TP01 Introduction à l'environnement de travail

Partie I

Environnement de travail

Le but de ce premier TP est de vous familiariser avec l'environnement de travail que nous allons utiliser toute l'année. Il est constitué de plusieurs composantes :

1. Gitlab (site internet) et GithubDesktop (application sur votre ordinateur) pour tout ce qui permet le suivi du code avec des sauvegardes régulières.
2. Pyzo pour tout ce qui concerne l'exécution de code Python

I.1 Gitlab

Si vous ne l'avez pas encore fait, allez activer votre compte sur `https://gitlab.pcsi-kleber.fr` en suivant la procédure décrite par votre professeur lors du premier cours. Il reste à présent à utiliser l'application GithubDesktop pour faire l'interfaçage avec votre ordinateur. Trois possibilités :

— Si vous êtes sur un des ordinateurs du lycée (salle E21-E22), il y a un lien à suivre directement sur le bureau.

— Si vous êtes sur un des ordinateurs du lycée mais que le lien n'existe pas encore (potentiellement salle E25), il faudra utiliser pour le moment l'interface web pour rajouter vos changements.

— Si vous êtes sur votre ordinateur propre, suivez le lien `https://desktop.github.com` et laissez-vous guider pour l'installation.

Sur l'interface internet du repository que votre professeur a créé pour vous, il devrait y avoir un bouton bleu en haut à droite intitulé « clone » sur lequel il vous suffit de cliquer. Copiez le lien en question (https) puis allez dans GithubDesktop et sélectionnez « Clone a repository ». Allez dans l'onglet « URL » tout à droite, collez le lien à l'endroit adéquat et appuyez sur « Clone ».

Github Desktop vient de récupérer les fichiers et les mettre dans un dossier dédié (chez moi c'est dans /Documents/GitHub/, mais vous pouvez voir où en allant dans le menu « Repository » et en choisissant « Show in Finder » ou l'équivalent sous windows). Nous sommes prêts pour commencer les modification.

I.2 Pyzo

Pour programmer, vous allez avoir besoin de vous familiariser avec votre environnement de développement (IDE en anglais pour Integrated Development Environment). Au lycée, l'environnement choisi est Pyzo. À nouveau deux cas de figure possibles :

— soit vous êtes sur un ordinateur du lycée, auquel cas, il suffit de rechercher le programme Pyzo dans la barre de recherche disponible en bas à gauche de votre écran.

— soit vous utilisez votre propre ordinateur et Pyzo doit être installé une première fois en suivant la procédure indiquée sur la page `http://pcsi.kleber.free.fr/IPT/pyzo.html`.

Commencez par identifier les différentes fenêtres qui vous sont accessibles :

— La fenêtre principale devrait être celle où vous pouvez écrire et enregistrer votre code dans un fichier (dit « fichier source »). Elle possède des lignes numérotées sur la gauche qui vous permettent de vous repérer facilement dans le fichier, notamment quand Python vous signalera l'existence d'un bug dans votre programme.

— La console est le Python interactif, c'est-à-dire que vous pouvez y rentrer des commandes pour tester leur comportement sans que ce soit sauvegardé dans un fichier (elle doit donc être cantonnée au rôle de test rapide). Elle est facilement reconnaissable au symbole `>>>` qui commence chacune des lignes

où vous pouvez rentrer du code. C'est aussi à l'intérieur de celle-ci que sera exécuté le code de votre fichier principal.

— La fenêtre d'aide est un accès direct à l'aide en ligne de Python dans laquelle vous pourrez rechercher diverses informations sur la syntaxe ou les conditions d'utilisation de commandes données.

— Enfin, l'« Inspecteur » vous permet d'accéder directement à certaines parties de votre code une fois que votre fichier aura acquis une certaine structure (en particulier via l'écriture de fonctions)

Avant de commencer à utiliser cet environnement, il est nécessaire de se familiariser avec lui en suivant le cheminement ci-après :

— Ouvrez (via `Ctrl-O` dans Pyzo) le fichier `TP01_decouverte.py` qui se trouve dans le sous-dossier `Mes Documents/GitHub/TP01-votreLoginGithub/` de votre dossier personnel.

