Pré-requis Python et environnement de travail IDLE

Ce T.P. a pour but de vous familiariser avec l'environnement de travail au lycée :

- dossier classe, avec ses deux répertoires **_travail** et **pnom** ;
- l'interface de codage IDLE,

et de reprendre des connaissances acquises au lycée et d'usage constant :

- types de base les plus courants ;
- opérations sur ces types de base, évaluation d'une expression ;
- déclaration, nommage et affectation de valeur à une variable, mécanisme de l'affectation ;
- écriture d'une fonction, principe de l'appel par valeur ;
- utilisation de fonctions prédéfinies ou issues d'un module.

1.1 Accès à la console d'IDLE

Le raccourci présent sur le bureau et dans le menu "Démarrer>programmes" pointe vers l'emplacement du fichier " **IDLE (Python GUI).exe**" qui lance le programme **IDLE** de la distribution **WinPython**.
Si vous avez installé la distribution **WinPython** sous le répertoire **C:** de votre ordinateur, le chemin vers le fichier de lancement d'IDLE est de la forme :

C:\WPy-3710\IDLE (Python GUI).exe

Remarque : c'est un raccourci qu'il vous est conseillé de créer vous-même sur vos machines.

On lance ce raccourci par un double-clic, et la fenêtre qui s'ouvre à un intitulé du type « Python 3.x.x Shell » et s'y affichent des lignes dont le contenu à l'allure suivante :

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

On y trouve une information sur la version de Python ainsi que sur le processeur de la machine utilisés.

Dans cette fenêtre, on peut déjà s'apercevoir que la **console interactive** (ou **terminal**) peut être utilisée comme une calculatrice, en saisissant par exemple une simple addition suivie de la touche « **Enter** ».

```
>>> 2+3
5
>>>
```

Remarque : Pour gagner du temps dans ce qui suit, on pourra utiliser les raccourcis clavier **Alt+p** et **Alt+n** – ou **Ctrl+p** et **Ctrl+n** sur Mac – qui permettent de faire défiler les instructions déjà validées à partir de la dernière ou de la première, respectivement.

1.2 La notion de « type »

Lorsque l'on manipule des objets en informatique, ces objets ont usuellement un type, qui conditionne la façon dont ils sont encodés en machine, les opérations que l'on peut leur appliquer et la façon dont elles sont effectuées. C'est pourquoi **il est capital de connaître le type des objets manipulés**.
La commande Python `type(x)` renvoie le type de l'objet `x`.
Les types de base du langage Python sont les entiers (`int`), les flottants (`float`), les chaînes de caractères (`str`) et les booléens (`bool`). Il existe d'autres types d'objets Python, comme, par exemple, les *n*-uplets (`tuple`), les listes (`list`), les fonctions (`function`), le type à valeur unique, `None`, (`NoneType`), les quatre premiers sont abordés ici, les quatre derniers seront bientôt plus précisément décrits.

Étant conçu sur le principe du **typage dynamique**, c'est Python lui-même qui assigne un type à une valeur, une expression ou un objet, à la suite de l'analyse qu'il (l'interpréteur Python) en fait.

Exercice 1. Deviner le type que Python attribue - ou non (!) - aux valeurs suivantes :

`33` ; `3.3` ; `"3.3"` ; `3,3` ; `33.` ; `3j` ; `"Python"` ; `True` ; `033` ; `3e-1` ; `'True'` ; `true` .

Vérifier ensuite en saisissant ces instructions dans la fenêtre interactive, en utilisant éventuellement la fonction `type()`.

1.3 Un peu de mathématiques

1.3.1 Addition, soustraction, multiplication, exponentiation

On dispose en Python des opérations d'addition, de soustraction et de multiplication et d'exponentiation (**attention** : notée `**` et non pas « ^ » en Python) sur les objets dont le type est numérique (entiers, flottants). Les règles usuelles de priorité entre les opérations - et celles relatives au parenthésage - sont respectées.

Remarque : On pourra tester ces mêmes opérations sur des flottants et aussi constater par exemple que la présence d'un flottant dans une expression induit une conversion de type « à la volée » en flottant de tous les entiers de l'expression. Ainsi, les expressions `2.+3`, ou `2**3.` sont évaluées à une valeur de type `float`.

Ce comportement préprogrammé de Python est à connaître, mais cela fait partie des « **bonnes pratiques** » **de n'effectuer des opérations que sur des objets de même type,** en particulier pour les **comparaisons de valeurs**.

1.3.2 Les divisions

1.3.2.a La division entière et le « modulo »

La *division entière* d'un entier a par un entier non nul b a pour résultat le quotient (usuellement noté q en mathématique) dans la division euclidienne de a par b. La division entière se note `//`.

L'obtention du reste dans la division euclidienne d'un entier *a* par un entier *b* est d'une grande utilité en informatique (par exemple pour mettre en place des tests de parité ou de divisibilité). En Python, ce reste s'obtient par la commande `%` que l'on lit « modulo ».

