Mise en route (séance double)

S. Chenevois, L. Jouhet, F. Junier,M.Rebout d'après un TP de S.Gonnord

Buts du TP

- ☞ Prendre en main les environnements de travail (Windows et Python3) qui vont vous accompagner pendant au moins deux ans.
- ☞ Découvrir Python3 : en mode « interprété ligne à ligne » et en mode « exécution d'un fichier » via les environnements Idle, Pyzo ou Spyder.
- ☞ (Commencer à) comprendre les notions de variable et d'affectation.
- ☞ Les plus rapides pourront découvrir les notions de boucle et de test.

1 Découverte de l'écosystème

1.1 Windows et Scribe

Les ordinateurs du lycée fonctionnent avec le système d'exploitation Windows (32 bits ou 64 bits, XP ou Seven selon les machines). Ils sont interconnectés en réseau : un ordinateur central joue le rôle de serveur pour les autres machines appelées stations de travail. L'architecture logicielle du réseau est Scribe : le serveur est sous Linux et gère un réseau de stations sous Windows.

EXERCICE 1 *Se loguer au réseau Scribe ; lors de la première connexion changer immédiatement de mot de passe si ce n'est déjà fait.*

Par défaut le login est de la forme `prenom.nom` *ou* `p.nom` *ou* `prenom.n` *pour les noms ou prénoms composés ou trop long et le mot de passe est la date de naissance au format JJMMAAAAA.*

L'interface graphique utilisateur est celle de la version de Windows utilisée avec quelques spécificités dans les actions permises à un utilisateur élève.

Les différents raccourcis du bureau ou du menu Démarrer permettent de lancer les applications disponibles.

Le lecteur local C : n'est pas visible pour les élèves. Les espaces offerts à l'utilisateur élève sur les disques durs du serveur pour stocker et gérer ses fichiers de données sont dénommés *lecteurs réseaux* :

- Lecteur Commun `T :`

  Il contient un répertoire [1] `Logiciels` avec les petits logiciels portables et un répertoire `Travail` en lecture seule pour les élèves. Ce dernier contient des documents destinés à tous les utilisateurs comme la plaquette de présentation de Scribe.

- Lecteur Personnel `U :`

  La racine de ce répertoire est accessible depuis le raccourci « Mes Documents » (dans le bureau ou l'explorateur de fichier). L'élève et son professeur peuvent lire et écrire dans ce répertoire à une exception près, son sous-répertoire `prive` n'est visible que de l'élève. Le professeur peut déposer/récolter un devoir dans le sous-répertoire `devoirs`.

- Lecteur Groupes `S :`

  Celui-ci contient au moins le répertoire de la classe qui contient deux sous-répertoires : `donnees` en lecture seule pour l'élève et lecture/écriture pour le professeur ; `travail` en lecture/écriture pour tous.

  Le professeur peut par exemple déposer un énoncé de devoir dans le répertoire `donnees` et les élèves peuvent rendre leur travail dans `travail` même s'il est préférable d'utiliser le sous-répertoire `devoirs` de leur lecteur personnel `U:`.

1. Selon les écoles, on parle de répertoire / dossier et sous-répertoire/sous-dossier

EXERCICE 2 *Visiter l'arborescence des fichiers de son lecteur Personnel `U :` . Faire un petit schéma (papier/crayon!) la représentant.*

EXERCICE 3 *Dans un dossier personnel de documents, créer un sous-dossier consacré à Python. Dans ce dernier, créer un sous-dossier consacré aux TP de première année, et dans ce sous-dossier, créer un sous-dossier consacré à ce TP.*

EXERCICE 4 *Reprendre l'exercice précédent, mais en donnant cette fois des noms de dossiers sans accent ni espace ou caractère spécial (du type `@` ou `&`). Par contre, le tiret `-` (« moins ») et le tiret-bas `_` (« underscore ») sont autorisés.*

1.2 La dynamique du CTRL-C

Dans les combinaisons de touche, du type « CTRL-C » ou « ALT-(», les deux touches n'ont PAS des rôles symétriques. À 18 ans, il n'est pas trop tard pour comprendre ça (10 ans plus tard, ce sera plus difficile). Vous attribuez à « doigt 1 » et « doigt 2 » deux doigts d'une même personne. Au ralenti, ça donne :

- Doigt 1 appuie sur la touche CTRL, et reste appuyé dessus jusqu'à nouvel ordre. Doigt 2 ne fait rien.
- Après un long moment, doigt 2 va finalement taper brièvement mais fermement sur la touche C. Doigt 1 est toujours sur CTRL, puisqu'on ne lui a pas dit de bouger.
- Bien plus tard, doigt 1 libère enfin la touche CTRL.

S'il vous faut dix secondes pour accomplir correctement ce mouvement, ce n'est pas un problème. S'il vous faut un dixième de seconde pour le faire n'importe comment, vous aurez régulièrement des ennuis et ne saurez jamais utiliser un clavier décemment. Normalement très rapidement, il vous faudra une demi-seconde (et probablement moins) pour faire le BON mouvement ; bravo.

