TP1 : Introduction au langage Python

CPES « Sciences des données, arts et cultures » : Introduction à l'algorithmique

Antoine GAUQUIER
antoine.gauquier@ens.fr

Pierre SENELLART
pierre@senellart.com

5 septembre 2023

Le but de ce premier TP est de découvrir (ou, pour certaines et certains, réviser) les notions élémentaires du langage Python. Le TP se découpe en deux parties. Une première partie introduit ces notions élémentaires au travers de petits exercies, et sera corrigée en classe. Une seconde partie propose des exercices plus approfondis, qui ne sera pas corrigée en classe, pour celles et ceux qui souhaitent aller un peu plus loin. Un corrigé sera toutefois déposé sur la plateforme Moodle quelques jours après le TP.

Note importante : Il sera nécessaire dans le cadre de ce cours de posséder votre propre ordinateur **portable** afin de pouvoir d'une part suivre correctement les TD/TP, mais aussi d'autre part réaliser les devoirs maisons qui vous seront demandés. Si vous ne possédez pas d'ordinateur personnel portable, nous vous invitons à contacter la scolarité du CPES, afin qu'un ordinateur vous soit prêté.

Installation d'un environnement de développement intégré pour le langage Python

Dans le cadre de ce cours, nous utiliserons un environnement de développement intégré (en anglais, *Integrated Development Environment*, plus communément appelé IDE) afin de faciliter l'écriture du code Python. Nous avons fait le choix de vous présenter l'IDE *Visual Studio Code* (plus communément appelé *VS Code*). Cet environnement a l'avantage d'être supporté pour les systèmes d'exploitation *MacOS*, *Windows* et *Linux*. Vous avez la liberté d'utiliser l'IDE de votre choix, cependant *VS Code* sera le **seul IDE** présenté dans le cadre de ce cours, en particulier concernant certaines fonctionnalités telles que le débogage.

Étape 1 : Installation de Python

Nous allons utiliser la dernière version de Python (3.11). Elle peut être téléchargée à l'adresse `https://www.python.org/`. Si vous avez Windows ou MacOS, nous vous recommandons d'installer directement les fichiers *installer* et de simplement suivre les étapes indiquées.

Si vous utilisez Linux, Python 3.11 est probablement disponible via le système de paquets de votre distribution (`apt`, `rpm`, etc.) ; sinon, il vous faudra télécharger une archive (un fi-

chier avec l'extension `.tgz`). Une fois téléchargée, il faut la décompresser, avec la commande `tar zxvf archive.tgz`.

Étape 2 : Téléchargement de Visual Studio Code

Visual Studio Code est téléchargable sur `https://code.visualstudio.com/download`. Vous trouverez des versions pour les trois OS présentés plus tôt. Pour Windows, privilégiez « User Installer » (ou cliquez directement sur le bouton sous le logo Windows). Pour les utilisateurs de Linux sous Ubuntu ou Debian, privilégiez la version en `.deb` qui vous simplifiera l'installation. Enfin, pour les utilisateurs de MacOS, privilégiez la version en `.zip` non-CLI. Il suffit ensuite de suivre les instructions et étapes d'installation qui vous seront indiquées.

Étape 3 : Configuration de Visual Studio Code pour le langage Python

La dernière étape consiste à installer une extension sur *Visual Studio Code* qui supporte le langage Python. Celle-ci peut se trouver ici : `https://marketplace.visualstudio.com/items?itemName=ms-python.python`, et peut se trouver directement dans *Visual Studio Code*, dans l'onglet « EXTENSIONS ».

Il est possible que cette extension soit déjà pré-installée sur l'IDE : dans ce cas, cette étape ne vous concerne pas.

Votre IDE est maintenant prêt pour le langage Python. Vous pouvez trouver des détails ou configurations supplémentaires sur le tutoriel officiel de *Visual Studio Code* concernant l'utilisation de Python : `https://code.visualstudio.com/docs/python/python-tutorial`. Vous débuterez le TP par l'utilisation de Python en mode interactif : il vous faut donc ouvrir un terminal (En haut à gauche de la fenêtre, onglet « Terminal » puis « New Terminal ».

Notions élémentaires de langage Python

Cette première série d'exercices porte sur les notions élémentaires de langage Python. Nous allons y aborder quelques notions fondamentales communes à grand nombre de langages de programmations, que sont les *opérateurs* pour les calculs entre nombres, les *variables* et les types de données, les structures de contrôle élémentaires ainsi que les *fonctions*. Pour chacune de ces notions, quelques paragraphes vous expliquent à quoi elles correspondent et comment s'en servir, puis un exercice avec plusieurs questions vous permet de vous exercer.