— Commencez par *lire* attentivement les 15 premières lignes du fichier qui constituent des commentaires laissés par votre serviteur pour vous aider à comprendre comment est organisé le fichier. Vous n'oublierez pas de toujours lire les commentaires laissés dans les divers fichiers à votre disposition [1].

— Écrivez [2] `print("Hello World !")` dans la fenêtre principale (n'oubliez pas les apostrophes ni les parenthèses) sur la ligne 29.

— Repérez dans les menus comment exécuter votre code et retenez le raccourci clavier. **ATTENTION :** pour que les chemins d'accès soient correctement initialisés, il faut sélectionner « Exécuter en tant que script » ou « Run file as script » (`Ctrl-Shift-E`) [3]. Faites-le et vérifiez que `Hello World !` s'imprime bien dans la console. Si au lieu de cela, il y a un message d'erreur, lisez-le bien et corrigez votre source en conséquence. En cas de problème, demandez des conseils au personnel encadrant.
NB : l'exécution en tant que script force Pyzo à enregistrer votre fichier, il est donc automatiquement sauvegardé [4].

— Décommentez [5] la ligne permettant de tester votre première fonction (sans l'avoir modifiée pour le moment) et exécutez le script. Python devrait vous tancer vertement.

```
Hello World !
test_hello_world (test_TP01.HelloTest) ... FAIL

======================================================================
FAIL: test_hello_world (test_TP01.HelloTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File ".../Info/TP01/test_TP01.py", line 15, in test_hello_world
    self.assertEqual(hello_world(),"Hello World !")
AssertionError: "Mais il va falloir changer des choses..." != "Hello World !"
- Mais il va falloir changer des choses...
+ Hello World !
----------------------------------------------------------------------
Ran 1 test in 0.121s
```

1. Si je les écris, ce n'est pas seulement pour faire joli, mais aussi pour qu'ils soient lus !
2. Vous pourrez après cet essai quasiment oublier l'instruction `print` qui ne vous servira que pour des débuggage et plus pour le travail courant.
3. Rappel : `Shift` est la touche qui vous permet d'écrire des majuscules quand vous la maintenez enfoncée (à ne pas confondre avec CapsLock...)
4. Si vous n'avez pas oublié le SHIFT, bien sûr...
5. C'est-à-dire supprimez le caractère « `#` » placé devant.

```
FAILED (failures=1)
```

Get used to it ! Il va falloir apprendre à décortiquer les messages d'erreur pour comprendre ce qui a pu tourner de travers. Ici, la fonction `hello_world` qui est prédéfinie ne renvoie pas la chaîne attendue par les procédures de test. Il va donc falloir la modifier.

— Retournez dans votre fichier source `TP01_decouverte.py` et modifiez-le pour que la fonction `hello_world` renvoie bien `"Hello World !"` et non la chaîne qui y est présente

```
1 def hello_world():
2     '''Normalement, cela doit renvoyer Hello World !'''
3     return "Hello World !"
```

— Exécutez à nouveau (`Ctrl-Shift-E`) votre fichier. Python devrait à présent vous signaler que tout va bien

```
test_hello_world (test_TP01.HelloTest) ... ok
----------------------------------------------------------------------
Ran 1 test in 0.012s
OK
```

— Vous savez à présent comment ouvrir/enregistrer des fichiers, lancer des tests, etc.Vous êtes donc prêts pour affronter la suite.

Partie II

Quelques exercices d'entraînement

Le fichier `TP01_decouverte.py` est déjà préécrit pour contenir des tests pour les objets qu'il vous faut définir et les fonctions qu'il vous faut écrire. Pour lancer les tests concernant un exercice particulier, il vous suffit de décommenter la ligne correspondante afin de vérifier votre avancée. Continuez à écrire dans le même fichier `TP01_decouverte.py` et suivez le guide... On n'oubliera pas de commenter abondamment toutes les fonctions créées.