Quelques champs d'application du même principe :

- CTRL-C : pour copier un fichier sous Windows, ou un morceau de texte préalablement sélectionné (grisé) sous un éditeur.
- CTRL-X : pour couper (un fichier ou un morceau de texte préalablement sélectionné). Il est mis dans le « presse-papiers », donc peut être ensuite collé.
- CTRL-V : pour coller (un fichier ou un morceau de texte) qui aura préalablement été mis de côté via un *copier* ou un *couper*.
- CTRL-S : pour sauver le fichier sur lequel on est en train de travailler (sur quasiment tous les éditeurs, dont Word, libre/open-office, Idle, Pyzo, Spyder, le bloc-notes Windows...

Enfin, il convient de révéler à certains l'utilisation de quelques touches mystérieuses du clavier. Remontons depuis la touche CTRL à gauche du clavier.

- SHIFT : sert à faire des majuscules... ou bien obtenir un chiffre sur la clavier principal.
- CAPS LOCK : ne sert PAS à obtenir une majuscule, contrairement à ce que pensent certains. Il est utile pour faire n majuscules, avec $n \geqslant 10$. En deçà de dix (bon, trois pour certains), on peut laisser un doigt appuyé sur SHIFT. Ici encore, 18 ans est un âge où on peut encore changer des habitudes absurdes [2].
- La touche dessus CAPS LOCK est la touche de tabulation, qui permet d'écrire grosso modo un bloc d'espace. En Python, les tabulations sont pour la plupart mises automatiquement par l'éditeur, mais certaines sont parfois à ajouter. On préférera alors cette touche à l'utilisation des espaces [3].
- La touche bizarre avec un 2 : elle n'existe pas, en fait ! Oubliez-la (et NON, on ne l'utilisera pas pour calculer le carré de quelque chose).

2. Il semblerait que l'utilisation systématique de CAPS LOCK pour faire UNE majuscule ne soit pas si rare... C'est probablement à cause d'une mauvaise compréhension de la dynamique du SHIFT : « Ben quand j'essaie d'appuyer sur les deux touches en même temps, souvent ça ne marche pas... ». Ben oui, forcément...

3. En utilisation « expert », certains préconisent (imposent !) le contraire. Soit ; mais nous ne ferons pas d'utilisation expert... et en fait, bien des experts préfèrent la tabulation, pour diverses raisons.

- La touche ALT GR à droite de la touche d'espace est celle qui permet d'obtenir les symboles en bas à droite des touches de chiffres : par exemple, on obtient les crochets via ALT GR-5 (ouvrant) et ALT GR-^{o} (fermant).

EXERCICE 5 *Avec un éditeur simple (par exemple le bloc-notes ; mais pas Word ou libre/open-office), créer un fichier contenant 128 fois la phrase « Si $q \neq 1$ alors $1+q+q^2+...+q^N = (1-q^{N+1})/(1-q)$.»*
Pour les symboles mathématiques, on pourra se contenter d'approximations.

Le fichier aura bien entendu été sauvegardé avant même de faire le travail de recopie de la phrase. À ce sujet, et par anticipation :

- Monsieur, ça vaut le coup de sauver, alors qu'on a encore rien écrit ?
- Oui. C'est comme ça ; et on ne discute pas.
- Monsieur, je peux mettre des caractères sympas dans le nom de fichier ? Genre des accents, ou des arobas ?
- Non. C'est comme ça ; et on ne discute pas.

Ces points de vue sont peut-être un peu extrêmes/discutables. Le jour où vous rencontrez quelqu'un qui a déjà écrit plus de 1000 lignes dans un programme, l'a fait tourner sur plusieurs machines, et vous donne un avis contraire, alors écoutez-le (il assurera le SAV si vous avez des ennuis à la fin de votre TIPE et qu'il est trop tard pour renommer décemment des tas de fichiers par exemple...). Ces consignes ne s'appliquent pas pour choisir le nom d'un fichier Word... ou même (pour être honnête) un programme Python destiné à périr corps et biens sitôt le TP terminé. Mais prenez tout de suite de bonnes habitudes !

EXERCICE 6 *Dans le dossier du TP, créer un sous-dossier, et placer dedans quatre copies du fichier précédent.*

1.3 Idle

Lancer Idle. Il apparaît une fenêtre « Python Shell » qui est l'*interpréteur*. C'est dans cette fenêtre qu'on va travailler au tout début. Il existe deux versions de Python, la 2 et la 3 et nous utiliserons uniquement la 3.