Python en tant que calculatrice : opérations arithmétiques entre nombres

Il existe deux manières d'exécuter du code Python. La première consiste à écrire du code dans un fichier, puis d'exécuter ensuite le code écrit (en utilisant un IDE par exemple, ou en ligne de commande). La deuxième consiste à utiliser Python de manière interactive, de la même façon que les calculatrices : on écrit une ligne de code Python que l'on envoie pour exécution, la ligne est ensuite exécutée par l'interpréteur, et une sortie est retournée (si elle existe) sur la sortie standard. Il est ensuite possible de répéter le cycle autant de fois que souhaité, en sachant que l'environnement Python reste ininterrompu (les variables déclarées persistent en mémoire). Afin de différencier l'entrée de la sortie, trois chevrons ainsi qu'un espace sont affichés lorsque Python est en attente de l'entrée de l'utilisateur (`>>>`).

Ce mode est particulièrement intéressant pour s'exercer à manipuler les *opérateurs*. Les opérations *arithmétiques* sur les nombres en Python suivent les règles de priorités opératoires usuelles, et incluent donc également l'utilisation des parenthèses. On y retrouve les 4 opérateurs de base (addition : `+`, soustraction : `-`, multiplication : `*` et division : `/`). On compte également d'autres opérateurs utiles : la division euclidienne, avec l'opérateur `//`, et le reste de la division euclidienne, avec l'opérateur `%`. Il est également de calculer des puissances en utilisant l'opérateur `**`.

Exercice 1: Manipulation des opérateurs *arithmétiques* sur les nombres.

Question 1. Calculez, avec l'interpréteur interactif de Python, la valeur de $\frac{50-5*6+20}{4}$.

Question 2. Calculez, avec l'interpréteur interactif de Python, la valeur décimale (approchée) de la fraction $\frac{17}{3}$. Vérifiez la valeur obtenue en utilisant les opérateurs de division euclidienne et de reste sur ces mêmes valeurs.

Question 3. Calculez, avec l'interpréteur interactif de Python, la valeur de 2^5 **de deux façons différentes**.

Les variables en Python

Il est possible en Python de déclarer des variables, et de leur affecter une valeur. Ces deux actions sont en réalité faites simultanément lors de l'affectation, puisque, contrairement à d'autres langages de programmation, il est impossible de déclarer une variable sans l'affecter en Python.

Pour rappel, une variable est un espace dans lequel on peut stocker une valeur de façon persistante (dans le champ de l'environnement d'exécution en cours). Afin de déclarer une variable et de lui affecter une valeur, il suffit d'utiliser l'opérateur d'*affectation* `=` entre le nom de la variable (à gauche) et sa valeur (à droite).

Le choix des noms de variable est libre, mais il doit respecter un certain nombre de règles. Premièrement, les noms de variables ne doivent pas contenir de mots réservés par le langage lui-même, qu'on appelle mots-clés du langage. Leur liste complète (pour Python 3.11) peut être trouvée ici : `https://docs.python.org/3.11/reference/lexical_analysis.html#index-13`. Un nom de variable peut comporter des lettres, chiffres ou le caractère `_`, mais ne peut commencer par un chiffre. Par compatibilité avec des environnements différents, il est conseillé de ne pas utiliser de lettres accentuées ou d'alphabet non latin.

Une fois la variable déclarée, il est possible de la réutiliser en lieu et place de la valeur qu'elle contient. Leur utilisation est donc particulièrement pratique afin de généraliser du code (comme nous le verrons plus loin avec les *fonctions*).

Ces variables possèdent un type, qui correspond au type de donnée que celle-ci contient. On compte plusieurs types de base en Python : les nombres entiers (`int`), les nombres à virgule flottante (`float`), les nombres complexes (`complex`), les chaînes de caractères (`str`), les variables booléennes (`bool`) et les types séquentiels (le plus courant étant `list`, mais d'autres existent : `tuple`, `range`, etc.). Il est possible de vérifier le type d'une variable Python, avec la méthode `type(variable)`. Avec ces types et variables, un certain nombre d'opérateurs viennent s'ajouter.

D'abord les opérateurs de *comparaison*, au nombre de huit :
— `<` : strictement inférieur
— `<=` : inférieur ou égal
— `>` : strictement supérieur
— `>=` : supérieur ou égal
— `==` : égalité
— `!=` : différent
— `is` : identité d'objet (équivalent à `==` pour certains types).
— `is not` : contraire de l'identité d'objet (équivalent à `!=` pour certains types).