II.1 Assignation

Dans un programme informatique, il est utile de pouvoir assigner une valeur à une variable (par exemple `ma_variable`), c'est-à-dire stocker cette valeur et pouvoir l'utiliser plus tard dans le programme sans avoir à connaître *effectivement* sa valeur. Savoir que l'ordinateur peut retrouver la valeur si on en a besoin suffit à notre bonheur.

En Python, l'assignation se fait à l'aide de l'opérateur =. L'assignation se fait toujours de la droite *vers* la gauche, c'est-à-dire que la valeur à stocker doit *toujours* être du côté droit du signe égal alors que le nom de la variable dans laquel on veut stocker cette valeur est *toujours* du côté gauche. Contrairement aux égalités que vous avez eu l'habitude de manipuler en mathématiques ou en physique, l'assignation informatique n'est *pas* symétrique et par conséquent, on ne pourra jamais trouver d'expression mathématique (avec des +, des -, des $*$ ou encore des /) du côté gauche d'une assignation, puisqu'elles ne constituent pas des noms de variables licites [6].

6. Un nom de variable licite commence forcément par une lettre (majuscule ou minuscule) ou un underscore (_) et ne contient que des lettres (majuscules ou minuscule), des chiffres ou des underscores. ATTENTION, ne *jamais* mettre d'accent dans un nom de variable : cela ne déclenchera peut-être pas d'erreur sur votre machine, mais cela peut le faire sur d'autres, notamment celle de votre correcteur...

1. Stocker la valeur 42 dans la variable `ma_variable`.

Stop Gitlab
Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Une fois qu'une variable a été définie, on peut l'utiliser dans la suite du programme, par exemple pour définir d'autres variables. On dispose ainsi de certains outils de calculs comme l'addition (+), la soustraction (-), la multiplication (*), la division (/), la division entière (//) qui donne le quotient de la division euclidienne telle que 3//2 vaille 1, la mise en puissance (**, par exemple 3**2 vaut 9), le modulo (%, par exemple 9%5 vaut 4)

2. Stocker dans la variable `reponse_grande_question_de_l_univers` le résultat de la suite d'opérations suivante :
   — Élever `ma_variable` à la puissance 5 ;
   — Diviser le résultat par le carré de {l'année du début de la révolution française auquel on a soustrait le carré de 5}.
3. Faire de même dans la variable `encore_une_reponse` avec la suite d'opérations :
   — Prendre l'année de début de la révolution française ;
   — La multiplier par 10 ;
   — Ajouter 2 ;
   — Prendre le résultat modulo 50.

Stop Gitlab
Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

II.2 Instructions conditionnelles

Arrivé à un certain stade du programme, la suite peut dépendre de la valeur actuelle d'une certaine variable. Par exemple, si celle-ci est supérieure à une valeur donnée, on exécutera telles instructions (exemple : si un bagage est trop lourd, on signale une surcharge) alors que sinon, on en exécute d'autres (s'il n'est pas trop lourd, il sera correctement pris en charge). En Python, cela se traduit par la structure `if`/`else`. Sur l'exemple précédent, cela donne

```
1 if poids_bagage > 20:
2     prise_en_charge = 'Surcharge !'
3     # Mettre ici le code de ce qu'il faut faire en cas de surcharge
4     # qui pourra s'étendre sur plusieurs lignes à condition de garder
5     # la même indentation que celle initialement choisie
6 else:
7     prise_en_charge = "C'est bon, on passe à la suite"
8     # Mettre ici le code en cas de non surcharge.
9     # Mêmes limitations que précédemment.
```

On voit ici poindre une notion qui sera très importante pour la suite : la notion de bloc. Python est un langage à *indentation significative*, c'est-à-dire que l'on contraint un ensemble d'instruction dans un « bloc » en gardant le même niveau d'indentation. Un nouveau bloc commence toujours après une instruction qui se termine par un double point « : », d'où l'obligation d'utiliser un double point après la condition vérifiée par le `if` (soit `if condition:`), de même qu'après le `else` (soit `else:`).

