

## Examen

Durée 2h00, tiers temps additionnel 40 minutes

**Consignes** l'examen dure 2h00 et est sur 20 points. Aucun document n'est autorisé. Un aide mémoire Unix et Python est disponible à la page 5. Le barème est indicatif et proportionnel à la difficulté des exercices.

**Conditions spéciales d'examen** en raison de la situation sanitaire et des règles en vigueur :

- le port du masque en salle d'examen comme sur le reste du campus est obligatoire
- ne pas anonymiser vos copies
- il est interdit d'échanger du matériel entre étudiants
- signer une fois la feuille d'emargement avec votre stylo personnel, lors du passage du surveillant
- déposez votre carte d'étudiant ou pièce d'identité sur la table, de façon à ce qu'elle puisse être lue par le surveillant
- votre copie est ramassée à votre place par les surveillants.
- une fois la copie rendue, merci de quitter la salle sans provoquer d'attroupement et dans le respect de la distanciation

Le non respect de ces règles peut entraîner l'exclusion **immédiate** de la salle d'examen.

### 1 Question de cours (1 point)

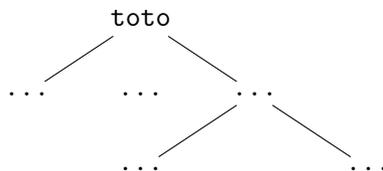
Qu'est-ce qu'un système d'exploitation ? (répondre en cinq lignes maximum).

### 2 Unix et ligne de commande (4 points)

- (1.5 point) On suppose être un utilisateur `toto` dont le répertoire personnel est « `/home/toto` ». On suppose que les commandes ci-dessous sont exécutées dans ce répertoire qui est initialement vide :

```
$ mkdir a
$ mkdir a/b
$ mkdir a/c
$ mkdir d
$ echo 'Hello' > f.txt
$ cp f.txt k.txt
$ mv *.txt a/b
$ mv a/b/[i-u].txt a/c
```

Dessiner, sous forme d'un arbre, le contenu final du répertoire `toto`, avec ce dernier comme racine. On s'attend à un dessin de la forme :



- (1.5 point) En utilisant la syntaxe avec permissions numériques, donner la ligne de commande permettant de rendre le répertoire `/home/toto/a` accessible :
  - en lecture, écriture et exécution pour le propriétaire,
  - en lecture et exécution pour le groupe et
  - sans permission pour les autres utilisateurs
- (1 point) On suppose que le contenu du répertoire courant est :

a\_5.Jpg abc.txt abc.jpg ab14.JPG 123.JPG Foo3.TXT

On exécute la commande : « `ls [a-z]*[0-9].[A-Z]*` ». Quels sont les noms de fichiers affichés ?

### 3 Réseaux (4 points)

- (3 points) On considère les adresses IP : 192.168.140.17 et 192.168.156.15, associées toutes les deux au masque : 255.255.240.0. Ces deux adresses dénotent-elles des machines sur le même sous-réseau ? On justifiera en donnant l'adresse réseau correspondant à chaque IP.

**Rappel :** on peut obtenir l'adresse Réseau  $R_1.R_2.R_3.R_4$  d'une adresse IP  $I_1.I_2.I_3.I_4$  associée à un masque  $M_1.M_2.M_3.M_4$  en calculant :  $R_i = I_i \& M_i$ , pour  $1 \leq i \leq 4$ , où  $\&$  est l'opération « et » bit à bit. Vous n'avez pas besoin de détailler sur votre copie les conversions en bases 2 dont vous pourriez avoir besoin.

- (1 point) Entre Ethernet et HTTP, lequel des deux protocoles se trouve le plus bas dans la couche des réseaux (question de cours, on ne demande pas de justification).

### 4 Programmation en Python (11 points)

On suppose qu'un site de commerce en ligne garde sous la forme d'un fichier texte les modifications de son stock de produits. Le fichier contient des lignes de la forme :

```
nom_produit:n
```

Un exemple de tel fichier est :

```
chaise en bois:3
table en verre:-2
coussin:-4
coffre de rangement:3
chaise en bois:-2
```

Un tel fichier signifie que trois chaises en bois ont été ajoutées au stock, puis que deux tables en verre ont été vendues, puis que quatre coussins ont été vendus, puis que trois coffres de rangement ont été ajoutés au stock puis que deux chaises ont été vendues.

Le but de cet exercice est d'écrire des fonctions Python permettant de calculer des statistiques et de faire des vérifications sur les stocks.

