

EA4 – Éléments d’algorithmique


TP n° 1 : premiers pas avec Python

Avant toute chose : vérifiez que vous êtes bien connecté(e) à Moodle par votre compte « Université de Paris » (avec lequel vous avez été inscrit(e) automatiquement à tous vos cours) et non par votre ancien compte « Université Paris Diderot », voué à disparaître.

Ensuite, et ensuite seulement, rattachez-vous à votre groupe via le module d’inscription <https://moodle.u-paris.fr/mod/choicegroup/view.php?id=245175>.

Avant de commencer le TP, une version 3.x.x de Python doit être installée sur votre ordinateur. Vous trouverez des explications pour le faire sur Moodle (<https://moodle.u-paris.fr/mod/page/view.php?id=243402>).

Version de Python : attention à ne pas utiliser une version 2.x.x de Python mais exclusivement une version 3.x.x, elles ne sont pas compatibles entre elles. L’interpréteur correspondant est en général lancé par la commande `python3`, et non `python`, et nous utiliserons systématiquement le nom `python3` dans les énoncés. Pour connaître la version exacte d’un interpréteur Python, lancez-le depuis votre shell avec l’option `--version`.

Modalités de rendu : À chaque TP, vous devrez déposer sur Moodle les exercices marqués par un symbole . Le rendu de l’exercice *K* du TP *N* doit être inclus dans le fichier `tpN_exK.py`, à télécharger depuis la section « Énoncés de TP ». Vous devez remplir les zones marquées par le commentaire **A REMPLIR** dans ce fichier. Ne modifiez pas les autres fonctions du fichier, sauf demande explicite de l’énoncé. Chaque fichier contient une fonction `main` qui teste les fonctions que vous devez programmer et qui vous affiche un score donné par le nombre de tests passés avec succès. Pour passer ces tests, vous devez exécuter le programme écrit.

Le rendu du TP est à déposer sur Moodle le jour même du TP, c’est à dire le jeudi.

Dans ce premier TP, les exercices 1 à 3 ont pour seul but de vous donner des éléments pour programmer en Python. Vous pouvez normalement les faire sans aide, donc n’hésitez pas à les faire avant jeudi, ou au contraire plus tard si le temps vous manque. L’exercice 4, quant à lui, est une application du cours et doit donc être fait pendant les heures de TP afin que vous puissiez poser des questions aux enseignants.

Exercice 1 : interactif ou non ?

Ce premier exercice vous permet de découvrir les différents moyens d’utiliser l’interpréteur de Python.

1. Depuis un terminal, lancez l’interpréteur de Python (version 3) grâce à la commande `python3`. Dans l’interpréteur, écrivez et exécutez une commande qui affiche la ligne de texte `Hello World!`. Quittez l’interpréteur en tapant `^D` (touches `Control` et `D` simultanément) ou en appelant la fonction `exit()`.

2. Créez un fichier `hello.py` contenant comme unique ligne de texte la commande précédente. Sauvegardez ce fichier et tapez dans un terminal la commande `python3 hello.py`. Observez le résultat. Quelle est la différence avec `python3 -i hello.py` ?
3. Lancez à nouveau l'interpréteur Python et tapez la commande `import hello`. Que se passe-t-il ? Grâce à la commande `dir()`, affichez la liste des identificateurs connus. Que constatez-vous ?
4. Rendez le fichier `hello.py` exécutable et tentez de l'exécuter. Que se passe-t-il ? Rajoutez la ligne `#!/usr/bin/env python3` au début du fichier et réessayez.
5. Faites les calculs $13/4$ puis $4/2$. Que constatez-vous ? Essayez encore $13//4$ puis $4//2$. Quelle est la différence ?
6. Faites le calcul $\sqrt{3} + 56/9.0 \times |-1/4| + 63^2$ soit en une seule fois, soit en plusieurs fois en utilisant une variable. Pour trouver la syntaxe des opérations, consultez l'aide sur les nombres entiers en tapant `help(int)` et sur la bibliothèque de fonctions mathématiques en tapant `help("math")`.
7. Dans le fichier `hello.py`, créez une fonction `affiche` qui affiche `Hello World!`. Puis testez les commandes suivantes :

```
import hello
affiche()
hello.affiche()
```

Sans quitter l'interpréteur, modifiez votre fonction `affiche` afin qu'elle affiche également `Bonjour le monde!`. Que se passe-t-il lorsque vous appelez `affiche` dans l'interpréteur ? Et si vous importez de nouveau `hello` au préalable ?

Testez maintenant les commandes suivantes :

```
import imp
imp.reload(hello)
hello.affiche()
```

Finalement, testez les commandes suivantes dans un nouvel interpréteur :

```
from hello import affiche
affiche()
hello.affiche()
```

8. Ajouter au fichier `hello.py`, une fonction `dev` qui affiche `developpeur` suivi de votre nom.

Testez ensuite les commandes suivantes dans un nouvel interpréteur :

```
from hello import *
affiche()
hello.affiche()
dev()
hello.dev()
```



Exercice 2 : variables, expressions conditionnelles

Vous utiliserez encore l'interpréteur Python. Vous pouvez utiliser `del Z` pour supprimer la variable `Z` lors de vos tests.