L'utilisation de ces opérateurs renvoie une valeur booléenne, indiquant si l'expression est vraie (`True`), ou fausse (`False`). On y ajoute trois opérateurs supplémentaires, que sont les opérateurs logiques « ou » (`or`), « et » (`and`) et « non » (`not`). Les opérateurs et fonctions natives propres aux chaînes de caractère seront présentées plus tard dans le cours.

Exercice 2: Manipulation de variables en Python.

Question 1. Définissez trois variables décimales `a`, `b` et `c` de votre choix. Vérifiez qu'elles possèdent bien le type `float`. Calculez leur produit. Déterminez leur classement par ordre croissant en les comparant deux à deux avec l'opérateur de comparaison `<`.

Question 2. On exécute à la suite les lignes de code suivantes : `a = 7`, `b = a - 4` et `a * b`. Quelle valeur sera retournée sur la sortie standard ?

Question 3. On souhaite calculer la distance euclidienne entre les points $A = (3, 4)$ et $B = (6, 8)$. Dans un premier temps, représentez chacun des points par une paire de variable. Ensuite, à partir des variables définies, écrivez la formule permettant le calcul de la distance euclidienne entre ces deux points. Vérifiez le calcul manuellement.

Question 4. Le cercle de centre $C = (x_C, y_C) = (2, 2)$ a un rayon de taille 5. Écrivez, à l'aide d'un opérateur de comparaison, et une fois avoir représenté le point C avec deux variables, l'expression booléenne permettant de vérifier si le point $D = (4, 6)$ est à l'intérieur du cercle en question.

L'écriture de code sur fichier et les structures de contrôle

Jusque là, nous avons vu Python en tant qu'interpréteur interactif, alternant entre notre entrée et la réponse de Python au travers de la sortie standard. Bien que ce mode soit intéressant pour découvrir Python, il est loin d'être d'être pratique, pour deux raisons principales. D'une part, il ne permet pas de déployer du code réutilisable, et d'autre part, certaines structures s'écrivent en plusieurs lignes. Le mode interactif permets leur utilisation, mais elle n'est toutefois pas évidente (en particulier, il est impossible de rectifier une erreur écrite dans une ligne précédente de la même structure). Les structures mentionnées ici sont appelées *structures de contrôle*. Ces structures permettent de faire des *embranchements* dans le code, pour réaliser certaines actions spécifiques. Il en existe deux familles : les structures de contrôle de décision et d'itération.

La première famille permet de faire du *conditionnement*, à savoir d'exécuter du code seulement si une (ou plusieurs) expressions booléennes sont vraies. Les mots-clés utilisés dans

ces structures sont les suivants : `if` suivi de la première condition, `elif` suivi de conditions alternatives si la précédente n'est pas réalisée, et enfin `else`, seul, qui définit le code à exécuter si aucune des conditions précédentes de la structure n'a été satsifaite. Chacune de ces expressions doit se terminer par deux points (`:`), et le code à exécuter doit être précéder d'une *indentation* (c'est-à-dire d'une succession d'espaces). Celle-ci délimite la condition du code à exécuter : il est donc **indispensable** d'indenter correctement son code, car contrairement à d'autres langages où l'indentation ne relève que de l'esthétique et de l'organisation du code, en Python elle est responsable de la délimitation de ces structures de contrôle. En particulier, les instructions suivant les `if`, `elif` et `else` de la même structure de contrôle doivent être toutes indentées du même nombre d'espace, qui doit être supérieur au nombre d'espace indentant les mots-clés `if`, `elif` et `else` (typiquement, 4 caractères espaces).

La deuxième famille permet de faire de l'*itération*. On peut définir l'*itération* comme une répétition finie d'actions. Pour que cette répétition soit finie, il faut définir un critère d'arrêt. Il en existe de deux sortes : les critères sur base d'expression booléenne, auquel cas on utilisera le mot-clé `while` suivi de la condition ; et les critères définissant un nombre d'itération fixé à l'avance, auquel cas on utilisera les mots-clé `for` et `in`, afin de définir l'itéré (la variable qui à chaque nouveau tour de boucle sera itérée) et les limites de valeurs que celui-ci prendra. Une méthode usuellement utilisée est la méthode `range(start, stop, step)`, qui représente une liste commençant à `start` et s'arrêtant juste avant `stop`, dont chaque valeur est espacée de `step` avec la valeur précédente. Ainsi, `range(0, 11, 2)` donnera la liste `[0, 2, 4, 6, 8, 10]`, et la structure de contrôle « `for index in range(0, 11, 2):` » réalisera 6 itérations, où la variable `index` prendra successivement les valeurs 0, 2, 4, 6, 8 et 10. Enfin, comme pour les structures de contrôle de décision, chaque structure d'itération doit se finir par deux points (`:`), et le code associé à la structure doit être indenté.