Exercice 2. Calculer à l'aide des opérateurs « division entière » et « modulo », le quotient, q, et le reste, r, dans les divisions euclidiennes de 23 par 7, de 2 par 3, de 15 par 3.

Vérifier dans chaque cas l'égalité $a = bq + r$ et la contrainte $0 \leq r < b$.

1.3.2.b La division décimale

La division décimale est utilisable avec des entiers ou des flottants, mais le résultat est toujours un flottant.

Hormis dans ce paragraphe, dans un souci de **bonnes pratiques, on effectuera une division décimale seulement avec des opérandes qui sont de type flottant.**

Exercice 3. Effectuer la division de 3,5 par 3 ; de 3 par 0,5 ; de 2 par 11 ; de 15 par 5.

Exercice 4. Tester la division par 0.

Exercice 5. Calculer la somme algébrique

```
1/10+1/10+1/10+1/10+1/10+1/10+1/10+1/10+1/10+1/10
```

La valeur affichée correspond-elle à la valeur attendue ? Comment expliquer ce résultat ?

1.3.3 Le module `math`

Il est possible d'utiliser les fonctions et constantes mathématiques usuelles (sin, cos, exp, ln - notée `log`, π, *e*, *etc.*) en Python. Cependant ces fonctions et constantes ne sont pas directement accessibles au démarrage de l'interface : elles sont définies dans un « module » dédié, nommé `math`, qui n'est pas chargé au démarrage, et il faut les importer dans la session en cours, via l'une des syntaxes suivantes :

Importation des seuls objets utiles

```
>>> from math import cos, pi
>>> cos(pi)
-1.0
```

Importation de tout le package

```
>>> from math import *
>>> log(e)
1.0
```

Il existe encore d'autres façons d'accéder (« importer ») aux fonctions d'un module, nous y reviendrons.

Exercice 6. **Q6.1.** Importer les fonctions `sin`, `cos`, `exp` et `log` du package math.

Q6.2. Calculer $\cos^2 1 + \sin^2 1$, $\cos^2\left(\frac{\pi}{4}\right) + \sin^2\left(\frac{\pi}{4}\right)$, $\ln(\exp(2))$, $e^{\ln 2}$. Commenter les valeurs obtenues.

1.4 Un peu de logique : le type booléen

On rappelle qu'en Python, on dispose des opérations logiques `not`, `and` et `or` sur les booléens `True` et `False`.

Exercice 7. **De tête :** après avoir chargé la constante `pi` du package math, quel serait la valeur des expressions `(1<=0 or 3.14<pi)`, `(1>=0 or 3.14<pi)` et `(1<=0 or 3.14>pi)` ? Vérifier.

Exercice 8. Évaluer dans la console interactive les expressions `not True or True` et `not (True or True)`. Qu'en déduisez-vous sur la priorité entre les opérations `not` et `or` ?

Exercice 9. **[Bonus]** Reconstituer les tables de vérité des opérateurs logiques NOT, AND et OR, à l'aide d'expressions utilisant ces opérateurs et les valeurs booléennes `False` and `True`.

Exercice 10. **[Évaluation paresseuse des expressions booléennes]** Est-il possible de prédire rapidement la valeur de l'expression `1 < 2 or 10e3 < 2e10 or 1/3 < 1.33` ?

Pourquoi l'évaluation de l'expression `2 == 3 and 1 / 0 < 7` ne provoque-t-elle pas d'exception ?

Remarque : Pour éviter les erreurs d'interprétation, il est expressément demandé de toujours utiliser des expressions parenthésées dans les tests comportant des plusieurs opérateurs booléens différents.

1.5 Affectations de variables

Exercice 11. Dans chacun des exemples suivants, on suppose qu'initialement, aucune variable n'est définie (on notera « X » dans le tableau à compléter pour indiquer qu'une variable n'est pas définie). On exécute ensuite une série d'instructions d'affectation.
Quelles sont les valeurs stockées dans les variables `x` et `y` après avoir validé les suites d'instructions suivantes ? Deviner et vérifier ensuite.

exemple 1

```
>>> x = 12
>>> y = x
>>> x = 13
```

exemple 2

```
>>> x, y =12, 7
>>> x = y
>>> y = x
```

exemple 3

```
>>> x = 12
>>> y = 7
>>> x, y = y,
x
```

exemple 4

```
>>> x,y=12,7
>>> z=x
>>> x=y
>>> y=z
```

exemple 5

```
>>> x,y=12,7
>>> x+=y
>>> y=x-y
>>> x=x-y
```

Compléter les tableaux suivants donnant l'état des variables après chaque validation d'instruction :

	exemple 1			exemple 2			exemple 3			exemple 4			exemple 5		
Etat des variables	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
Initialement															
Après exécution															
de la 1ère instruction															
de la 2ème instruction															
de la 3ème instruction															
…															

Exercice 12. **Q12.1** Vérifier que la commande « Restart shell » du menu Shell de la fenêtre IDLE (raccourci : Ctrl+F6) a bien pour effet de réinitialiser l'état des variables dans la console interactive.

