Diplôme inter-universitaire “Enseigner l’Informatique au Lycée” (DIU-EIL)

Polycopié d’exercices (Bloc 1)

Table des matières

Résumé

Ce cahier Ce cahier d'exercices a été réalisé pour l'enseignement "Bloc1" du DIU "Enseigner l'Informatique au Lycée", proposé par l'université Lyon1 en collaboration avec d'autres partenaires (Lyon2, ENSL, ENSSIB ...)

Il peut être librement distribué (License CC-BY-SA 4.0).

Intervenants Les personnes suivantes ont été impliquées dans l'écriture de ces exercices et des supports Python associés, et l'enseignement correspondant au sein du Bloc 1 :

— 2018/2019 : Pascal Busac, Laure Gonnord, Olivier Laurent, Serge Miguet, Matthieu Moy, Nicolas Pronost, Frédéric Suter.

— 2019/2020 : Nicolas Louvet.

`https://diu-eil.univ-lyon1.fr/bloc1/index.html`

Contenu Algorithmique, programmation, architecture du bloc 1

1. Représentation de l'information
   (a) Codage des nombres flottants.
   (b) Fichiers et formats usuels, compression et archivage
2. Langages et programmation
   (a) Types structurés, p-uplets, tableaux et dictionnaires
   (b) Traitement de données en tables (recherche, tris, fusion)
   (c) Modularité, bibliothèques
   (d) Diversité des langages de programmation
   (e) Spécification, prototypage et tests

TD 1
Prise en main

1.1 Échauffement

EXERCICE #1 ▸ **Prise en main**
Ouvrez votre environnement Python (Spyder, Pyzo, ...), sauvez votre fichier courant dans un endroit pertinent, avec un nom pertinent.

EXERCICE #2 ▸ **Utilisation de Python comme calculatrice**
Dans la partie "invite de commande" de votre environnement Python, faire des petits calculs :

```
>>> 2 + 2
4
```

En particulier ceux des transparents "introduction et prérequis", par exemple :

- `1.0 + 1`
- `"Bonjour " + "a tous"`
- `"Bonjour" + 10`
- `2 == 2`
- `2 == 3`
- `(1 == 2) == (3 == 4)`

EXERCICE #3 ▸ **Fichier Python et interprète ligne à ligne.**
Maintenant dans la partie "éditeur" (donc dans votre *fichier source Python* (ou *fichier de script*, mais je n'aime pas trop cette dénomination ; ou *fichier Python*), réécrire les mêmes expressions de calcul qu'à l'exercice précédent, et trouvez les menus de votre éditeur qui permettent d'évaluer ligne à ligne comme précédemment. *Remarquez que dans ce mode l'évaluateur python imprime les valeurs que vous calculez, même si vous ne lui dites pas explicitement.*.

EXERCICE #4 ▸ **Récupération d'un fichier source extérieur**
Récupérez le fichier `demo1.py` sur la page du DIU. Sauvez-le au bon endroit **NE PAS cliquer sur le lien, mais clic droit "enregistrez-sous"**, puis ouvrez-le dans votre environnement Python. Exécutez les premières lignes comme précédemment.

EXERCICE #5 ▸ **Interprétation du fichier complet**
Dans votre environnement python, exécutez le programme complet [1]. Remarquez que seules les sorties écran sont visibles.

EXERCICE #6 ▸ **Valeur de retour de fonction versus impression**
Toujours dans le même fichier, on a écrit deux versions "presque identiques" d'une fonction, qu'on vous laisse analyser. *Il est important de bien séparer le calcul des entrées/sorties. Ce sont des activités différentes.*

1.2 Premiers tours de piste

Les exercices de cette section seront faits dans un nouveau fichier Python, avec un nom pertinent, dans un endroit pertinent. *Il est important de bien vérifier que vos élèves savent faire cela, régulièrement.*

1. sous Spyder, par exemple, voici une démo `https://matthieu-moy.fr/cours/liesse/spyder/editeur/`

EXERCICE #7 ▸ **If**
Écrivez un programme qui initialise une variable entière à une valeur donnée par l'utilisateur (utilisez la fonction `input`) et imprime un message différent selon sa parité.

EXERCICE #8 ▸ **"Même exo"?**
Écrire une fonction :

```
def est_pair(x)
"""retourne/renvoie Vrai si x:entier est pair
"""
```

et réalisez les entrées/sorties dans le reste du programme pour qu'il ait le même comportement que le précédent.

À partir de maintenant, mettez une documentation minimale au début de vos fonctions, comme dans l'exercice précédent.

EXERCICE #9 ▸ **While**
Sur le même modèle que précédemment (séparation calcul/entrées-sorties), érire un programme qui demande deux entiers à l'utilisateur, calcule et imprime leur pgcd.

EXERCICE #10 ▸ **Factorielle récursive**
Même question, avec une implémentation récursive de la factorielle.

À partir de maintenant les programmes seront systématiquement testés, mais il est inutile de faire des entrées sorties systématiquement, vous pouvez directement initialiser des variables dans votre programme python...

EXERCICE #11 ▸ **For/range**
En utilisant les constructions `for i in range(...)`, écrire un programme qui calcule la somme de tous les i pairs de 2 à 42 inclus.

TD 2

Structures de données 1 : listes/tables

2.1 Premiers pas : listes simples

EXERCICE #1 ▶ **Listes**
Écrire un programme qui déclare et initialise une liste (d'entiers), et calcule la somme de ses éléments. *Remarquer qu'il n'est pas simple de distinguer* $i, l, 1$ *et conclure sur le fait que l'utilisation de* `l` *comme identifiant de liste (ou whatever else) est à bannir.*

EXERCICE #2 ▶ **Itérer sur des listes**
En supposant que `mylist` soit une liste d'entiers, que font les instructions suivantes ? *Essayer d'abord de répondre sans ordi.*

```
[k ** 2 for k in mylist]
```

et

```
[k for k in mylist if k % 2 == 0]
```

EXERCICE #3 ▶ **Initialisation de listes**
Il peut être utile de savoir initialiser des listes. En vous inspirant des fonctions précédentes, déclarer et initialiser des listes :

— entiers pairs de 2 à 42 compris
— entiers pris au hasard entre 0 et 1000, en nombre n.