1. Expliquez la différence entre une variable non-définie et une variable dont la valeur est `None`. Pour cela, exécutez l'instruction `print(Z)` quand `Z` est non-définie, vaut `None`, ou vaut la chaîne vide.
2. Écrire une expression qui s'évalue à `True` lorsque `Z` vaut `None` et à `False` lorsque `Z` est définie et différente de `None`.
3. Écrire une expression qui s'évalue à la chaîne `"sans valeur"` si `Z` vaut `None`, `"chaîne vide"` si `Z` vaut la chaîne vide et `"autre"` dans les autres cas où `Z` est définie. (Attention, il s'agit d'une expression et pas d'un segment de code qui affiche la valeur.)
4. À quelle valeur l'expression `True if x > 0 else False` s'évalue-t-elle pour `x` ayant les valeurs 3, -5, `None` ou non-défini ?
Même question pour l'expression `None if x==None else True if x > 0 else False`.
5. Dans le fichier `tp1_ex2.py`, compléter la fonction `expression_5` qui retourne une expression valant `True` si $x > 0$ et `False` si $x \leq 0$ ou si `x` vaut `None`.




Exercice 3 : listes

1. À l'aide des fonctions `range` et `list`, créez les listes suivantes :
 - la liste des entiers consécutifs de 0 à 9,
 - la liste des entiers consécutifs de 2 à 10,
 - la liste des entiers pairs de 2 à 10,
 - la liste décroissante des entiers pairs de 10 à 2.
2. En utilisant la fonction `range`, affichez les entiers de 0 à 9 (un par ligne).
3. On peut également créer une liste en la définissant *en compréhension*, c'est-à-dire en décrivant comment construire ses éléments, de la façon suivante :
`[<expression> for <element> in <iterable>]`
 Par exemple, `[i+1 for i in range(3)]` crée la liste `[1, 2, 3]` et `[i for i in 'abc']` crée la liste `['a', 'b', 'c']`.
 - a. En utilisant ce mécanisme, créez les listes `L1 = [0, 2, 4, 6, 8, 10, 12]` et `L2 = ['a', 'b', 'c', 'd', 'e', 'f']`.
 - b. En utilisant la méthode `reverse`, inversez l'ordre des éléments de `L1`.
 - c. En utilisant la fonction `zip`, créez la liste
`L3 = [('a', 12), ('b', 10), ('c', 8), ('d', 6), ('e', 4), ('f', 2)]`
4. `L[a:b:p]` correspond à la sous-liste de la liste `L` contenant les éléments de `L` d'indices `a`, `a+p`, ..., `a+kp` avec `k` maximal tel que `a+kp` reste dans l'intervalle `[a,b[` : en particulier, si `p > 0` et `b > a`, `b-p ≤ a+kp < b`. Chaque paramètre `a`, `b` ou `p` peut être omis ; le pas `p` par défaut est 1, et les bornes sont les extrémités de la liste. Par exemple, `L[:4]` est la sous-liste formée des 4 premières cases de `L`, alors que `L[3::-1]` est son image miroir.
 - a. Affichez la sous-suite de `L3` contenant les éléments d'indices 2 à 4.
 - b. Affichez la sous-suite de `L3` contenant les éléments d'indices impairs.

- c. Copiez la liste L3 dans une nouvelle liste L4.
5.  Dans le fichier `tp1_ex3.py`, complétez la fonction `somme_impairs(x)` qui calcule la somme de tous les entiers impairs de 1 à x en effectuant des additions.
 6.  Complétez la fonction `test_somme(n)` qui vérifie pour chaque entier positif x inférieur à n que la somme des nombres impairs de 1 à x est égale à $(x/2)^2$ si x est pair et à $((x+1)/2)^2$ si x est impair.
 7. Si vous importez le fichier `tp1_ex3.py` dans l'interpréteur Python, qu'affiche l'aide en ligne `help(tp1_ex3.somme_impairs)` ? Et celle de `testDataSomme` ? Regardez le code de ce fichier pour comprendre la différence.

Exercice 4 : opérations arithmétiques revisitées

Lors du premiers cours, vous avez vu plusieurs algorithmes pour les opérations d'addition ($n_1 + n_2$) et de multiplication ($n_1 \times n_2$) sur les entiers. Le but de cet exercice est de vous faire calculer le nombre d'opérations élémentaires utilisées par chaque algorithme, afin de comparer leur complexité. Pour cela vous aurez à compléter le fichier `tp1_ex4.py`.

1.  Complétez la fonction `addition(nb1, nb2)` qui effectue l'addition d'entiers vue en cours, pour qu'elle retourne, en plus du tableau résultat, le nombre d'opérations arithmétiques élémentaires effectuées.
2.  Complétez la fonction `additionV(nb1, nb2)` qui reprend la fonction d'addition écrite ci-dessus mais de façon à ce qu'elle puisse également s'appliquer à des tableaux de tailles différentes. Cette modification change-t-elle le nombre d'opérations élémentaires effectuées ?
3.  Complétez les fonctions qui effectuent la multiplication d'entiers vue en cours pour qu'elles retournent, en plus du tableau résultat, le nombre d'opérations arithmétiques élémentaires effectuées lors d'un appel à chacune d'entre elles.

Tester ces fonctions sur de grands nombres en ajoutant des tests dans les fonctions `testDataMul`. Ces mesures reflètent-elles les complexités vues en cours ?