EXERCICE 7 *Taper dans l'*interpréteur *les expressions suivantes ;* Il faut taper ces 15 expressions sur 15 lignes différentes, et sans point-virgule. La fenêtre Idle aura donc l'aspect suivant :

```
>>> 2 + 5
7
>>> 2 * 5
10

2+5;   2*5;   2**5;   2.1/3.5;   2.2/3.5;   17.0/5.0;   17 // 5;   17 / 5;   17 % 5 ;
-3//2; 3//-2; -(3//2); (-3)%2; 3%(-2); -(3%2)
```

Depuis l'interpréteur, on peut créer une nouvelle fenêtre : soit via « CTRL-N » soit via le menu Fichier (ou File). Cette nouvelle fenêtre est une fenêtre d'*édition*. Ce qu'on tape dedans est appelé à être sauvé (en fait IMMÉDIATEMENT) puis exécuté.

EXERCICE 8 *Créer une nouvelle fenêtre. Sauver immédiatement le fichier dans un endroit opportun (ben... un des dossiers créés il y a quelques minutes !), avec un nom raisonnable (qui peut être par exemple :* `TP-01-02.py`*). Ensuite, dans ce fichier, taper les lignes suivantes :*

```
2 * 5
print(3 * 5)
4 * 5
```

Sauvegarder le fichier via CTRL-S, puis l'exécuter via F5. Observer ; comprendre.

Lorsqu'on « quitte » Idle, ce qui est dans l'interpréteur est perdu à tout jamais. Par contre, ce qui a été édité et sauvé... est sauvé ! Et est donc récupérable plus tard.

EXERCICE 9 *Quitter Idle. Le relancer, et charger le fichier créé à la session précédente. Constater qu'il n'a pas changé! L'exécuter. Le modifier, le sauvegarder et l'exécuter.*

Des petits calculs peuvent souvent se faire dans l'interpréteur. Mais dès qu'on veut faire quelque chose d'un tant soit peu structuré, travailler dans l'éditeur est quasiment indispensable.

EXERCICE 10 *Dans l'éditeur, saisir les séquences d'instructions suivantes :*

```
## Commentaire :
## Boucle for 1/2

for i in range(2,5):
    print(i**2)
    print(5//2)
```

```
## Boucle for 2/2


for i in range(2,5):
    print(i**2)
print(5//2)
```

```
## Boucle while 1/2
i = 4
while i > 1:
    print(i**2)
    i = i - 1
    print(5/2)
```

```
## Boucle while 2/2
i = 2
while i < 5:
    print(i**2)
    i = i + 1
print(5/2)
```

Sauver, exécuter, comprendre.
Essayer de taper et exécuter cette boucle en restant dans l'interpréteur.

1.4 Pyzo et Spyder

On passe à d'autres *Integrated Development Environment* (IDE), à savoir Pyzo ou Spyder.

EXERCICE 11

1. *Quitter Idle ; lancer Pyzo ou Spyder.*
2. *Sous Pyzo, une fenêtre pop-up s'ouvre, choisir le Français, fermer la fenêtre et Pyzo, pour enregistrer le changement dans le fichier de configuration. Relancer Pyzo et suivre le tutoriel de démonstration.*
3. *Repérer les différentes fenêtres (éditeur, shell/interpréteur, explorateur de fichier, affichage de la documentation . . .).*
4. *Ouvrir le fichier* `TP-01-02.py` *créé avec Idle.*
5. *Pour exécuter le fichier, on peut cliquer sur l'icône dans la barre d'outils ou appuyer sur* `F5` *sous Spyder ou* `CTRL + E` *sous Pyzo.*
6. *On peut également exécuter une sélection du code source en appuyant sur* `F9` *sous Spyder ou* `ALT + RETURN` *sous Pyzo.*
7. *Il est pratique de découper le code source d'un TP en cellules délimitées par* `#%%` *sous Spyder ou* `##` *sous Pyzo. On peut alors exécuter uniquement le code de la cellule courante avec* `CTRL + RETURN` *sous Pyzo ou en cliquant sur l'icône dédiée sous Spyder.*
8. *Les outils explorateur de variable sous Spyder ou Workspace sous Pyzo permettent de suivre les valeurs et les types des variables.*
9. *Enfin, sous Spyder, le répertoire de travail est le répertoire où se trouve le fichier source alias le script mais sous Pyzo c'est le répertoire personnel de l'utilisateur sur la machine (* `/home/utilisateur` *sous Linux). Si le script utilise des ressources comme des fichiers, il faut définir leur chemin d'accès soit avec un chemin absolu (déconseillé car la portabilité risque d'être cassée), soit avec un chemin relatif par rapport au répertoire de travail. Dans ce cas, il faut exécuter une fois le fichier source en tant script avec* `CTRL + SHIFT + E`.

2 Variables

Une *variable* est une zone de la mémoire de l'ordinateur qui

- porte un nom ;
- contient de l'information (une « valeur »).