Dans le programme de la formation, il y a les tuples "p-uplets". Il n'y a pas grand chose à dire dessus (à part qu'une fois créé, un tuple n'est pas modifiable). Vous pouvez par exemple lire la documentation ici : `https://docs.python.org/fr/3/tutorial/datastructures.html#tuples-and-sequences`.

EXERCICE #4 ▶ **Tuples et listes**
Écrire une fonction Python qui prend une liste d'entiers en paramètre et retourne un couple (2-uplet) de listes, la première étant la sous-liste des entiers pairs, et l'autre la sous-liste des entiers impairs.

Dans la suite on va coder des algorithmes de base sur "les tableaux". En Python (ce n'est pas le cas d'autres langages de programmation), ces objets seront en fait les listes que nous venons de voir [1]. Les règles du jeu sont que les "données en table" soient toutes du même type (entier chez nous), et que l'on accède à ces données uniquement avec leur indice (`mylist[42]`). Les variables listes seront alors un poil abusivement nommées `tab` :-)

EXERCICE #5 ▶ **Listes, etc.**
Pour approfondir vous trouverez plein d'exercices intéressants dans ce polycopié `http://www.prepamantes.fr/wp-content/uploads/2013/08/Bases_algo+python.pdf` (pages 41 et suivantes).

2.2 Recherches

EXERCICE #6 ▶ **Recherche dans un tableau**
Écrire une fonction de recherche d'un élément dans un tableau.

EXERCICE #7 ▶ **Recherche dans un tableau trié**
Écrire une fonction de recherche dichotomique dans un tableau trié. On fera une version récursive et une version itérative.

1. Il y a bien d'autres utilisations des listes, par exemple allez voir à l'adresse `https://docs.python.org/fr/3/tutorial/datastructures.html`

2.3 Tris

Test si liste triée

Avant de commencer à écrire des algorithmes de tri, on s'intéresse à tester si une liste est triée (pas besoin de la triée si elle l'est déjà). Programmer une fonction Python qui teste si une liste passée en paramètre est triée.

Tri par insertion

Le tri par insertion est l'algorithme utilisé par la plupart des joueurs lorsqu'ils trient leur « main » de cartes à jouer. Le principe consiste à prendre le premier élément du sous-tableau non trié et à l'insérer à sa place dans la partie triée du tableau.

a. Dérouler le tri par insertion du tableau [5.1, 2.4, 4.9, 6.8, 1.1, 3.0].

b. Ecrire en Python la procédure de tri par insertion, par ordre croissant, d'un tableau de réels :

Procédure tri_par_insertion (tab : tableau de n réels) **Précondition** : tab[0], tab[1], ... tab[n-1] initialisés **Postcondition** : tab[0] ≤ tab[1] ≤ ... ≤ tab[n-1]

c. Un algorithme de tri est dit « stable » s'il préserve toujours l'ordre initial des ex-aequos. Dans notre exemple, l'algorithme est stable si des valeurs identiques restent dans leur ordre d'apparition avant le tri. L'algorithme de tri par insertion est-il stable ?

Tri par sélection

Le principe du tri par sélection est le suivant. Pour chaque élément i de 1 à n-1, on échange tab[i] avec l'élément minimum de tab[i..n]. Nous devons donc rechercher, pour chaque itération le minium d'un sous-tableau de plus en plus petit. Les éléments à gauche de i sont à leur place définitive et donc le tableau est complémentent trié lorsque i arrive sur l'avant dernier élément (le dernier élément étant forcément le plus grand

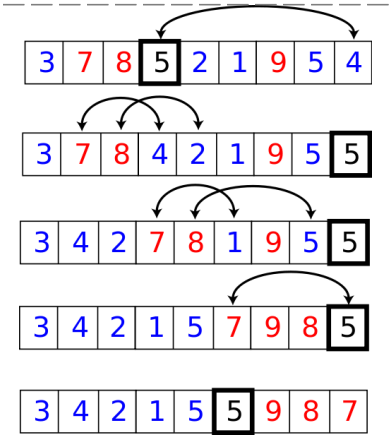
Ecrire en Python la procédure de tri par sélection, par ordre croissant, d'un tableau de réels :

Tri rapide *(quick sort)*

Cette méthode de tri consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le *partitionnement*. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Concrètement, pour partitionner un sous-tableau :

- on place le pivot arbitrairement à la fin (peut être fait aléatoirement), en l'échangeant avec le dernier élément du sous-tableau (étape 1 ci-contre)
- on place tous les éléments inférieurs au pivot en début du sous-tableau (étapes 2 et 3)
- on place le pivot à la fin des éléments déplacés (étapes 4 et 5)



Écrire en Python la procédure de partitionnement et la procédure récurvise de tri rapide.

Tri par fusion interne

Explication Wikipedia-fr :

*À partir de deux listes triées, on peut facilement construire une liste triée comportant les éléments issus de ces deux listes (leur *fusion*). Le principe de l'algorithme de tri fusion repose sur cette observation : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste (en fait, le plus petit des deux, à supposer qu'aucune des deux listes ne soit vide, sinon la réponse est immédiate).*

Ce procédé est appelé fusion et est au cœur de l'algorithme de tri suivant :

•Si le tableau n'a qu'un élément, il est déjà trié.
•Sinon, séparer le tableau en deux parties à peu près égales.
•Trier récursivement les deux parties avec l'algorithme du tri fusion.
•Fusionner les deux tableaux triés en un seul tableau trié.

Écrire en Python une version récursive de l'algorithme du tri par fusion d'un tableau d'entiers.

TD 3

Structures de données Python 2 : dictionnaires

EXERCICE #1 ▸ **Archive zip**
Récupérer sur la page web du bloc l'archive zip contenant tous les fichiers nécessaires à ce TD. Enregistrer au bon endroit, puis dés-archiver correctement.