Q12.2. **[Bonus]** Affecter la valeur 2 à une variable `a`. Tester la commande `id()`, cette commande renvoie un entier correspondant à l'adresse mémoire de la **valeur** affectée à `a` (la "vraie" adresse de la valeur

dans la mémoire vive est une valeur hexadécimale que l'on obtient en saisissant `hex(id(a))`). Si l'on affecte une nouvelle valeur à `a`, cette adresse change-t-elle ? Qu'en est-il si l'on effectue entre temps un Restart ?

2 Écriture de scripts à l'aide de l'éditeur de texte d'IDLE

L'éditeur de texte s'ouvre via le menu « File > New File » (raccourci : Ctrl+N). Après avoir écrit les lignes d'instructions constituant le programme, il faut enregistrer le fichier avec l'extension .py. L'exécution de la séquence des instructions se fait alors via le menu « Run > Run Module » (raccourci : F5 ou Ctrl+F5 sur Mac).

2.1 La fonction `print()`

Un appel à la fonction `print` permet de provoquer un affichage dans la console interactive.

Exercice 13. Écrire un script **Hello.py** qui affiche la phrase "Hello World !" dans le terminal. Exécuter le script afin de vérifier son bon fonctionnement.

Lors de l'exécution d'un script (fichier de texte contenant une suite d'instructions, enregistré avec l'extension **.py**), les instructions sont traitées par la machine mais aucun affichage n'est produit, sauf en cas d'erreur au moment de l'interprétation du programme.
La commande `print()` permet de forcer l'affichage dans le terminal d'une chaîne de caractères, du résultat de l'évaluation d'une expression ou du contenu d'une variable.

- Exemple d'utilisation de la fonction `print` :

Test de l'algorithme d'échange de la valeur de deux variables dans la console interactive

```
>>> a = 1
>>> b = 2
>>> temp = a
>>> a = b
>>> b = temp
>>> a
2
>>> b
1
```

Test de l'algorithme d'échange de la valeur de deux variables dans un script

```
a = 1
b = 2
print("état initial des variables")
print("a =", a, "b =", b)
temp = a
print("état des variables")
print("a =", a, "b =", b, "temp =", temp)
a = b
print("état des variables")
print("a =", a, "b =", b, "temp =", temp)
b = temp
print("état final des variables")
print("a =", a, "b =", b, "temp =", temp)
```

Exercice 14. Écrire un script **TestDivisionEuclidienne.py** dans lequel on affecte des valeurs entières positives, choisies de façon arbitraire, à deux variables `a` et `b`, et dans lequel on affecte ensuite à deux variables `quotient` et `reste`, respectivement, le quotient et le reste dans la division euclidienne de a par b.

Ajouter à ce script des appels à la fonction `print` permettant de vérifier, par lecture explicite à l'écran, que l'égalité $a = bq + r$ et la condition $0 \leq r < b$ sont bien vérifiées.

- Affichage à obtenir à l'exécution si l'on a affecté la valeur 25 à a et la valeur 7 à b :

```
Dans la division euclidienne de a = 25 par b = 7,
le quotient vaut 3 et le reste vaut 4, et on vérifie que :
7 * 3 + 4 vaut 25, et que la condition 0 <= r and r < b est évaluée à True.
```

On a mis en évidence ci-dessus en gras les valeurs qui s'affichent suite à l'évaluation d'une expression. Tester votre script avec une valeur positive pour a et une valeur strictement négative pour b (il doit s'afficher que la condition est évaluée à `False`).

2.2 Définition d'une fonction

Exercice 15. Écrire dans un fichier **fonctions.py** une fonction, `somme`, de paramètres a et b (a et b désignent des entiers) et qui renvoie la somme des entiers a et b.

Définir une fonction `somme_prod`, de paramètres a et b (a et b désignent des entiers) et qui renvoie la somme et le produit des entiers a et b.

Exercice 16. Écrire dans un fichier **ex14.py** une fonction `div_eucl`, de paramètres entiers a et b, et qui renvoie le quotient et le reste dans la division euclidienne de a par b.

Exercice 17. Définir une fonction `est_rectangle`, de paramètres entiers a, b et c et qui renvoie un booléen (`True` ou `False`) indiquant si le triangle de côtés a, b et c est un triangle rectangle ou non.

Exercice 18. Définir une fonction affichant 10 fois « Bonjour ».

Exercice 19. Définir une fonction qui affiche la table de multiplication de 7, sous la forme :

« 1 x 7 = 7 »

« 2 x 7 = 14 »

…

« 10 x 7 = 70 »