implémente cet algorithme dans un langage informatique, cela devient un programme. On pourrait très bien coder cet algorithme sur calculatrice ou sur Python, ce qui nous donnerait deux programmes différents permettant d'exécuter le même algorithme.

2 Première utilisation de Python

2.1 En classe

Depuis votre session élève, ouvrez le dossier mathématiques, puis le dossier WinPython, puis ouvrez le logiciel Spyder, qui est un *environnement de développement* (en anglais *IDE - Integrated development environment*) pour Python.

Parallèlement, créez sur votre session un dossier `Info_rentrée_PCSI`,puis un sous-dossier TP1 où vous enregistrerez les fichiers relatifs à ce premier TP.

2.2 Chez vous

Télécharger Anaconda (gratuit pour les particuliers et étudiants). Une fois que le logiciel Anaconda est téléchargé et installé, lancez-le, puis ouvrez Spyder.

Parallèlement, créez à l'emplacement de votre choix un dossier `Info_rentrée_PCSI`,puis un sous-dossier TP1 où vous enregistrerez les fichiers relatifs à ce premier TP.

3 Manipulation des fenêtres

- En lançant le logiciel Spyder, plusieurs fenêtres sont apparues : la console pour exécuter directement des instructions, un fichier temp.py dans lequel on peut écrire un

programme entier à exécuter, ainsi qu'une troisième fenêtre avec plusieurs onglets (se mettre sur l'onglet *Variable Explorer* si ce n'est pas le cas). *Variable Explorer* est un navigateur de variables qui sert à visualiser toutes les variables que vous avez définies dans votre programme, avec leurs noms, leurs types respectifs, et leurs valeurs.

- Supprimer le navigateur de variables. Pour cela, allez dans *View* puis *Panes* puis décochez *Variable explorer*.

- Remettez le navigateur de variables. Pour cela, vous avez deux possibilités :
  1. allez dans *View* puis *Panes* puis décochez *Variable explorer*
  2. allez dans *View* puis *Window Layout* puis *Spyder Default Layout*

- Dans *View* puis *Window Layout*, testez à présent *Matlab Layout*. Vous obtenez un affichage avec davantage de fenêtres, avec en particulier un historique de commandes et une arborescence. Cet affichage est riche, mais avec beaucoup d'informations inutiles pour le moment.

- Dans *View* puis *Window Layout*, testez à présent *Horizontal Split*. Vous obtenez cette fois-ci un partage d'écran avec seulement deux fenêtres qui a l'avantage d'être particulièrement aéré, ce qui peut être pratique si vous avez un petit écran. En revanche, on ne visualise plus par défaut l'éditeur de variables, mais cela ne signifie pas qu'il n'est plus accessible. Vous pouvez l'afficher en le cherchant dans le menu *View*

Pour la suite, on utilisera *Spyder Default Layout*. Sélectionner l'onglet *Variable explorer*. A ce stade, la fenêtre est vide car aucune variable n'a été définie.

4 Python comme calculatrice

Exercice 1.

1. Ouvrir Spyder, puis réaliser les opérations suivantes dans l'interpréteur (la console).

   $50+3\times(12,5-31) =$ ..... $\quad \frac{3}{2} =$ ..... $\quad \frac{2}{3} =$ ..... $\quad 2^5 =$ .....

2. Évaluer les expressions suivantes : `23//3` et `23 % 3`, puis `24//3` et `24 % 3`.
3. Évaluer les expressions suivantes : `abs(1.3)` et `abs(-1.3)`.
4. Compléter le tableau ci-dessous :

Opération	**Symbole**	**Opération**	**Symbole/Commande**
Somme		Produit	
Différence		Division numérique	
	`//`		`%`
Puissance		Valeur absolue	

Exercice 2. Exécuter les commandes ci-dessous :

```
>>> sin(0)
>>> pi
```

Les fonctions et constantes mathématiques usuelles ne sont pas disponibles immédiatement à l'ouverture de Spyder ; il faut charger une librairie (ou module) complémentaire : le module `math`.

Exécuter la commande ci-dessous :

```
>>> from math import *
```

La commande `import` donne accès à une nouvelle librairie. Le symbole `*` signifie que toutes les commandes issues du module importé sont désormais accessibles.

Dans le navigateur de variables, combien de nouvelles variables sont désormais accessibles ? A quoi correspondent-elles ?

Exécuter les commandes ci-dessous.

```
>>> sin(0)
>>> pi
```

Remarque : On constate effectivement que la fonction sinus et la constante π sont maintenant accessibles.