- (2 points) Écrire une fonction `charge_fichier(f)` prenant en argument une chaîne de caractères représentant un nom de fichier et qui renvoie un tableau de couples  $(n, q)$  où  $n$  est la chaîne de caractères correspondant au nom du produit et  $q$  la variation de quantité. On ne demande pas de gérer les erreurs, c'est à dire que vous pouvez supposer que le fichier dénoté par `f` existe, est accessible en lecture et qu'il est au bon format.

**Indication :** on pourra charger le contenu du fichier comme un tableau de chaînes de caractères, puis séparer chaque chaîne selon le caractère « : ». La première chaîne est le nom du produit. La seconde chaîne devra être privée de son `\n` final et convertie en entier. Pour le fichier donné en exemple, la fonction renvoie le tableau :

```
[ ("chaise en bois", 3), ("table en verre", -2), ("coussin", -4),
  ("coffre de rangement", 3), ("chaise en bois", -2) ]
```

- (1 point) Écrire une fonction `existe(tab, p)` prenant en argument un tableau `tab` de paires tel que renvoyé par la fonction `charge_fichier` et un nom de produit `p` et qui renvoie `True` si ce nom de produit apparaît dans le tableau et `False` sinon.
- (1.5 point) Écrire une fonction `max_var(tab)` prenant en argument un tableau `tab` de paires tel que renvoyé par la fonction `charge_fichier` et qui renvoie le nom du produit ayant la plus grande variation quantité en valeur absolue. Par exemple, pour le tableau correspondant au fichier décrit plus haut, la fonction renvoie la chaîne `"coussin"` parce que ce dernier est associé à `-4` qui a la plus grande valeur absolue.

4. (2 points) Écrire une fonction `stock_final(tab)` prenant en argument un tableau de paires tel que renvoyé par la fonction `charge_fichier` et qui renvoie un dictionnaire dont les clés sont les noms de produit et les valeurs le stock final des produits. On suppose que tous les produits ont un stock initial de 100. Par exemple pour le tableau correspondant au fichier décrit plus haut, on renverra le dictionnaire :

```
{
    "chaise en bois" : 101,
    "table en verre" : 98,
    "coussin" : 96,
    "coffre de rangement" : 103,
}
```

5. (1.5 point) Écrire une fonction `stock_valide(dic)` prenant en argument un dictionnaire tel que renvoyé par la fonction `stock_final`. La fonction `stock_valide(dic)` ne renvoie rien si tous les stocks sont positifs ou nuls et lève une erreur avec `assert` si l'un des stocks est négatif.
6. (1.5 point) Compléter la fonction `restock(dic)` prenant en argument un dictionnaire tel que renvoyé par la fonction `stock_final`. La fonction `restock(dic)` renvoie un tableau contenant les noms des produits dont le stock est nul. Vous devez **obligatoirement** utiliser un tableau par compréhension.

```
1 def restock(dic):
2     return [ ... ] # compléter le ...
```

Il est interdit de rajouter des instructions avant le `return` ou de définir des fonctions auxiliaires.

7. (1.5 point) Écrire une fonction `affiche_stock(dic)` prenant en argument un dictionnaire tel que renvoyé par la fonction `stock_final` et qui l'affiche dans la console au format

```
nom_produit:s
```

où `s` est le stock du produit. Les produits doivent être triés par ordre alphabétique. Ainsi, pour le dictionnaire donné en exemple dans la question 4, on affichera :

```
chaise en bois:101
coffre de rangement:103
coussin:96
table en verre:98
```

Attention, dans un dictionnaire, Python tel que celui pris en argument par la fonction, les clé **ne sont pas triées par ordre alphabétique par défaut**.



# Aide-mémoire Unix

## Expansion de la ligne de commande

On rappelle que lorsqu'une commande de la forme

$$\text{com } f_1 \dots f_n$$

est entrée dans le *shell* alors :

- Le fichier exécutable `com` est cherché dans les répertoires systèmes. S'il n'est pas trouvé, le *shell* renvoie une erreur
- Les motifs *glob*  $f_1 \dots f_n$  sont développées pour essayer de correspondre à des fichiers. Si aucun fichier ne correspond pour un motif *glob*  $f_i$  se dernier est laissé tel quel
- La liste des fichiers trouvés est passée en argument à la commande `com` qui est exécutée

## Redirections

On rappelle que lorsqu'une commande de la forme

$$\text{com } f_1 \dots f_n > o$$

est exécutée, alors la sortie standard de la commande est redirigée dans le fichier *o*. Si ce dernier existe il est écrasé. Les opérateurs de redirection sont :

- > redirige la sortie standard et écrase le fichier
- >> redirige la sortie standard et ajoute en fin de fichier
- 2> redirige la sortie d'erreur et écrase le fichier
- 2>> redirige la sortie d'erreur et ajoute en fin de fichier
- < redirige le fichier vers l'entrée standard du programme

## Commandes de base

`cat f` : affiche les lignes du fichier *f* sur la sortie standard.