Un dictionnaire est une collection non-ordonnée, éditable et indexée. L'indexation se fait par une clé qui peut être de n'importe quel type immuable comme des chaînes de caractères ou des nombres. Une liste ne peut donc pas être une clé. À une clé est associée une valeur. Nous pouvons donc voir un dictionnaire comme un ensemble non-ordonné de paires (clé,valeur). Les clés comme les valeurs d'un même dictionnaire peuvent être de différents types.

Pour créer un dictionnaire, les accolades sont utilisées. `{}` est donc un dictionnaire vide. Pour ajouter des paires (clé,valeur), nous les espaçons par des virgules avec des deux points entre la clé et la valeur.

Par exemple :

```
dictionnaire = {cle1 : valeur1 , cle2 : valeur2 , ...  clen : valeurn}.
```

Les opérations principales sur un dictionnaire sont l'ajout d'une paire (clé,valeur) et l'accès à une valeur par sa clé. La suppression d'une paire d'un dictionnaire est évidement aussi possible. Dans un dictionnaire, il n'y a pas deux clés identiques (l'essai de l'ajout d'une clé déjà existante va changer la valeur associée à la clé et non ajouter une deuxième occurrence de la clé). Par contre deux clés différentes peuvent être associées à la même valeur.

Les opérations classiques sur les containers comme len, min/max, enumerate, in etc. sont disponibles sur les dictionnaires et sont effectuées sur les clés. En plus de ces opérations classiques, des opérations spécifiques aux dictionnaires suivantes sont disponibles telles que :

Opération	**Exemple (d est le dictionnaire)**
Accès à une valeur par la clé	`valeur = d[cle]`
Modification d'une valeur par la clé	`d[cle] = valeur`
Vider le dictionnaire	`d.clear()`
Suppression d'une paire (clé,valeur)	`del d[cle]`
Récupération des clés (liste)	`d.keys()`
Récupération des valeurs (liste)	`d.values()`
Récupération des paires (liste de tuples)	`d.items()`

FIGURE 3.1 – Opérations simples sur les dictionnaires

3.1 Échauffement

EXERCICE #2 ▸ **Fonctions de base dictionnaires**

1. Donner les instructions permettant de créer un dictionnaire associant des prénoms et des âges.
2. Donner les instructions permettant d'itérer sur le dictionnaire et d'afficher les paires sous le format 'prenom a age ans.'

On peut s'apercevoir qu'un dictionnaire n'est pas très différent d'une liste de tuples (`d.items()` ressemble beaucoup à d lui-même). Alors pourquoi utiliser des dictionnaires? La réponse est dans la performance des opérations. Accéder et modifier une paire est beaucoup plus rapide dans un dictionnaire que dans une liste

car dans une liste on doit la parcourir pour trouver l'élément alors que dans un dictionnaire on y accède directement.

Ceci se fait grâce à une table de hachage. Une fonction de hachage de cette table sert à mettre en correspondance une clé et l'indice de sa valeur dans la liste stockant les valeurs. En réalité la fonction donne une valeur de hachage qui donne l'indice. La fonction de hachage dépend du type de la clé, mais dans tous les cas, elle donne un identificateur (valeur de hachage) idéalement unique pour la clé. Ensuite l'identificateur est converti en indice dans la liste des valeurs du dictionnaire.

Mais alors pourquoi ne pas toujours utiliser des dictionnaires au lieu des listes si c'est plus rapide? C'est parce que la table de hachage prend de la place en mémoire. On choisit donc entre dictionnaire et liste en fonction du compromis souhaité entre performance et espace mémoire.

EXERCICE #3 ▶ **Conversion Liste Dictionnaire**
Écrire une fonction `dictToList` qui convertit un dictionnaire en une liste de tuples (sans utiliser la fonction `items()`) et une fonction `listToDict` qui convertit une liste de tuples en dictionnaire.

3.2 Comparaison Liste/Dictionnaire

EXERCICE #4 ▶ **Charger une liste à partir d'un fichier**
Dans l'archive, on fournit `diu-dico-english.py` ainsi que le dictionnaire `dico.en2fr`. Ouvrir le fichier Python et expérimenter. *Il n'est pas nécessaire de comprendre plus que cela comment est réalisé le chargement à partir du fichier. Un TD y sera consacré.*.

EXERCICE #5 ▶ **Charger un dictionnaire à partir d'un fichier**
Remplir les trous de l'exemple précédent avec des fonctions utilisant un dictionnaire [1]. Comparez les temps d'exécution.

EXERCICE #6 ▶ **Utilisation de dictionnaires en classe**
Imaginer des scénarios d'usage pour des élèves, quel dictionnaire pour quel usage? Construire un tel exercice.

3.3 Pour aller plus loin : implémenter les dictionnaires (tables de hachage)

Cette partie nécessitera l'usage de la librairie matplotlib qu'il faudra donc peut-être installer. On va dans cette section implémenter des tables de hachage, en utilisant une librairie fournie de listes triées. C'est une variante de cette structure de données qui est fournie par Python sous le nom de dictionnaire.

Une table de hachage est un compromis entre les listes chaînées efficaces sur les ajouts et les tableaux (ou listes contiguës) efficaces sur les accès. Soit un ensemble fini $\mathscr{D}$ de données (ici les mots du dictionnaire) que l'on va stocker dans un tableau `ht`. Pour ranger un élement `x` de $\mathscr{D}$, on calcule son image par une fonction de hachage `hash`, qui donne son indice (appelé "indice de hachage") dans `ht`, `x` sera donc rangé dans `ht[hash(x)]`. Il reste à trouver une fonction `hash` adéquate :

— l'idéal est de trouver une fonction bijective entre $\mathscr{D}$ et l'intervalle d'indices du tableau, ainsi chaque case du tableau contient 1 élément de $\mathscr{D}$. Mais il est difficile de trouver une telle fonction bijective, ne serait-ce que parce que $\mathscr{D}$ est rarement connu a priori.