Exercice 3. A votre avis, que va renvoyer Python lorsque vous allez exécuter les commandes suivantes ?

```
>>> cos(0)
>>> sin(pi)
>>> 3*0.33
>>> 3*0.333
```

Vérifier votre conjecture en exécutant effectivement ces commandes.

Remarque : Les écarts entre les résultats attendus et les résultats obtenus sont liés à des erreurs d'arrondis. Tout cela sera étudié en cours dans le chapitre sur la représentation des nombres. Il est cependant utile d'avoir tout de suite en tête que certaines erreurs d'arrondis peuvent avoir lieu dès que l'on utilise le type *float*.

Exercice 4. Déterminer (des valeurs approchées) des expressions suivantes.

$\ln(10) =$ ..... $\cos\left(\frac{\pi}{5}\right) =$ ..... $\frac{1+\sqrt{5}}{4} =$ .....

Que peut-on conjecturer concernant les deux dernières expressions ?

5 Variables et affectations

Lors de la rédaction de programmes, on sera amené à utiliser des *variables*. Ces variables permettent de manipuler les données du programme.

Les objets créés sous Python sont stockés dans la mémoire de l'ordinateur. Un *nom de variable* est une suite de caractères qui renvoie à une adresse mémoire où a été créé un objet.

5.1 Affectation d'une valeur à une variable

En pratique, pour affecter (ou assigner) une valeur à une variable on utilise le symbole `=`. Ce symbole est à distinguer de l'égalité mathématique pour les tests qui s'écrit `==` sous Python.

Exemple 1.

La commande suivante signifie que désormais le caractère `x` renvoie à la valeur `1`.

```
>>> x=1
>>> x
1
```

Exercice 5. Examiner la série de commandes ci-dessous. **Prédire** le résultat, puis le confirmer avec Python.

```
>>> x=1
>>> y=2
>>> x=x+y
>>> y=x**y
>>> y
?
```

Attention. Il existe des mots *réservés* ne pouvant faire office de nom de variable (`def`, `if`, `while`, `True`, `False`...).

5.2 Affectations multiples

Le langage Python offre la possibilité de réaliser en une seule instruction plusieurs affectations.

Affectations successives : La commande ci-dessous se comprend de la manière suivante : `x` représente la valeur `1` et `y` représente la valeur `x` (donc `1`).

```
>>> y=x=1
>>> y
1
>>> x
1
```

Affectations parallèles. La commande ci-dessous signifie que `a` représente la valeur `1` et que `b` représente la valeur `2`.

```
>>> a,b=1,2
>>> a
1
>>> b
2
```

La commande ci-dessous permet d'échanger les valeurs contenues dans deux variables.

```
>>> x,y=y,x
```

Exercice 6. Examiner la série de commandes ci-dessous. **Prédire** le résultat, puis le confirmer avec Python.

```
>>> var1,var2=1,2
>>> var1,var2=var2,var1+var2
>>> var3=var1**var2
>>> print(var1,var2,var3)
?
```

Remarque : La commande `print()` permet d'afficher la valeur d'une ou plusieurs variables séparées par des virgules.

6 Premier programme Python

Les programmes écrits dans l'interpréteur de Spyder ne peuvent pas être sauvegardés : ce n'est pas très pratique si vous souhaitez relancer une même série de calculs avec des valeurs différentes pour les variables.

Pour écrire et sauvegarder des programmes, la procédure générale est la suivante :

1. Ouvrir un nouveau fichier dans l'éditeur.
2. Écrire et enregistrer le script. Le nom du fichier doit se terminer par l'extension **.py**.
3. Exécuter le script : Exécution → Exécution ou F5 ou cliquer sur l'icône représentant une flèche verte. **Les lignes du script sont interprétées les unes à la suite des autres.**
4. Les résultats sont donnés dans l'interpréteur.

Exercice 7. On souhaite rédiger un programme élémentaire qui calcule le prix d'une commande de livres. Les trois valeurs suivantes sont représentées par les variables :

`nbr` = entier désignant le nombre de livres commandés
`prix` = prix unitaire d'un livre
`reduc` = pourcentage de réduction (compris entre 0 et 100) dont bénéficie le client

Le programme devra afficher le montant de la facture.

1. Trouver une expression algébrique représentant le montant de la facture en fonction des variables `nbr`, `prix` et `reduc`.
2. Ouvrir un nouveau fichier que vous sauvegardez sous le nom `FactureLivres.py`.
3. Affecter les variables comme indiqué ci-dessus dans le cas d'une commande de 27 livres dont le prix unitaire est de 30,50 euros pour un client bénéficiant de 5 % de réduction.
4. Afficher le résultat dans l'interpréteur. Pour afficher un résultat, on peut faire appel à la fonction `print`. Pour afficher une phrase, vous pouvez, par exemple, utiliser la syntaxe : `print('Le montant de la facture est de',m,'euros')`. Vous afficherez la valeur tronquée de `m` à deux chiffres après la virgule (utilisez `round`). Pour en savoir plus sur la commande `round`, tapez `help(round)`.