`cd p` : le répertoire dénoté par le chemin *p* devient le répertoire courant.

`chmod nnn p` : modifie les permissions du fichier ou répertoire dénoté par le chemin *p*. Les permissions sont données comme trois entiers représentant respectivement les permissions pour le propriétaire, le groupe et les autres. Chaque entier est constitué de trois bits : bit en lecture, bit en écriture et bit en exécution.

`cp f1 f2` : si *f<sub>2</sub>* est un nom de fichier ou un fichier n'existant pas : copie *f<sub>1</sub>* sous le nom *f<sub>2</sub>*. Si *f<sub>2</sub>* est un nom de répertoire existant, copie *f<sub>1</sub>* dans le répertoire *f<sub>2</sub>*.

`echo s` : affiche la chaîne de caractères *s* sur la sortie standard.

`ls f1 ... fn` : affiche les fichiers *f<sub>i</sub>* (ou une erreur si les *f<sub>i</sub>* ne correspondent pas à des noms de fichiers existants).

`mkdir p` : crée le répertoire dénoté par le chemin *p*.

`mv p1 ... pn d` : déplace les fichier ou répertoires *p<sub>i</sub>* dans le répertoire *d* qui doit exister.

`rm f` : efface le fichier *f*.

`rm -rf p` : efface le répertoire *p*.

`sort f` : affiche les lignes triées du fichier *f* sur la sortie standard.

# Aide mémoire Python

## Entiers

Les entiers Python sont de taille arbitraire. Les constantes entières s'écrivent simplement `0`, `42`, `-233`. Les opérations et fonctions sur les entiers sont :

- + addition entre deux entiers (opérateur binaire).
- soustraction entre deux entiers (opérateur binaire).
- \* multiplication entre deux entiers (opérateur binaire).
- // division **entière** entre deux entiers (opérateur binaire).
- / division **flottante** entre deux entiers (opérateur binaire).
- % reste dans la division entière (opérateur binaire).
- \*\* puissance entre deux entiers (opérateur binaire)
- int(s)** conversion d'une chaîne `s` en entier. Lève une exception si la chaîne n'est pas au bon format.

## Flottants

Le type **float** représente des nombre flottants. Les constantes flottantes s'écrivent en notation scientifique `0.5`, `42e3`, `-233.8e-20`. Les opérations et fonctions sur les flottants sont :

- + addition entre deux flottants (opérateur binaire).
- soustraction entre deux flottants (opérateur binaire)
- \* multiplication entre deux flottants (opérateur binaire)
- / division entre deux flottants (opérateur binaire)
- \*\* puissance entre deux flottants (opérateur binaire)

## Booléens

Les constantes booléennes sont **True** et **False**. Les opérations et fonctions sur les booléens sont :

- and** « et » logique entre deux booléens (opérateur binaire).
- or** « ou » logique entre deux booléens (opérateur binaire).
- not** négation d'un booléen.

## Chaînes de caractères

Le type **str** représente des chaînes de caractères. Les chaînes de caractères constantes sont délimitées par des guillemets simples : `'Hello, world !'`, des guillemets doubles : `"Hello, world !"` ou des triples guillemets doubles. De façon usuelle, la séquence d'échappement `\n` représente un retour à la ligne. Les opérations et fonctions sur les chaînes sont :

`s[i]` permet d'accéder au `i`<sup>ème</sup> caractère de la chaîne. Ce dernier est renvoyé comme une chaîne de taille 1.

- + concaténation entre deux chaînes (opérateur binaire).

**len(s)** longueur d'une chaîne.

`s.lower()` renvoie une copie de la chaîne `s` où toutes les lettres sont converties en minuscules.

`s.split(c)` sépare la chaîne `s` selon le caractère de séparation `c` et renvoie un tableau des composantes. Par exemple :

```
1 s = "a,b,cd"
2 t = s.split(",")
3 # t contient [ "a", "b", "cd" ]
```

`s.strip()` renvoie une copie de la chaîne `s` où les blancs (espaces, tabulations, retours à la ligne) en début et en fin de chaîne ont été supprimés.