— l'utilisation d'une fonction injective (ie qui associe à chaque `x` de $\mathscr{D}$ une case différente de $\mathscr{D}$) mène à un tableau de taille $sup\{hash(x), x \in \mathscr{D}\}$, c'est-à-dire à un tableau à trous, ce qui peut générer une perte importante de place.

— le principe consiste alors à prendre une fonction surjective, obtenue généralement par modulo sur une taille limitée de tableau, soit `TABLE_SIZE` $\ll card(\mathscr{D})$. On tombe alors sur un problème de "collisions" parce que plusieurs données peuvent avoir le même indice de hachage. On choisit donc une structure mixte formée d'un tableau statique de listes chaînées contenant les éléments de même indice de hachage (appelées "listes de collisions"). Si possible ces listes sont ordonnées pour optimiser les accès. L'ajout d'un élément `x` revient alors à calculer `hash(x)` de coût quasi constant et insérer `x` dans la liste `ht[hash(x)]`, en $o(n)$ n étant la taille de la liste correspondante. Toute la difficulté est de trouver un

1. Nous avons généré ce fichier "élève" à partir de notre source de correction en utilisant le script `generate_skeletons.py` disponible à l'adresse `https://gitlab.com/moy/generate-skeletons`. C'est extrêmement pratique, essayez!

bon compromis entre la taille du tableau et la taille des listes. Remarquez que prendre `TABLE_SIZE=1` revient à faire une simple liste.

EXERCICE #7 ▸ **Coder une table de hachage**
Les fichiers nécessaires `diu-dico-hash.py, libdiulistechainee.py dico.english` sont dans l'archive. Ouvrir le fichier Python `diu-dico-hash`, l'objectif est de remplir les `TODO`.

EXERCICE #8 ▸ **Librairie de listes chaînées**
Ouvrir la librairie, pour voir.

Exemple fourni : dictionnaire anglais Le dictionnaire anglais fourni, avec la fonction de hachage qui calcule la somme des codes ASCII des caractères du mot modulo `TABLE_SIZE` donne la hachtable suivante si `TABLE_SIZE=50` :

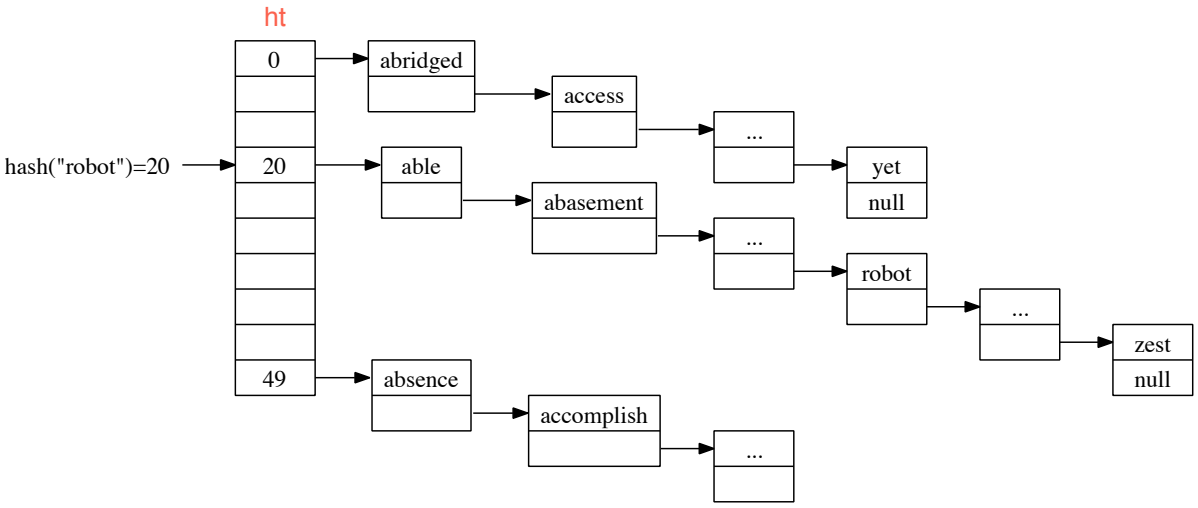


FIGURE 3.2 – Table `ht` du dictionnaire anglais

TD 4

Vérification fonctionnelle : spécifications, tests

4.1 Analyse de programme - correction et invariants

Se rapporter aux transparents du cours sur la page web du DIU. Cette partie sera refaite dans d'autres blocs, il s'agit d'une première mise en situation. Ces exercices seront principalement faits sur papier.

EXERCICE #1 ▸ **Recherche d'un élément dans un tableau**
Prouver la correction de la recherche d'un élément dans un tableau, sans rupture prématurée du flot; puis avec.

EXERCICE #2 ▸ **Recherche dans un tableau trié**
Réécrire la précondition de cet algorithme, et ce qu'on veut prouver. Trouver le bon invariant et le prouver.

EXERCICE #3 ▸ **Fonction de McCarthy**
On va étudier le programme suivant :

```
def mc(n):
    if n > 100:
        return n - 10
    else:
        return mc(mc(n + 11))
```

On cherche notamment à déterminer la fonction calculée par ce programme.

1. Reproduire le code et expérimenter en l'exécutant sur différentes valeurs de $n \in \mathbb{N}$.
2. Modifier le code pour faire apparaître les différents appels récursifs du programme lors de l'exécution.

Les questions suivantes se font principalement sur papier.

3. Déterminer la valeur de la fonction calculée par mc pour $n > 100$.
4. Montrer que si $90 \leq n \leq 100$, $\text{mc}(n) = \text{mc}(n+1)$. En déduire $\text{mc}(n)$ pour $90 \leq n \leq 100$.
5. Montrer que l'intervalle $[90, 111]$ est un invariant de la fonction, *i.e.* :

$$90 \leq n \leq 111 \implies 90 \leq \text{mc}(n) \leq 111$$

6. Montrer que, pour tout $n < 90$, $\text{mc}(n) \in [90, 111]$.
7. Déterminer la valeur de la fonction pour $n < 90$.
8. En conclure quelle est la fonction calculée par le programme mc pour $n \in \mathbb{N}$.