On commence par *affecter* une variable en Python : `x = 45`
En « pseudo-code », on note plutôt $x \leftarrow 45$.
Ensuite, x contient la valeur 45, donc par exemple $x + 2$ vaut 47.
On peut très bien *réaffecter* à x une autre valeur.
On dit que les affectations *écrasent* les valeurs précédentes.

```
>>> x = 5 * 3
>>> y = 6
>>> x + y
21
>>> x = 36
>>> x + y
42
```

EXERCICE 12 *Taper les instructions suivantes (dans l'interpréteur de Pyzo ou Spyder) :*

```
x = 10;   y = x;   x = 15
```

Encore une fois, les instructions seront tapées sur des lignes différents, sans point-virgule (accepté comme séparateur d'instruction mais son usage n'est pas recommandé en Python) :

```
>>> x = 10
>>> y = x
>>> x = 15
```

Après ces instructions, que valent x et y ? « Deviner »... puis vérifier.

EXERCICE 13 *Quelles sont les valeurs de x et y après les instructions suivantes ?*

```
x = 42;   y = 10;   x = y;   y = x
```

Prédire, puis vérifier.

EXERCICE 14 *Quelles sont les valeurs de x et y après les instructions suivantes ?*

```
x = 42;   y = 10;   z = x;   x = y;   y = z
```

Prédire, puis vérifier.

EXERCICE 15 *Affectation parallèle*

Lorsqu'il rencontre le symbole d'assignation =, Python évalue d'abord l'expression à droite puis l'assigne au nom (ou identificateur) de variable à gauche.
L'expression à droite peut être un tuple et Python peut l'affecter à un tuple d'identificateurs à gauche. On peut parler d'affectation parallèle.
Le module `dis` permet de désassembler le `bytecode` utilisé par l'interpréteur Python en une sorte de langage d'assemblage et permet de mieux comprendre le fonctionnement de l'interpréteur :

```
>>> import dis
>>> dis.dis('a = 2; b = 3; a, b = b, a')
  1           0 LOAD_CONST               0 (2)
              3 STORE_NAME               0 (a)
              6 LOAD_CONST               1 (3)
              9 STORE_NAME               1 (b)
             12 LOAD_NAME                1 (b)
             15 LOAD_NAME                0 (a)
             18 ROT_TWO
             19 STORE_NAME               0 (a)
             22 STORE_NAME               1 (b)
             25 LOAD_CONST               2 (None)
             28 RETURN_VALUE
```

L'affectation parallèle est pratique pour échanger les contenus de deux variables x et y sans variable de stockage ou pour faire des permutations circulaires. On en profite pour vérifier que Python est un langage à typage dynamique.

```
>>> a, b, c = True, 'Hello', 1
>>> a, b, c
(True, 'Hello', 1)
>>> a, b, c = c, a, b
>>> a, b, c
(1, True, 'Hello')
```

1. *Ecrire un script Python qui réalise une permutation circulaire des contenus de trois variables sans affectation parallèle et avec une seule variable de stockage.*
2. *Ecrire un script Python qui réalise une permutation des contenus de deux variables de même type sans affectation parallèle et sans variable de stockage. Tester pour* `x, y = 4, 5`*. Les types des valeurs doivent être conservés. Adapter pour la permutation circulaire de trois variables.*

Ce qui suit est un exercice typique de « DS d'informatique » :

EXERCICE 16 *Dans l'éditeur, taper les lignes suivantes.À l'exécution, que va-t-il se passer ? Prédire, puis vérifier.*

```
x = 10
y = 15
z = x + y
x = y
y = z
print(x + y + z)
```

On termine cette partie en observant deux *types* particuliers : les chaînes de caractères et les tableaux (ou listes).

EXERCICE 17 *Exécuter les commandes ou expressions suivantes dans l'*interpréteur*, une par une :*

```
x = "truc";   y = "bidule";   x[1];   y[2];   x[4];   x[-1];   z = x+y;
z;   t = x+" "+y;   t;   print(t);   print(x,y);   print(x+y);   x[2] = "z";  x
```

EXERCICE 18 *Exécuter les commandes suivantes dans l'interpréteur :*

```
t = [6, 12, 'a'];  t[1];  t[3];  t[0];  t[-1];  len(t);  t[1] = 2;  t;  t+t
```

3 Premières boucles, premiers tests

3.1 Observons

EXERCICE 19 *Dans l'éditeur, taper puis sauver/exécuter les lignes suivantes :*

```
for i in range(10,15):
    print("bonjour ",i)
```

Remplacer `print("bonjour ",i)` *par* `print("bonjour " + i)`*, exécuter.*
Remplacer enfin `print("bonjour ", i)` *par* `print("bonjour " + str(i))` *et exécuter.*

EXERCICE 20 *Dans l'éditeur, taper puis sauver/exécuter les lignes suivantes :*

```
somme = 0
for i in range(1,5):
    somme = somme + i
```