Pour la suite des exercices, nous écrirons le code Python dans un fichier que l'on exécutera. Cela est facilité par l'utilisation d'un IDE, en l'occurrence ici, *VS Code* si vous utilisez celui du cours. Comme vous n'interagissez plus avec Python à chaque ligne, vous pouvez dès lors aérer votre code (passer des lignes lorsque nécessaire) et éventuellement y ajouter des commentaires (du texte non considéré par l'interpréteur), qui débutent avec le caractère `%`. Enfin, étant donné qu'il n'y a plus d'interaction, la sortie standard n'affichera plus de contenu à l'exécution de chaque ligne. Il vous faut donc manuellement indiquer à Python les éléments que vous souhaitez faire afficher. Pour ce faire, il vous suffit d'utiliser la fonction `print(contenu)`, où contenu correspond à ce que vous souhaitez afficher. Il peut s'agir de variables, de valeurs, ou bien de contenu textuel (à préciser entre `" "` ou `' '`). Si vous souhaitez afficher plusieurs éléments, une bonne pratique est d'écrire `print(contenu1, contenu2, contenu3)` par exemple.

Exercice 3: Manipulation des structures de contrôle en Python

Question 1. Résolvez à nouveau la question 3 de l'exercice 1, en utilisant cette fois-ci une boucle `for` (structure de contrôle itérative). Affichez le résultat à chaque itération de la boucle.

Question 2. Déclarez trois variables a, b et c qui sont des nombres à virgules flottantes (`float`). Écrivez ensuite une structure de contrôle de décision permettant d'afficher les variables dans leur ordre croissant. Par exemple, si `a = 3.14`, `b = 2.72` et `c = 1.62`, alors la sortie standard devrait afficher : `c < b < a` (ne tenez pas compte des cas d'égalité possible ici).

Question 3. À l'aide d'une structure de contrôle de décision, et des opérateurs *arithmétiques*, écrire un programme qui permets de déterminer si un entier (représenté par une variable du nom de votre choix) est pair ou impair. Prévoyez un affichage dans la sortie standard en conséquence.

Question 4. On souhaite déterminer la plus grande valeur entière (et positive) de n telle que $n^2 - 4n$ ne dépasse pas 12. En utilisant une structure de contrôle itérative de type `while`, déterminer cette valeur de n, et affichez-la.

Les fonctions en Python

Nous avons vu jusque là quelques exercices qui tentent de résoudre des problèmes, sous le prisme d'un cas particulier. On pourrait par exemple reprendre les questions 3 et 4 de l'exercice 2, qui, respectivement, calculent la distance euclidienne de deux points de $\mathbb{R}^2$ et vérifient si un point est inclus ou non dans un cercle.

Pour ce faire, il faut envisager une portion de code générique, appelée *fonction*, prenant des *arguments* en entrée (les valeurs concrètes prises par les variables de la *fonction*), et, éventuellement, une valeur de *retour*. Concrètement, pour déclarer une fonction, il faut utiliser le mot-clé `def`, suivi du nom de la fonction. Ce nom de fonction doit respecter les mêmes consignes que pour les noms de variables. Ensuite, il faut préciser, entre parenthèses, les *arguments* de la fonction. Pour le précédent calcul de distance euclidienne, il faudrait par exemple préciser quatre variables correspondant aux deux paires de points. Deux points viennent terminer cette ligne déclarative de la fonction. Et de la même façon que pour les structures de contrôle, le code de la fonction doit être indenté par rapport à la ligne déclarative. Ainsi, on pourrait écrire la ligne déclarative générique suivante : « `def ma_fonction(argument1, argument2, argument3, argument4):` » pour une fonction à quatre arguments. Si la fonction doit retourner quelque-chose, on le fait alors avec le mot-clé `return`, suivi du résultat (de la valeur) à retourner, qui peut être de toute nature.

Afin de les utiliser, il vous suffit, une fois définies, de les appeler. Vous verrez l'utilisation d'une fonction générique d'appel, appelée `__main__`, dans le prochain cours.

Exercice 4: Manipulation des fonctions en Python.