4.2 Tests unitaires - TP

EXERCICE #4 ▸ **Démo**
Récupérer l'archive sur la page web du cours. Nous proposons ici plusieurs manières de tester des algorithmes de tri (ici `tri_insertion`), en "testant" si la sortie de l'algorithme de tri est bien triée [1] :

— Dans un premier temps, uniquement avec Python : `demo_test0.py`, `demo_test1.py`.

— Puis `pytest`, librairie de test, lancée par Python : `demo_test2`, `demo_test3.py` [2].

1. Il faudrait aussi vérifier que les éléments du tableau d'entrée sont tous là, ceci est laissé en exercice au lecteur.
2. On peut aussi lancer le *logiciel pytest* à la ligne de commande, mais c'est un peu sportif suivant l'IDE Python que vous utilisez, donc on oublie pour l'instant

Expérimentez! Un peu d'explications : `pytest` lance chacune des fonctions préfixées par `test_`.

```
@pytest.mark.parametrize('execution_number', range(5))
def test_one_random(execution_number, random_array):
    tri_insertion(random_array)
    assert est_bien_trie(random_array)
```

Le assert est ce qui va déterminer si le test est ok ou non. Remarquez comment la fonction `random_array` est appelée pour générer un tableau aléatoire, et comment on peut réaliser un nombre paramétré de tests.

EXERCICE #5 ▸ **Testez vos tris**
Maintenant testez vos implémentations des tris du TD précédent, avec une des infrastructures fournies. Trouvez-vous des bugs?

EXERCICE #6 ▸ **Tests unitaires pour des exercices d'élèves**
Récupérer une liste des premiers éléments de la suite factorielle par exemple sur le site ici :`https://oeis.org/search?q=factorial&sort=&language=&go=Search`

Faire un fichier de tests qui pourra tester du code élève fournissant la factorielle. Réfléchir sur les contraintes à donner pour que "tout se passe bien".

Pour aller plus loin :

`https://code-maven.com/slides/python-programming/testing-with-pytest`.

4.3 Programmation défensive

EXERCICE #7 ▸ **Moyenne**
Écrire une fonction qui calcule la moyenne de deux entiers positifs, de la manière la plus défensive possible. Utiliser `pytest` pour comparer aux fonctions de librairie min et max.

Cet exercice a été délicatement copié du site `https://swcarpentry.github.io/python-novice-inflammation/08-defensive/index.html`. Je vous conseille de faire l'ensemble de ces exercices, ils sont tous très pertinents.

TD 5

Représentation des données en machine

Quelques trucs utiles

En Python, `chr(i)` renvoie la chaîne représentant un caractère dont le code de caractère Unicode est précisément `i` ; réciproquement, `ord(c)` renvoie le code Unicode du caractère représenté par la chaîne donnée. Notez aussi que le constructeur `int()` permet déjà d'obtenir un entier à partir d'une chaîne de caractère :

```
>>> int('314')
314
>>> int('1010', 2)
10
>>> int('1a', 16)
26
```

Pour convertir un entier en une chaîne de caractères, vous pouvez aussi utiliser les fonctions `bin()` et `hex()`, ainsi bien-entendu que la méthode `format()` de la classe `str` :

```
>>> bin(456)
'0b111001000'
>>> hex(458)
'0x1ca'
>>> "{:x}".format(458)
'1ca'
```

5.1 Binaire, hexadécimal, complément à deux

On commence par des exercices classiques de décomposition en base, vous pouvez sauter si vous êtes à l'aise.
2020 : ajout de quelques indications et exercices, par N. Louvet

EXERCICE #1 ▶ **Fonctions standard Python**
Python fournit les fonctions standards bin(n) et hex(n) qui réalisent les conversions décimal vers binaire, et hexadécimal vers binaire. Vous pouvez expérimenter avec le fichier `demo_repnombres.py` (archive sur la page du DIU).

EXERCICE #2 ▶ **Conversions en base b**

On vous fournit un fichier `change_repr.py`. Remplissez les deux fonctions à trous :

— `lire_repr(rep, b)` interprète la chaîne de caractères *rep* en base *b* et retourne le nombre décimal associé. On utilisera la fonction auxiliaire `nombre`.

— `repr_nombre(n, b)` réalise la conversion inverse. On utilisera la fonction auxiliaire `chiffre`.

Tester, etc.

EXERCICE #3 ▶ **Calcul avec l'algorithme de Horner**

Le principe de la conversion par la méthode de Horner de l'entier naturel $n = (x_3 x_2 x_1 x_0)_\beta$ est le suivant. On a :

$$n = x_3 \cdot \beta^3 + x_2 \cdot \beta^2 + x_1 \cdot \beta^1 + x_0, = (x_3 \cdot \beta^2 + x_2 \cdot \beta^1 + x_1) \cdot \beta + x_0, = ((x_3 \cdot \beta + x_2) \cdot \beta + x_1) \cdot \beta + x_0.$$

Si on exécute l'algorithme suivant,

$r_3 = x_3$
$r_2 = r_3 \times \beta + x_2$
$r_1 = r_2 \times \beta + x_1$
$r_0 = r_1 \times \beta + x_0$

alors par construction on a l'égalité $r_0 = x_3 \cdot \beta^3 + x_2 \cdot \beta^2 + x_1 \cdot \beta^1 + x_0$. Si on effectue tous les calculs dans une base d'arrivée γ, on obtient l'écriture de n en base γ. En outre, on comprend bien que la méthode de Horner n'est pas limitée à des nombres de 4 chiffres en base β, et qu'elle peut se généraliser facilement.