Pour pouvoir écrire des instructions conditionnelles, vous aurez besoin de pouvoir comparer deux éléments, ce que vous pouvez faire avec les expressions suivantes :

- `x > y` renvoie `True` si `x` est strictement plus grand que `y` ;
- `x >= y` renvoie `True` si `x` est plus grand ou égal à `y` ;
- `x < y` renvoie `True` si `x` est strictement plus petit que `y` ;
- `x <= y` renvoie `True` si `x` est plus petit ou égal à `y` ;
- `x == y` renvoie `True` si `x` est égal (ou identique si on ne compare pas de simples nombres) à `y` ;
- `x != y` renvoie `True` si `x` est différent de `y` ;

4. Stocker dans la variable `x` la valeur préalablement stockée dans la variable `valeur_mystere` que Python connaît déjà (attention, comme il se doit, il n'y a pas d'accent dans le nom de la variable).
5. Si `x` est un multiple de 5, stockez dans la variable `resultat_if` la chaîne de caractères `"x est un multiple de 5."`. Sinon, si `x` est un multiple de 3, stockez dans la même variable la chaîne `"x est un multiple de 3."`. Sinon stockez la chaîne `"Pas de chance..."`

Notez que, dans l'exercice précédent, si `x` est à la fois multiple de 5 et multiple de 3, il ne sera estampillé que « multiple de 5 ».

Stop Gitlab

Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

II.3 Définir une fonction

Une fonction est un objet qui prend des valeurs en paramètres et, après calculs, manipulations et autres, renvoie un autre objet via l'instruction `return`. Un fonction correspond donc à un « bloc » d'instructions que l'on pourra « appeler » plus loin dans le programme, ce qui évitera de réécrire plusieurs fois de suite les mêmes instructions ou types d'instructions. Une fonction s'introduit avec le mot-clé « `def` » et il ne faudra pas oublier le « : » en fin de première ligne pour dire à Python qu'un nouveau bloc commence. Exemple d'une fonction prenant 3 paramètres en entrée :

```
1 def ma_fonction(param1,param2,param3):
2     instruction1
3     instruction2
4     return objet_a_renvoyer
```

6. Définir une fonction `verification_surcharge` qui prenne un seul argument `poids_bagage` (exprimé en kg) et qui vérifie s'il y a ou non surcharge avant d'embarquer un bagage dans la soute d'un avion. La fonction doit renvoyer
   - `"Surcharge !"` si le poids excède strictement 100 kg ;
   - `"Surtaxe..."` si le poids appartient à]20 kg ; 100 kg] ;
   - `"Tout va bien."` si le poids appartient à [0 ; 20 kg] ;
   - `"What the hell !?!"` sinon.

Dans l'exercice précédent, on pourra utiliser la construction en `elif` (contraction de `else` et de `if`) qui s'insère (autant de fois que l'on veut) entre le premier `if` et le `else` final pour rajouter une condition à vérifier au cas où toutes les précédentes auraient échouées.

```
if condition1:
    a_executer_si_condition1_vraie
elif condition2:
    a_executer_si_condition2_vraie_mais_condition1_fausse
elif condition3:
    a_executer_si_condition3_vraie_mais_condition2_et_condition1_fausses
else:
    a_executer_si_conditions_1_a_3_fausses
```

II.4 Faire des boucles

L'intérêt de la programmation est de pouvoir répéter des jeux d'instructions automatiquement. Pour ce faire, on peut par exemple utiliser une boucle `for` dont la syntaxe est la suivante :

```
n = 10                    # Initialisation du nombre de tours de boucle
for compteur in range(n): # On donne successivement à compteur les valeurs 0 à n-1
    print(compteur)       # Affichage de la valeur courant du compteur à l'écran
    # Bien sûr, vous ne devrez pas utiliser print vous-même, c'est juste pour
    # vous montrer explicitement ce qui se passe ici...
```

La fonction `range` permet de faire plein de choses, mais dans sa configuration simple précédente, elle donne à la variable `compteur` toutes les valeurs entre `0` et `n-1` : on en a donc bien `n` au total. On peut alors utiliser la valeur du compteur dans la boucle.