Question 1. Implémentez les questions 3 et 4 de l'exercice 2 sous forme de fonctions. Appelez celles-ci au travers de plusieurs exemples afin de vérifier leur bon fonctionnement.

Question 2. Une année bissextile est une année dont le mois de février court jusqu'au 29. Une année est bissextile si l'une des deux conditions suivantes est remplie : (1) l'année est divisible par 4 sans toutefois être divisible par 100 (cas des années qui ne sont pas multiples de 100), ou (2) l'année est divisible par 400 (pour le cas des années multiples de 100). Implémentez une fonction qui prend en entrée une année, et qui indique en sortie (via un `return` ou bien directement par un affichage dans la sortie standard) si l'année est bissextile. Testez ensuite la fonction pour les années 1900, 2000, 2018 et 2020.

Question 3. Écrivez une fonction permettant l'affichage de la table de multiplication (de 1 à 10) d'un nombre entier passé en argument. Par exemple, l'appel de la fonction pour la valeur 5 doit donner en sortie standard un affichage de 10 lignes, la première étant « `1 * 5 = 5` » et la dernière « `10 * 5 = 50` ».

Pour aller plus loin

Cette seconde série d'exercices est destinée aux étudiants ayant déjà fait de la programmation (et pour qui les notions élémentaires sont donc déjà maîtrisées), ou bien pour ceux souhaitant aller plus loin.

Exercice 5: Récursivité.

Question 1. Implémentez une fonction récursive permettant le calcul de la factorielle de n'importe quel nombre entier positif ou nul. Pour rappel, la factorielle est définie comme étant :

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times \cdots \times (n-1) \times n \quad \forall n \in \mathbb{N}^*, \text{ et } 0! = 1.$$

Question 2. Implémentez une fonction récursive permettant le calcul de la suite de *Fibonacci* (`https://fr.wikipedia.org/wiki/Suite_de_Fibonacci`), définie comme suit :

$$F_n = F_{n-1} + F_{n-2}, \text{ avec : } F_0 = 0 \text{ et } F_1 = 1.$$

Question 3. On considère le problème du calcul de fonds disponible dans un livret d'épargne. La somme initiale sur le livret est représentée par la variable u_0. On considère par ailleurs que l'on ne réinjecte pas d'argent sur le livret ensuite. Chaque année, des intérêts sont versés sur ce dernier, au taux t, tel que $t \geq 1$ (comme ce sont des intérêts). L'objectif est, à partir de u_0 et t, définir la valeur u_n, correspondant au montant disponible sur le livret après n années d'épargne au même taux t fixe. Implémentez une fonction récursive permettant d'effectuer ce calcul. Vérifiez ensuite manuellement, à l'aide d'une *suite géométrique*, que la valeur retournée est correcte.

Exercice 6: Le chiffre de César

Le *chiffre de César* (`https://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage`) désigne une méthode de chiffrement simple utilisée par César pour ses correspondances secrètes. Il consiste à décaler toutes les lettres d'un texte d'un nombre fixe de places dans l'alphabet (on considère ici l'alphabet latin à 26 lettres, mais il faut garder à l'esprit que cette méthode est implémentable avec n'importe quel alphabet, pourvu qu'il soit ordonné : lettres et nombres, caractères spéciaux, lettres accentuées, etc.). Ainsi, avec un décalage de 2, le « A » deviendrait un « C », le « B » un « D », etc. Il est possible d'envisager n'importe quel valeur de décalage, en gardant en tête qu'il existe 26 lettres dans l'alphabet (ainsi, un décalage de 27 revient à un décalage de 1).
Une version particulière de ce décalage est le *ROT-13* (`https://fr.wikipedia.org/wiki/ROT13`), avec une valeur de décalage de 13. Ce mode de chiffrement n'est toutefois pas du tout sécurisé, comme nous allons le voir ici, en codant un *déchiffreur de César*.
Implémentez une fonction qui, prenant en argument une chaîne de caractère, affiche en sortie tous les déchiffrements de César de 1 à 13, qui correspondent donc aux décalages dans le sens inverse alphabétique. Ainsi, en entrant la chaîne « URYYB », vous obtiendrez une première ligne affichant « TQXXA » (décalage de 1), et ainsi de suite jusqu'à la dernière ligne « HELLO », (décalage de 13, et donc ici le message *en clair*, à savoir non-crypté). Essayez ensuite votre fonction sur les exemples suivants : « ZLWA », « PAGLQ », « SBWKRQ », « LYNB » et « GRMUXOZNSOWAK ».