1. Testez l'algorithme de Horner pour convertir l'entier $n = (11010101)_2$ en décimal.
2. Plus fastidieux à faire à la main : testez l'algorithme de Horner pour convertir l'entier $n = (567)_{10}$ en binaire. Combien de multiplications avez vous posées? Combien en auriez-vous posées si vous aviez utilisé la définition de la notation positionnelle pour faire cette conversion?
3. Ecrivez une fonction en Python pour convertir une chaîne de caractères représentant un entier positif en décimal, en un entier non-signé en machine, en utilisant l'algorithme de Horner. Si k est le nombre de chiffres dans la chaîne d'entrée, combien de multiplications sont utilisées? Que penser de cette méthode?

EXERCICE #4 ▶ **Et encore**

1. Donnez en Python une fonction qui permet d'obtenir, sous la forme d'une chaîne de caractères, le codage en complément à 2 d'un entier sur k bits.
2. Donnez en Python une fonction qui permet de convertir en un entier une chaîne de longueur k formée de caractères `'0'` ou `'1'`, en l'interprétant comme le codage en complément à deux d'un entier sur k bits.

5.2 Virgule flottante

EXERCICE #5 ▶ **Représentation en machine, norme IEEE-754 : TD**

La norme IEEE-754 définit la manière dont sont représentés les flottants sur les ordinateurs actuels. On considère le code suivant :

$c =$	s	code de l'exposant : c_e	code de la mantisse : c_m
	1 bit	k bits	$p-1$ bits

Pour un flottant normal, la valeur codée est $f(c) = (-1)^{s(c)} \times m(c) \times 2^{e(c)}$, avec

— $s(c) = s$ le signe du flottant.

— $m(c) = (1, c_m)_2$, ce qui signifie que le 1 de tête est codé implicitement.

— $e(c)$ dépend de $(c_e)_2$, mais doit être interprété avec moultes précautions...

Comme c_e est un code sur k bits, notons que $0 \leq (c_e)_2 \leq 2^k - 1$.

— $(c_e)_2 = 0$ et $(c_m)_2 = 0$ indique que le nombre représenté est 0. Il y a deux codages de 0 : -0 ou $+0$ selon s.

— $(c_e)_2 = 2^k - 1$ indique une valeur exceptionnelle (+Inf, -Inf, NaN) : $1/0 = +\text{Inf}$, $0/0 = \text{NaN}$, $a + \text{Inf} = \text{Inf}$, ...

— $1 \leq (c_e)_2 \leq 2^k - 2$ indique que le nombre représenté est normal. Alors,

$$m(c) = (1, c_m)_2 \quad \text{et} \quad e(c) = (c_e)_2 - \underbrace{(2^{k-1} - 1)}_{\text{biais}}.$$

Les types float et double du C correspondent à deux formats prévus par la norme IEEE-754, appelés *simple précision* et la *double précision* :

format	mantisse		exposant				taille
	précision	$p-1$	k	biais	$e_{\min}$	$e_{\max}$	
simple	$p = 24$	23	8	127	-126	127	32 bits
double	$p = 53$	52	11	1023	-1022	1023	64 bits

— Donnez les valeurs extrêmes λ et σ pour la simple et la double précision.

— Donnez le codage de $(1,5)_{10}$ en simple précision, sous la forme d'un code en hexadécimal.

EXERCICE #6 ▸ **Arrondi correct**

À savoir ! Pour représenter un réel r par un nombre flottant, il faut en général arrondir r. La norme IEEE-754 propose 4 modes d'arrondi, mais on n'utilise que l'*arrondi au plus proche* (RN) dans ce TD : RN(r) est le flottant le plus proche de r, et si r est équidistant de deux flottants consécutifs alors RN(r) est celui dont la mantisse se termine par un 0. La norme impose *l'arrondi correct* pour les opérations +, -, ×, / et $\sqrt{\cdot}$: le résultat calculé doit être le résultat exact arrondi selon le mode d'arrondi courant.

— Donnez la représentation positionnelle en base 2 de $(0,1)_{10}$.

— Est-il possible de représenter exactement $(0,1)_{10}$ par un flottant binaire ?

— Représentez $(0,1)_{10}$ par un flottant sur 8 bits de précision. S'il y a lieu, effectuez un arrondi au plus proche.

EXERCICE #7 ▸ **À lire à la maison**
Le texte de la documentation `https://docs.python.org/fr/3.6/tutorial/floatingpoint.html` est très intéressant. Faire les petites manipulations.

5.3 Ascii

EXERCICE #8 ▸ **Ascii "à la main"**
exercice du à N. Louvet

1. On considère les informations ci-dessous, obtenues en tapant des commandes dans un shell Linux :

```
nlouvet@ninjutsu:~/$ cat toto.txt
Toto a fait du vélo tout l'été !!!
nlouvet@ninjutsu:~/$ file toto.txt
toto.txt: UTF-8 Unicode text
nlouvet@ninjutsu:~/docsvn/diu-lyon/nicolas/td-num$ hexdump -C toto.txt
00000000  54 6f 74 6f 20 61 20 66  61 69 74 20 64 75 20 76  |Toto a fait du v|
00000010  c3 a9 6c 6f 20 74 6f 75  74 20 6c 27 c3 a9 74 c3  |..lo tout l'..t.|
00000020  a9 20 21 21 21 0a                                 |. !!!.|
00000026
nlouvet@ninjutsu:~/$
```

   En vous basant sur votre culture générale, votre bon sens, et sans consulter de table des codes ASCII :

   — Quel est le code ASCII de `'a'` ?

   — Quel est le code ASCII de `'A'` ?

   — Quel est le code UTF-8 de `'é'` ?

   — Quel est le codage de votre prénom en code ASCII ?