Après exécution, que valent `i` *et* `somme` *?*

EXERCICE 21 *Dans l'éditeur, taper puis sauver/exécuter les lignes suivantes :*

```
#script 1
if 10**2 < 99:
    print("pif 1")

if 5**2 > 20:
    print("paf 1")

if 10**2 < 99 or 5**2 > 20:
    print("plof 1")
```

```
#script 2
if 10**2 < 99:
    print("pif 2")
    if 5**2 > 20:
        print("paf 2")

if 10**2 < 99 or 5**2 > 20:
    print("plof 2")
```

```
#script 3
if 10**2 < 99:
    print("pif 3")
    if 5**2 > 20:
        print("paf 3")
    if 10**2 < 99 or 5**2 > 20:
        print("plof 3")
```

EXERCICE 22 *Dans l'éditeur, taper puis sauver/exécuter les lignes ci-contre :*

```
somme = 0
for i in range(10,18):
    if i%3 == 0:
        somme = somme + i
```

Prévoir la valeur de `somme` *; vérifier.*

3.2 Faisons nous-mêmes

EXERCICE 23 *Écrire un programme qui demande à l'utilisateur de rentrer trois réels et qui les renvoie dans l'ordre croissant. La fonction* `input` *permet de récupérer une entrée clavier. Lire, et comprendre, ce qui se passe ci-dessous :*

```
>>> a = input('Entrez votre âge : ')
Entrez votre âge : 51
>>> 1 + a
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> a = int(input('Entrez votre âge : '))
Entrez votre âge : 51
>>> a+1
52
```

EXERCICE 24 D'après un exercice du site `www.france-ioi.org/`

Écrire un programme Python qui doit lire un entier : un nombre de personnes à sonder. Ensuite, pour chaque personne sondée, il doit lire ses caractéristiques sous la forme de cinq entiers : son âge en années, son nombre d'années d'études après le bac en codant par -1 *une personne qui n'a pas le bac, un entier valant* 0 *si la personne est célibataire ou* 1 *sinon, son nombre d'enfants et un entier valant 1 si la personne a une voiture et 0 sinon.*

On veut déterminer pour chaque personne à quel point elle correspond aux 5 critères suivants :

- *il a au moins 40 ans*
- *il a au plus deux ans d'étude après le bac*
- *il est célibataire*
- *il a des enfants*
- *il a une voiture*

Lorsque cela n'est pas précisé explicitement, les inégalités sont au sens large.

Pour chaque personne, le programme doit tester tous les critères et s'ils sont vérifiés tous les 5 il doit afficher "Client très probable". Si seulement 3 ou 4 sont vérifiés il doit afficher "Client probable". Si aucun n'est vérifié il doit afficher "Client impossible" et dans les autres cas, il doit afficher "Client peu probable".

Soit une suite de variables aléatoires (X_n) indépendantes et de même loi, dont l'espérance est μ.

La *loi faible des grands nombres* assure que pour tout $\varepsilon > 0$ on a :

$$\lim_{n\to+\infty} \mathbb{P}\left(\left|\frac{\sum_{k=1}^{n} X_k}{n} - \mu\right| > \varepsilon\right) = 0$$

Si on réalise n fois la même expérience aléatoire modélisée par une variable aléatoire X_1, la moyenne observée est une réalisation de la variable aléatoire $\frac{\sum_{k=1}^{n} X_k}{n}$. La loi faible des grands nombres nous dit que les valeurs de cette moyenne observée sont probablement proches de l'espérance μ lorsque n est assez grand.

Pour estimer l'espérance μ d'une variable aléatoire , il suffit donc de réaliser un grand nombre de réalisations de celle-ci et de calculer la moyenne des observations.

EXERCICE 25 *On considère une variable aléatoire réelle X de loi de probabilité :*

$$\mathbb{P}(X=-4)=0,2, \quad \mathbb{P}(X=1)=0,7 \text{ et } \mathbb{P}(X=11)=0,1$$

1. *Écrire un programme qui simule une réalisation de la variable aléatoire X. On utilisera la fonction* `random` *du module* `random` *qui retourne un nombre pris au hasard parmi les flottants dans l'intervalle* $[0;1[$.

```
>>> import random
>>> random.random()
0.2395246092276051
```

2. *Modifier le programme précédent pour vérifier expérimentalement la valeur de l'espérance de X à l'aide de la loi faible des grands nombres.*

EXERCICE 26 *Écrire un programme réalisant les opérations suivantes :*

1. *Affiche la phrase : «* `Résolution de l'équation :` $ax^2+bx+c=0$. *»*
2. *Demande d'entrer les valeurs de a, b et c.*
3. *Si* $a=0$ *affiche la phrase : «* `L'équation est du premier degré.` *» puis la résout.*
4. *Si* $a\neq 0$ *affiche la phrase : «* `L'équation est du second degré.` *» puis précise le nombre de solutions, indique si elles sont réelles ou complexes et affiche leur valeur.*

On importera la fonction `sqrt` *du module* `math` *avec* `from math import sqrt`.