`s.upper()` renvoie une copie de la chaîne `s` où toutes les lettres sont converties en majuscule.

`str(v)` convertit la valeur `v` en chaîne.

## n-uplet

On peut regrouper plusieurs valeurs en les mettant dans un *n*-uplet. Les *n*-uplets sont délimités par des parenthèses et les valeurs séparées par des virgules : `(1, True, "AB")`.

`p[i]` permet d'accéder au *i*<sup>ème</sup> élément du *n*-uplet `p`.

`a, b, c, d = p` permet de définir les variables `a`, `b`, `c`, `d` comme les composantes correspondantes du *n*-uplet `p`.

`len(p)` nombre d'éléments d'un *n*-uplet.

## Tableaux

Les tableaux Python permettent de stocker des collections ordonnées de valeurs. Les tableaux constants sont délimités par des crochets et les éléments séparés par des virgules : `["A", "b", "TOTO"]`.

`t[i]` permet d'accéder au *i*<sup>ème</sup> élément du tableau `t`.

`t[i] = v` permet de remplacer le *i*<sup>ème</sup> élément du tableau `t` par `v`.

`+` concaténation entre deux tableaux (opérateur binaire).

`t * n` concaténation répétée *n* fois d'un tableau `t`. (opérateur binaire). Par exemple, `["A"] * 3` renvoie `["A", "A", "A"]`.

`len(t)` taille d'un tableau.

## Tableaux donnés par compréhension

La notation

```
[ e for x in t if c ]
```

renvoie le tableau formé par les expressions `e`. Ces dernières sont calculées tour à tour à partir de tous les éléments `x` du tableau `t` qui vérifient la condition `c`. Par exemple :

```
1 t = [ (x * 2, str(x)) for x in range(1,5) if x % 2 == 0 ]
2 # t contient
3 # [ (4, "2"), (8, "4") ]
```

Ici, `x` prend tour à tour les valeurs 1, 2, 3, 4 (5 est exclus du `range`). Les seules valeurs vérifiant `x % 2 == 0` sont 2 et 4, on renvoie donc le tableau « `[(2 * 2, str(2)), (4 * 2, str(4))]` ».

## Dictionnaires

Les dictionnaires Python permettent d'associer des clés à des valeurs. Les dictionnaires constants sont délimités par des accolades et les éléments séparés par des virgules. Les clés et les valeurs sont séparés par « `:` » : `d = {"A":1, "B": 42 }`.

`d[k]` permet d'accéder à la valeur associée à la clé `k`.

`d[k] = v` permet de remplacer ou ajouter la valeur associée à la clé `k`.

`k in d` renvoie `True` si et seulement si la clé `k` est dans le dictionnaire `d`.

`d.keys()` renvoie le tableau des clés du dictionnaire `d`.

`d.values()` renvoie le tableau des valeurs du dictionnaire `d`.

## Entrées sorties

`print(...)` permet d'afficher les valeurs passées en paramètres sur la sortie standard. Les valeurs affichées sont séparées par des espaces.

`open(p)` renvoie un descripteur de fichier correspondant au chemin `p`, donné comme une chaîne de caractères. Lève une erreur si le fichier n'existe pas ou n'est pas accessible en lecture.

`list(f.readlines())` renvoie le tableau des lignes du descripteur de fichier `f`. Toutes les lignes sauf éventuellement la dernière se terminent par `\n`.

`f.close()` referme le fichier associé au lignes du descripteur de fichier `f`.

## Comparaisons

En Python les opérations de comparaison permettent de comparer n'importe quelles valeurs du même type :

`<`, `<=`, `>`, `>=`, `==`, `!=` comparaisons entre deux valeurs (respectivement, inférieur, inférieur ou égal, supérieur, supérieur ou égal, égal et différent), (opérateur binaire).

`sorted(t)` renvoie une copie triée du tableau `t` trié par ordre croissant.

`min(a, b)` et `max(a, b)` renvoie le minimum ou maximum de deux valeurs.

Les entiers, flottants et chaînes de caractères sont comparés selon leur ordre usuel. Les booléens peuvent être comparés, et `False` est plus petit que `True`. Les n-upplets sont comparés composante par composante, par exemple `(1, 10)` est plus petit que `(2, 5)` qui est lui même plus petit que `(2, 6)`. Il en va de même pour les tableaux. Les dictionnaires **ne sont pas comparables** (les comparer ou appeler `sorted` sur un tableau de dictionnaires lève une exception).