2. Le fichier `song.txt` contient le (1er) refrain d'une chanson célèbre d'un auteur-compositeur-interprète, musicien, peintre et poète américain (toujours en vie en 2019) : retrouvez le titre de cette chanson !

```
nlouvet@ninjutsu:~/$ file song.txt
song.txt: ASCII text
nlouvet@ninjutsu:~/$ hexdump -v -e '8/1 "%02X " "\n"' song.txt
48 6F 77 20 64 6F 65 73
20 69 74 20 66 65 65 6C
2C 20 68 6F 77 20 64 6F
65 73 20 69 74 20 66 65
65 6C 3F 0A 54 6F 20 62
65 20 6F 6E 20 79 6F 75
72 20 6F 77 6E 2C 20 77
69 74 68 20 6E 6F 20 64
69 72 65 63 74 69 6F 6E
20 68 6F 6D 65 0A 41 20
63 6F 6D 70 6C 65 74 65
20 75 6E 6B 6E 6F 77 6E
2C 20 6C 69 6B 65 20 61
20 72 6F 6C 6C 69 6E 67
20 73 74 6F 6E 65 0A
nlouvet@ninjutsu:~/$
```

TD 6
Fichiers

Adapté de Matthieu Moy, Benjamin Wack, . . . , stage Liesse Grenoble

6.1 Fichiers, fichiers textes

Pour organiser des données sur un disque dur, on utilise généralement des fichiers et des répertoires (parfois appelés « dossiers »).

— Les fichiers (« file » en anglais) contiennent l'information à proprement parler.

— Les répertoires (« directory » en anglais) contiennent des fichiers ou d'autres répertoires.

Un fichier est une suite d'octets (1 octet = 8 bits = 1 nombre de 0 à 255). Par exemple, pour un fichier contenant un programme python, et plus généralement pour tous les fichiers textes, chaque octet correspond à un caractère du programme. Chaque octet est un nombre, et il existe une correspondance entre ces nombres et les caractères : le code ASCII [1]. Par exemple, le code ASCII dit que le nombre 97 est associé au caractère a, le nombre 98 au caractère b, etc.

Pour d'autres formats de fichier plus évolués (texte mis en forme par LibreOffice ou Word, image JPEG ou PNG...), la suite de caractères n'a pas forcément de sens pour un être humain, mais les logiciels appropriés savent les lire.

EXERCICE #1 ▶ **Prise en main**

Téléchargez et extrayez l'archive ZIP disponible sur le site du cours. Ouvrez le fichier `display_file.py` qui s'y trouve. Ce fichier contient une fonction display_file(name) qui permet d'afficher les 10 premiers caractères d'un fichier.

Vous n'avez pas besoin de comprendre les détails, mais vous pouvez comprendre le principe en sachant que :

— open() ouvre le fichier. La valeur renvoyée est appelée un "descripteur de fichier", on peut la voir comme un curseur qui avance dans le fichier au fur et à mesure qu'on lit.

— f.read(1) lit un caractère à la position courante dans le fichier. Elle renvoie une chaîne vide si f est déjà à la fin du fichier.

— ord(c) renvoie le code ASCII du caractère c.

— f.close() ferme le descripteur de fichier f. Si on oublie de le faire, Python va continuer à utiliser des ressources de l'ordinateur inutilement.

Exécutez `display_file.py`. Vous devriez voir :

— Pour les fichiers binaires (`traitement-de-texte.odt`, et l'image `cpp.png`), une suite de caractères plus ou moins incompréhensible (certains ne s'affichent pas correctement, c'est normal). Pour l'image PNG, on voit apparaître les caractères PNG au début du fichier, ce n'est pas une coïncidence. Attention, même si `traitement-de-texte.odt` *ressemble* à un fichier texte, c'est un fichier au format OpenDocument bien plus compliqué qu'un fichier texte (on aurait eu un résultat similaire avec un fichier produit par MS Word). Pour obtenir un fichier texte avec MS Word ou LibreOffice, on peut faire : « Enregistrer Sous » puis choisir « Texte » comme format d'enregistrement.

— Pour les fichiers textes (programme Python, fichier CSV), les caractères correspondent à ce qu'on voit quand on ouvre le fichier.

1. les choses se compliquent si on veut représenter correctement les caractères accentués pour lesquels ASCII ne suffit pas malheureusement

6.2 Calcul à partir de données contenues dans un fichier

Nous allons maintenant faire un calcul simple sur des données lues depuis un fichier. On suppose qu'un instrument de mesure a fourni des données dans un fichier texte, avec une valeur par ligne, comme ceci :

```
1
12.3
43
3
10
```

6.2.1 Lecture d'un fichier ligne à ligne

Le but de cette section est de calculer la moyenne de ces valeurs. On commence avec le programme suivant :

```
f = open('donnees.txt', 'r')
ligne = f.readline()
while ligne != '':
    print("ligne␣=", ligne)
    ligne = f.readline()
```

Explications :

— Pour lire dans un fichier, il faut d'abord l'ouvrir. C'est ce que fait la fonction `open`, qui ouvre le fichier `donnes.txt` en lecture (`'r'`, pour « read »).

— La fonction `open` renvoie un objet `f` que l'on peut utiliser avec `f.readline()` qui veut dire « lire la ligne suivante dans le fichier ». Une fois la fin du fichier atteint, `f.readline()` renvoie la chaîne vide.

La variable `ligne` va donc contenir successivement les chaînes `"1"`, `"12.3"`, `"43"`, `"3"` et `"10"`[2]. On va maintenant extraire les nombres contenus dans cette ligne, et en faire la moyenne.

EXERCICE #2 ▶ **Ordre des instructions dans la boucle**
Essayez d'inverser les lignes print(ligne) et ligne = f.readline() et exécutez le programme. Cette inversion provoque deux problèmes : la première ligne n'est plus affichée, et une ligne blanche est affichée en fin de programme. Expliquez ces problèmes. Attention, il y a une ligne

En pratique, il est donc important d'exécuter ligne = f.readline() en dernier : c'est cette instruction qui marque le passage à l'itération suivante, donc c'est la dernière chose qu'on fait avant de revenir en tête de boucle et de tester ligne != '' à nouveau.