7. Calculer [7] à l'aide d'une boucle la somme des entiers de 1 à 100 et stocker le resultat dans la variables `somme_entiers_1_a_100`.

Stop Gitlab

Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

8. Calculer de même la somme des carrés des entiers de 1 à 100 et stocker le résultat dans la variable `somme_des_carres_1_a_100`.
9. Stocker dans la variable `somme_divisibles_par_333_et_813` la somme des entiers entre 1 et 1 000 000 qui soient à la fois divisibles par 333 et 813.

Stop Gitlab

Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

7. Pour la petite histoire, il paraît que l'instituteur de Gauss aurait demandé à ses élèves de faire ce calcul pour avoir la paix pendant une bonne heure (faire les sommes successives à la main, c'est long...), mais Gauss a mis une dizaine de secondes à trouver la réponse. Sauriez-vous comment il a fait ?

II.5 Quelques mélanges

Les exercices suivants vont toujours demander de définir une fonction (car cela permet de tester plus facilement les différents comportements possibles) mais peuvent faire intervenir n'importe laquelle des notions abordées précédemment.

10. L'auberge dans laquelle vous avez prévu de passer la nuit ce soir propose des tarifs très intéressants, pour peu que l'on n'arrive pas trop tard. En effet, plus on arrive tôt moins on devra payer. Vous devez construire une fonction (nommée `prix_auberge`) vous donnant directement le prix à payer en fonction de votre heure d'arrivée (entier entre 7 et 24 [l'auberge ferme passé minuit]). Le prix de base est de 10 euros plus 5 euros pour toute heure après midi. Il ne peut cependant pas dépasser 53 euros. Votre fonction doit renvoyer un entier qui est le prix à payer (en euros) correspondant à l'heure d'arrivée donnée.

Stop Gitlab
Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

11. Vous arrivez devant un pont que vous devez absolument emprunter pour arriver avant la nuit au village situé de l'autre côté de la rivière. Cependant, la traversée du pont n'est pas gratuite et le tarif dépend de votre chance au jeu. En effet, le gardien vous demande de lancer deux dés et le prix dépendra des valeurs que vous obtiendrez. Vous devez écrire une fonction `prix_peage` pour vérifier qu'il applique bien le bon tarif. Celle-ci reçoit deux entiers compris entre 1 et 6 en arguments : la valeur de chaque dé. Si la somme est supérieure ou égale à 10 alors vous devez payer une taxe spéciale (36 euros), sinon vous payez deux fois la somme des valeurs des deux dés. Votre fonction doit non seulement renvoyer le prix à payer, mais aussi une chaîne de caractère contenant soit `"Special tax"`, soit `"Regular tax"` selon le cas qui se présente [8].

Stop Gitlab
Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

12. Un personnage important de la cité n'est pas rentré chez lui hier soir et tout le monde est à sa recherche. Or tout habitant de la ville a un numéro unique qui lui est associé et doit signer une sorte de registre quand il sort de la ville. Vous souhaitez savoir si le registre a été signé, auquel cas il faudra étendre les recherches à l'extérieur de la ville.
    Vous devez écrire une fonction `recherche` qui prend deux paramètres en entrée : un entier (`numero`), le numéro d'une personne recherchée, puis une liste [9] (`registre`) qui correspond aux signatures du registre des sorties. Si le numéro est présent dans la liste (il peut l'être plusieurs fois) vous devez renvoyer le texte `"Sorti de la ville"` sinon `"Encore dans la ville"`.