Remarque : Pour faire figurer des commentaires dans un script, on les rédige précédés d'un symbole `#`. Ils apparaissent ainsi dans le script sans être interprétés.

7 Types

7.1 Les différents types

Sous Python, toutes les variables ont un **type**. Voici les exemples de types les plus fréquemment utilisés :

1. `int` : les entiers relatifs. *Cf chapitre "Représentation des nombres en machine".*
2. `float` : les nombres à virgule flottante. *Cf chapitre "Représentation des nombres en machine".*
3. `str` : les chaines de caractères.
4. `bool` : les booléens. Il s'agit d'expressions dont la valeur est `True` ou `False`.

7.2 Manipulation des chaînes de caractères

Pour les définir, il faut les écrire entre *quotes* (simples ou doubles). Les éléments d'une chaine sont numérotés à partir de 0. Pour extraire un caractère, il suffit d'accoler à la chaine (ou la variable qui la représente) son indice entre crochet.

```
>>> message='bonjour'
>>> print(message)
bonjour
>>> message[2]
'n'
```

Si cela a un sens, les commandes `int` et `float` convertissent une chaine de caractères respectivement en un entier ou un flottant. Et inversement, la commande `str` convertit un nombre en une chaîne de caractères.

Voici une liste non exhaustive d'opérations autorisées sur les chaines de caractères.

Opération	**Symbole/Commande**	**Opération**	**Commande**
Concaténation	+	Extraction de caractères	*Chaine*`[n:m]`
Extraction d'un caractère	*Chaine*`[n]`	Longueur	`len`(*Chaine*)

Exercice 8. On suppose que la variable `s` est affectée de la chaîne de caractères `'salut'`. Que produisent les instructions suivantes ? Vous commencerez par les deviner avant de vérifier sur l'interpréteur.

```
>>> l=len(s) ; s
>>> s[0] ; s[l-1]
>>> s[l]
>>> s[0 : 3]
>>> s[2 : 3]
>>> s[2 : 2]
>>> t = s+' '+s ; len(t)
```

```
>>> 1>0
True
>>> 1==0
False
```

7.3 Manipulation des booléens

Opération	Opérateur	Opération	Opérateur
Égal	==	Différent	!=
Strictement supérieur	>	Strictement inférieur	<
Supérieur ou égal	>=	Inférieur ou égal	<=
ou	`or`	et	`and`
non	`not`	Test d'appartenance	`in`

Exercice 9. Prédire la valeur des booléens suivants et confirmer votre résultat avec l'interpréteur Python.

`not(1>0 and 0>=1)` : ..... `1!=0 or 1==1` : ..... `'a' in 'mathématiques'` : .....

Remarque : La commande `type(expression)` permet d'obtenir le type d'une expression.

8 La commande `input`

La commande `input` *interrompt* l'exécution des instructions et *attend* que l'utilisateur réalise une entrée. Elle récupère les entrées sous forme d'une **chaîne de caractères**.

Exemple 2.

Ci-dessous, la commande `input` récupère l'age de l'utilisateur sous la forme d'une *chaîne de caractères*.

```
>>> X=input('Quel âge avez-vous ?')
Quel âge avez-vous ? 17
>>> X,type(X)
('17', <class 'str'>)
```

Exercice 10. Reprendre l'**Exercice 7** en utilisant la fonction `input`. Votre programme interroge l'utilisateur au sujet du nombre de livres, puis lui affiche le montant de sa facture.

9 Exercices de synthèse

Exercice 11. Écrire un programme Python qui demande à l'utilisateur une valeur numérique et affiche une valeur tronquée à deux décimales de la racine carrée de cette valeur.

Exercice 12. Écrire un programme, sauvegardé dans un fichier `horaire.py`, qui demande à l'utilisateur : l'horaire de départ de son train, l'horaire d'arrivée de son train et qui affiche la durée du trajet en minutes, puis en heures et minutes.

On suppose que l'utilisateur rentre les horaires sous le format : `XXhYY` (exemples : `19h51`, `08h01`).

Tester le programme pour les entrées : départ 17h51 ; arrivée 19h38. Dans l'interpréteur, le résultat doit ressembler à ça :

```
>>>
Horaire de départ ? 17h51
Horaire d'arrivée ? 19h38
Votre trajet dure 107 minute(s), soit 1 heure(s) et 47 minute(s).
```