EXERCICE #3 ▶ **Calcul de moyenne**
En ajoutant quelques lignes au programme ci-dessus, calculez la moyenne des nombres lus. Attention, la fonction `f.readline()` renvoie une chaîne de caractère. Pour la convertir en nombre, on peut utiliser `float(...)`.

On peut bien sûr généraliser la méthode à des fichiers d'entrée plus compliqués, par exemple avoir plusieurs valeurs par ligne (typiquement séparées par des virgules).

6.3 Fichiers CSV

Nous allons maintenant travailler sur des fichiers permettant d'écrire plusieurs valeurs par ligne. Une convention classique est le CSV : *comma-separated values*.
Un exemple de fichier CSV est donné dans `donnees-2-colonnes.csv`.

```
0,10
1,12
4,15
```

2. plus précisément, la variable `ligne` contient elle-même une fin de ligne, ce qui fait que `print(ligne)` affiche deux retours à la ligne

EXERCICE #4 ▸ **Ouvrir un fichier "pas py" dans l'éditeur**
Ouvrez le fichier `donnees-2-colonnes.csv` dans Spyder (menu File → Open, puis choisir « All files » comme type de fichier).

Nous allons calculer des intégrales par la méthode des trapèzes, à partir de données disponibles dans un un fichier CSV[3] : la première colonne donne l'instant de la mesure t, et la seconde la vitesse v au moment de la mesure. L'objectif est de calculer la position du mobile après la dernière mesure.

EXERCICE #5 ▸ **Découpage d'une ligne**
Utiliser `ligne.split(',')` pour transformer `ligne` en une liste de chaînes (par exemple la liste `["0", " 10"]` pour la première ligne).

Extraire le premier élément et le convertir en flottant avec `float(...)` et mettre le résultat dans une variable `t`. De même, extraire le deuxième élément dans une variable `v`.

Vérifiez votre programme en ajoutant une ligne `print(t, v)`.

6.3.1 Calcul d'intégrale

EXERCICE #6 ▸ **Méthode des trapèzes**
Modifiez votre programme pour calculer la position du mobile en intégrant sa vitesse selon la méthode des trapèzes.

Indice : Pour calculer l'intégrale, ajoutez une variable `position` initialisée à 0 en début de fichier, puis dans le corps de la boucle, ajoutez une ligne du type `position = position + ...`. Pour faire une intégrale avec la méthode des trapèzes, il suffit d'ajouter $(t - t_{prec})\frac{v+v_{prec}}{2}$, où t_{prec} (resp. v_{prec}) est la valeur qu'avait t (resp. v) au précédent tour de boucle, ou 0 au premier tour (à vous de les définir en Python).

EXERCICE #7 ▸ **Écriture des résultats intermédiaires**
Ajoutez un affichage intermédiaire (avec print) de chaque valeur, pour produire un affichage comme :

```
0.0,0.0
1.0,11.0
4.0,51.5
```

En fait, cette sortie est aussi au format CSV, on peut la sauvegarder dans un fichier. Pour écrire dans un fichier, il faut :

— Ouvrir le fichier avec, par exemple : out_file = open('resultat.csv', 'w')
— Écrire dans le fichier avec : out_file.write (...)
— Fermer le fichier avec : out_file.close(). Tant que le fichier n'est pas fermé, les écritures faites via out_file.write (...) peuvent ne pas être appliquées (Python les garde en mémoire mais n'envoie pas les données sur disque immédiatement).

EXERCICE #8 ▸ **Écriture dans un fichier**
Modifiez votre programme pour qu'il produise son résultat dans un fichier `resultat.csv`. Ouvrez `resultat.csv` pour vérifier que le résultat a bien été calculé.

Si le temps le permet, essayez de modifier les valeurs du fichier d'entrée, relancez votre programme et vérifiez que `resultat.csv` est bien mis à jour (sous Spyder : menu File → Revert pour recharger un fichier ouvert).

6.3.2 Manipulations de chaînes

Pour découper les lignes du fichier CSV, nous avons utilisé ligne.split (',') . En réalité, vous avez tous les éléments pour coder cette fonction vous-mêmes, et c'est un exercice de manipulation de chaînes et de listes intéressant :

EXERCICE #9 ▸ **Découpage de chaîne**
Écrire une fonction decoupe(chaine) qui prend en paramètre une chaîne et qui renvoie une liste contenant les éléments de la chaîne séparés par des virgules. Par exemple :

3. Dans la vraie vie, on utiliserait un module Python comme csv mais ici nous allons faire les choses "à la main"

```
>>> decoupe("1,2,14")
['1', '2', '14']
>>> decoupe("1,2,14,666,")
['1', '2', '14', '666', '']
```

Indice : le code peut ressembler à ceci :

```
def decoupe(chaine):
    indice_virgule = -1
    resultat = []
    for i in range(len(chaine)):
        # ...
    return resultat
```

indice_virgule est l'indice de la dernière virgule rencontrée. On ajoute les éléments au résultat avec resultat.append(). Pour rappel, on peut accéder à la sous-chaîne de i à j − 1 avec chaine[i : j].

Vous pouvez tester votre code en remplaçant ligne.split (',') par un appel à decoupe dans votre programme (exercices 6.3 et 6.3.1).

6.4 Utilisation de la bibliothèque CSV

EXERCICE #10 ▶ **Exemple**
La bibliothèque `csv` est vraiment très simple à utiliser, on vous fournit un exemple dans l'archive (`ex_csvlib.py`).

EXERCICE #11 ▶ **À vous**
Vous trouverez sur le web une infinité de ressources sous ce format, regardez par exemple :

- Génération de données bidon, ou aléatoires : `http://www.generatedata.com/#generator`
- `https://data.grandlyon.com/`
- `http://donnees.ville.montreal.qc.ca/` une bible, par exemple `http://donnees.ville.montreal.qc.ca/dataset/monuments`
- `https://www.data.gouv.fr/fr/datasets/population/` le format dbase est lisible par OO, normalement, ensuite "save as" pour sauver un CSV.

Imaginez un exercice à faire avec vos élèves avec du traitement de telles données.