EXERCICE 27 *Soit une solution aqueuse d'un acide faible noté AH, de concentration c. La constante d'acidité du couple* $AH/A^{\ominus}$ *est notée* K_a *et* $pK_a=-\log(K_a)$.

1. *Ecrire la réaction de l'acide sur l'eau et dresser un tableau d'avancement. La concentration en ions oxonium à l'équilibre sera notée h.*
2. *Déterminer l'équation de second degré vérifiée par h.*
3. *Adapter le programme de l'exercice 26 : le programme doit désormais demander la constante d'acidité, la concentration c de l'acide en solution et afficher le pH (rappel :* $pH=-\log(h)$*) ainsi que le taux de dissociation de l'acide défini comme le quotient de la concentration de la base à l'équilibre à la concentration initiale de l'acide en solution.*

   NB : vérification du fonctionnement correct du programme : une solution d'acide éthanoïque CH_3COOH*, de concentration c = 0,1* $mol.L^{-1}$ *a un pH d'environ 2,9.*
4. *Modifier une nouvelle fois le programme précédent pour qu'il affiche les deux grandeurs précédentes (pH et taux de dissociation) pour des concentrations variant de 1* $mol.L^{-1}$ *à une concentration minimale* c_{min}*, l'utilisateur ayant entré* $pc_{min}=-\log(c_{min})$*. Le nombre de valeurs de c pour lesquelles les calculs seront effectués sera également à entrer par l'utilisateur.*
5. *Tester le programme précédent pour différentes valeurs de* pK_a *et de concentrations.*
6. *Vérifier que les résultats obtenus sont cohérents avec la loi de dilution d'Ostwald qui peut s'énoncer :*

   « le taux de dissociation d'un acide faible est d'autant plus grand qu'il est dilué. »

7. *Conjecturer la valeur de la limite du taux de dissociation pour un acide infiniment dilué. Le vérifier par un petit calcul (MANUEL) de limite.*

8. *Analyser plus en détails les résultats obtenus à l'aide de ce modèle. Conclure.*

Une suite de caractères entre simples (ou doubles) quote constitue une expression de type chaîne de caractères. On peut y insérer des caractère spéciaux en les échappant avec un \ : saut de ligne \n, tabulation \t, simple quote ', double quote " :

```
>>> chaine
'J\'apprends la programmation \n avec "Python"'
>>> print(chaine)
J'apprends la programmation
 avec "Python"
```

Les caractères d'une chaîne de caractères sont accessibles par l'opérateur crochet, qui permet aussi le *slicing* ou découpage en tranches comme pour les tableaux (ou listes). La fonction `len` retourne la longueur d'une chaîne.

```
>>> ex = 'abcd'
>>> len(ex)
4
>>> ex[0], ex[3], ex[-1], ex[0:2], ex[:2] , ex[0:2:1], ex[-1:-5:-1], ex[::-1]
('a', 'd', 'd', 'ab', 'ab', 'ab', 'dcba', 'dcba')
```

Il existe 2 façons de parcourir une chaîne (idem pour un tableau/liste). La première est plus lourde mais permet de récupérer les indices des caractères :

```
>>> for k in range(len(ex)):
...     print(ex[k], end='')
...
abcd
>>> for j in ex:
...     print(j, end='')
...
abcd
```

Enfin on peut concaténer les chaînes de caractères :

```
>>> ex = ex + 'e'
>>> ex
'abcde'
```

EXERCICE 28 1. *Compléter le programme Python ci-dessous qui prend en entrée une chaîne de caractères, la convertit en minuscules puis qui retourne une nouvelle chaîne où les voyelles ont été triplées.*

```
chaine = input('Entrez une chaine de caractères :\n')
chaine = chaine.lower() #on convertit la chaine en minuscules
voyelles = 'aoiueyéëèäàîù'
newchaine = ''
for lettre in chaine:
    .....
```

2. *Écrire un programme Python qui demande à l'utilisateur une chaîne de caractères et qui retourne la chaîne miroir obtenue en parcourant l'entrée à l'envers.*

3. *Écrire un programme Python qui demande deux chaînes de caractères et qui détermine s'il s'agit de palindromes.*

EXERCICE 29 *Lancer le logiciel Rurple depuis le partage réseau T. Ce logiciel permet de déplacer un robot nommée Rurple sur une scène en utilisant le langage de programmation Python augmenté de quelques fonctions spécifiques.*

Voici les principales fonctions qui permettent de déplacer le robot ou de le faire interagir avec son environnement :

- *« `avance( )` » (ou appui sur touche A) : le robot avance d'une case devant lui.*
- *« `gauche( )` » (ou appui sur touche G) : le robot effectue un quart de tour vers la gauche.*
- *« `depose( )` » (ou appui sur touche D) : le robot dépose une bille sur la case où il se trouve.*
- *« `prends( )` » (ou appui sur touche P) : le robot ramasse une bille sur la case où il se trouve.*