Stop Gitlab
Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

8. Vous pouvez faire une chose pareille avec l'instruction `return prix,remarque` où `prix` contient la valeur en euros et `remarque` est la chaîne de caractère associée.

9. Une liste est un ensemble d'objet (par exemple des entiers, mais on peut tout y mettre) auquels on peut accéder de différentes manières. Dans cet exemple, le mieux est de prendre un à un chaque élément à l'aide d'une boucle `for` en utilisant la syntaxe suivante autorisée en Python : « `for element in liste:` » qui à chaque tour de boucle met dans la variable `element` le prochain élément de la liste `liste` (ici, vous voudrez certainement changer les noms de variables en « `for nb in registre:` » et utiliser la variable `nb` dans le corps de la boucle)

13. Grâce à un certain nombre d'informateurs plus ou moins fiables, le chef de la police a recueilli des indications qui devraient lui permettre enfin de démasquer cet espion qui lui échappe depuis des semaines. La population de la ville étant relativement importante, il vous demande votre aide afin d'automatiser un peu les choses. On va commencer par écrire une fonction `est_espion` qui estime la probabilité qu'une personne soit un espion, on verra plus tard comment étendre cela à toute la ville. La fonction reçoit cinq arguments (3 entiers et 2 booléens) : la taille en cm de la personne, l'âge en année, le poids en kg, un booléen (`True` ou `False`) qui dit si la personne possède un cheval et un dernier booléen qui dit si la personne a des cheveux bruns. On veut déterminer à quel point la personne concernée correspond aux 5 critères suivants caractéristiques de l'espion :
    - il aurait une taille supérieure ou égale à 178 cm et inférieure ou égale à 182 cm,
    - il aurait au moins 34 ans,
    - il pèserait strictement moins de 70 kg,
    - il n'a pas de cheval,
    - il a les cheveux bruns.

    Lorsque cela n'est pas précisé explicitement, les inégalités sont au sens large.
    La fonction doit tester tous les critères et s'ils sont vérifiés tous les 5, elle doit renvoyer `"Tres probable"`. Si seulement 3 ou 4 sont vérifiés, elle doit renvoyer `"Probable"`. Si aucun n'est vérifié elle doit renvoyer `"Impossible"` et dans les autres cas, elle doit renvoyer `"Peu probable"`.

Stop Gitlab

Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

14. Maintenant que votre fonction de vérification d'espion est opérationnelle, le chef de la police voudrait étendre les vérifications à l'ensemble des habitants de la ville. Il vous demande d'écrire une fonction `verification_ville` qui prend une liste `infos_habitants` en arguments qui, pour chaque habitant, donne les cinq informations nécessaires à l'usage de votre fonction `est_espion`. En utilisant la fonction précédente, étendre vos recherches à la ville entière[10] en renvoyant une liste contenant pour chaque habitant la mention « Très probable », « Probable », « Peu probable » ou « Impossible » correspondante. NB : On peut construire une liste « au vol » avec la méthode `.append`. L'exemple suivant (à adapter selon vos besoins) construit une liste qui contient les 10 premiers entiers pairs en partant d'une liste vide `[]` et en rajoutant un élément à la fin à chaque tour de boucle à l'aide de `append` :

```
1 L = []                # Initialisation de la variable L à une liste vide
2 for i in range(10): # On itère sur 10 entiers de 0 à 9
3     L.append(2*i)    # et on stocke à chaque fois le double dans la liste L
```

Stop Gitlab

Allez sur Github Desktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

10. On a ici affaire à une liste de listes, c'est-à-dire une liste `infos_habitants` où chaque élément de cette liste est elle-même une liste de 5 éléments. Python permet de faire des « assignations simultanées » pour récupérer facilement des informations stockées dans une liste de taille donnée. Par exemple si `sous_liste` contient les 5 informations, alors vous pouvez utiliser une construction du type `taille,age,poids,cheval,brun=sous_liste` et chaque variable aura pour valeur l'élement correspondant dans la sous-liste.