L'interface graphique est composé d'une barre de menu classique en haut et de trois fenêtres :

- *La fenêtre de gauche accueille le programme (ou script) des instructions que l'on souhaite faire exécuter au robot. Celui-ci lit le programme de haut en bas et exécute les instructions de façon séquentielle. Certaines instructions comme les conditions ou les boucles permettent de faire des sauts en avant ou en arrière dans le code source du programme. On peut enregistrer/ouvrir/créer un nouveau un programme/script (au format `*.py` comme un script Python) en sélectionnant le menu Programme. Il faut enregistrer un nouveau programme avant de l'exécuter.*
- *La fenêtre en haut à droite contient la grille d'évolution du robot, on peut régler sa taille à partir du menu Scène et il est possible d'insérer des murs ou des billes dans la grille en cliquant dessus avec le bouton gauche. On peut ouvrir/créer une nouvelle/enregistrer (au format *.wld) une scène depuis le menu Scène. Pour réinitialiser la scène, on peut cliquer sur le bouton correspondant de la barre d'outils ou effectuer la combinaison de touches sur `CTRL + R`.*
- *La fenêtre en bas à droite contient trois volets Affichage/Fonctions/Variable. Dans Affichage seront écrites les messages du robot (c'est le programmeur qui le fait parler . . .). Dans Fonctions se trouvent la liste des fonctions spécifiques à Rurple en plus des fonctions classiques de Python. Dans Variables on pourra observer l'évolution du contenu des variables.*

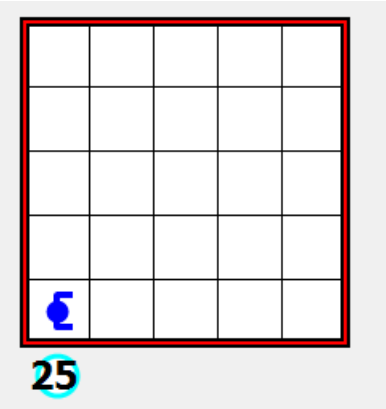
En dessous de la barre de menu se trouve un barre d'outils avec trois icônes cliquables pour contrôler l'exécution d'un programme (lecture puis interprétation par le robot) selon quatre fonctionnalités : Exécuter, Suspendre, Stopper, Exécuter pas à pas (fait une pause après chaque ligne/instruction).

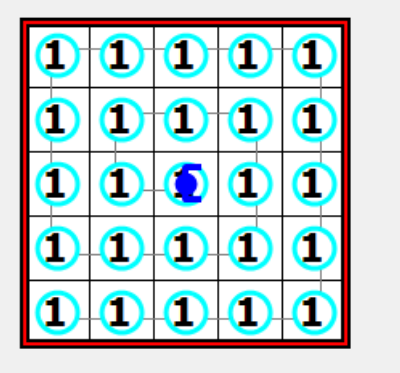
1. *Écrire un programme qui permet au robot de cueillir toutes les billes déposées sur les cases d'une scène rectangulaire de dimensions $n \times 1$ avec éventuellement plus d'une bille par case.*

   On utilisera les fonctions `mur_devant()` qui retourne un booléen indiquant s'il y a un mur devant le robot et `bille_au_sol()` qui retourne un booléen indiquant si la case contient au moins une bille.

2. *Dans le menu scène, sélectionner Nombre de billes et placer quatre billes dans la poche du robot. Écrire un programme qui permet au robot de faire un tour de scène et de déposer une bille à chaque coin de n'importe quelle scène rectangulaire.*
3. *Écrire un programme qui permet à `Rurple` de déposer une bille sur chacune des cases d'une scène de dimensions $n \times n$, où n est un entier impair. Le robot doit parcourir la scène en spirale depuis le coin inférieur gauche (Voir figure 2 ci-dessous).*

   On commencera par placer n^2 billes dans la poche de `Rurple` depuis le menu `Scène/Nombre de billes`.





EXERCICE 30 *Calculer* $\sum_{k=10}^{100} k^2$.

EXERCICE 31 *Soit un entier* $n \geqslant 1$. *Ecrire un programme qui détermine le seul entier* p, *positif ou nul tel que* $2^p \leqslant n < 2^{p+1}$.
Encadrer le nombre d'itérations en fonction de $\ln n$.

EXERCICE 32 *On considère la suite* (u_n) *définie par* $\begin{cases} u_0 = 1 \\ u_{n+1} = 2u_n + n \end{cases}$.
Ecrire un programme Python qui prend en entrée un entier n et qui retourne le terme de rang n de la suite (u_n).

EXERCICE 33 *Ecrire un programme Python qui compte le nombre de décompositions distinctes d'un entier naturel n comme somme de deux carrés d'entiers naturels.*

EXERCICE 34 Project Euler ; problem 1

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.
Find the sum of all the multiples of 3 or 5 below 1000.

EXERCICE 35 Project Euler ; problem 6

The sum of the squares of the first ten natural numbers is, $1^2 + 2^2 + ... + 10^2 = 385$
The square of the sum of the first ten natural numbers is, $(1 + 2 + ... + 10)^2 = 55^2 = 3025$
Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is $3025 - 385 = 2640$.
Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

4 Encore des boucles

EXERCICE 36
1. *Ecrire un programme Python qui permet d'afficher le triangle 1 ci-après.*
2. *Ecrire un programme Python qui permet d'afficher le triangle 2 ci-après.*
3. *Ecrire un programme Python qui permet d'afficher le triangle 3 ci-après.*

Triangle 1

```
0
01
012
0123
01234
012345
0123456
01234567
012345678
0123456789
```

Triangle 2

```
0123456789
 012345678
  01234567
   0123456
    012345
     01234
      0123
       012
        01
         0
```

Triangle 3

```
     *
    ***
   *****
  *******
 *********
***********
```

EXERCICE 37 *Calculer* $\sum_{k=843}^{1789} \cos k$.

Éteindre l'écran, sortir un papier et un crayon (si ce n'était déjà fait) ; calculer cette somme à la main et vérifier.

EXERCICE 38 *Donner une évaluation de la « somme de série »* $\sum_{n=1}^{+\infty} \frac{(-1)^{n+1}}{n}$*, définie comme la limite*[4] *des sommes partielles* $S_N = \sum_{n=1}^{N} \frac{(-1)^{n+1}}{n}$ *lorsque N tend vers* $+\infty$.

La théorie dit que cette somme vaut $\ln 2$, et que la différence entre cette valeur et S_N est « équivalente » à $\frac{(-1)^N}{2N}$.

EXERCICE 39 *Calculer* $\sum_{i+j=1000} (i^2+j)(i+j^2)$ *(la somme porte sur des indices* $i, j \in \mathbb{N}$*).*

EXERCICE 40 *Calculer de trois façons différentes* $\sum_{1 \leqslant i \leqslant j \leqslant 5000} (i-j)^3$*. En regroupant judicieusement les termes, on peut éviter deux boucles imbriquées et réduire de façon spectaculaire le nombre d'instructions et donc le temps d'exécution.*

En informatique, tous les caractères sont codés par des entiers. La correspondance de référence s'appelle *Unicode*. La fonctions `ord` retourne l'ordinal Unicode d'un caractère et `chr` retourne le caractère associé à un ordinal :

```
>>> ord('a')
97
>>> chr(97)
'a'
>>> [chr(k) for k in range(ord('a'), ord('e') + 1)]
['a', 'b', 'c', 'd', 'e']
```

EXERCICE 41 *1. Le programme Python ci-dessous permet d'obtenir la figure 1.*

```
n = -1
while n < 0 or n > 26:
    n = int(input('n = '))
for y in range(n):
    for x in range(n):
        shift = min(x,y)
        if x  < n - 1:
            print(chr(ord('a') + shift), end='')
        else:
            print(chr(ord('a') + shift), end='\n')
```

4. Dont il convient de prouver qu'elle existe ; attendre quelques mois pour ça !

Modifier le programme pour obtenir la figure 2 puis la figure 3.

2. *Écrire un programme qui affiche la figure 4.*

```
#Figure 1

aaaa
abbb
abcc
abcd
```

```
#Figure 2

aaaaaaa
abbbbba
abcccba
abcdcba
abcccba
abbbbba
aaaaaaa
```

```
#Figure 3

ddddddd
dcccccd
dcbbbcd
dcbabcd
dcbbbcd
dcccccd
ddddddd
```

```
#Figure 4

abcdcba
bbcdcbb
cccdccc
ddddddd
cccdccc
bbcdcbb
abcdcba
```

EXERCICE 42 Projet Euler n° 63

The 5-digit number, $16807 = 7^5$*, is also a fifth power. Similarly, the 9-digit number,* $134217728 = 8^9$*, is a ninth power. How many n-digit positive integers exist which are also an nth power?*

5 Besoin d'indications ?

- Exercice 15. Stocker d'abord la somme de toutes les valeurs dans l'une des variables. Les flottants sont des représentations approchées des réels.
- Exercice 30. Il n'y a pas grand chose à modifier par rapport à l'exercice 20! Notons qu'il existe une façon efficace de trouver le résultat :

```
>>> sum(k**2 for k in range(10,101))
338065
```

- Exercice 34. Il suffit de boucler sur tous les entiers majorés par 1000 et, pour chacun, tester la divisibilité par 3 et 5. On trouvera : 233168. By the way, «below» signifie quoi précisément ?
- Exercice 41. Pour les figures 2, 3 et 4, il faut utiliser les symétries de la figure.

  La fonction `max` peut être utile, par exemple :

```
>>> chr(ord('a')+max(0,1))
'b'
```