université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

vendredi 7 janvier 2022

Introduction à l'Informatique
L1 MI, LDD1 IM, LDD1 MNSI

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 9. 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'émargement 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 entrainer l'exclusion **immédiate** de la salle d'examen.

1 Question de cours (1 point)

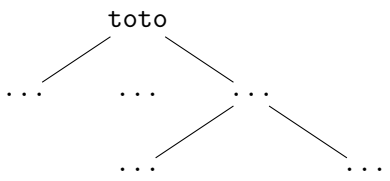
Rappeler brièvement les rôles distincts des formats HTML et CSS dans le cadre de la conception d'une page Web.

2 Unix et ligne de commande (4 points)

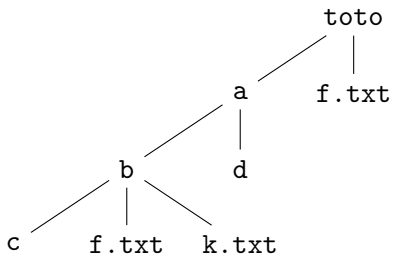
1. (1.5 point) On suppose être un utilisateur `toto` dont le répertoire personnel est « `/home/toto` ». On suppose que les commandes ci-dessous sont exécutées dans ce répertoire qui est initialement vide :

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

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 :



Réponse:
On obtient comme arbre final :



2. (2 points) On suppose que l'on est placé dans le répertoire `/tmp`. On suppose trois utilisateurs `alice`, `bob` et `charlie`. Les utilisateurs `bob` et `alice` sont dans le groupe `user` et l'utilisateur `charlie` dans le groupe `staff`. On suppose que les commandes suivantes sont exécutées par `bob` :

```
mkdir a
echo 'Salut les amis' > a/t.txt
chmod 751 a
chmod 664 a/t.txt
```

On suppose que le répertoire `/tmp` est accessible et lisible par tout le monde. Répondre par oui/non au questions en justifiant brièvement :

(a) `alice` peut se déplacer dans le répertoire `a`

(b) `charlie` peut se déplacer dans le répertoire `a`

(c) `alice` peut lister le contenu du répertoire `a`

(d) `charlie` peut lister le contenu du répertoire `a`

(e) `alice` peut lire le fichier `a/t.txt`

(f) `charlie` peut lire le fichier `a/t.txt`

(g) `alice` peut modifier le contenu du fichier `a/t.txt`

(h) `alice` peut supprimer le fichier `a/t.txt`

Réponse: On rappelle que les trois chiffre des permissions représente celles du propriétaire, du groupe et de tous les autres. Ainsi, pour le répertoire `a`, la permission `751` signifie :
— lecture, écriture et exécution (7), pour `bob`
— lecture et exécution (5) pour `alice` (qui est dans le même groupe)
— exécution (1) pour `charlie` (qui est dans un autre groupe, donc les permissions « pour les autres » s'appliquent).
Pour le fichier `t.txt`, sur le même principe :
— lecture et écriture (6), pour `bob`
— lecture et écriture (6), pour `alice`
— lecture (4) pour `charlie`

(a) oui car elle a les droits en exécution sur ce répertoire

(b) oui car il a les droits en exécution sur ce répertoire

(c) oui car elle a les droits en lecture sur ce répertoire

(d) non car il n'a pas les droits en lecture sur ce répertoire

(e) oui car elle a les droits en lecture sur ce fichier

(f) oui car il a les droits en lecture sur ce fichier

(g) oui car elle a les droits en écriture sur ce fichier

(h) non car il n'a pas les droits en écriture sur ce fichier

3 Réseaux (4 points)

1. (3 points) On considère les adresses IP : `192.168.128.10` et `192.168.159.42`, associées toutes les deux au masque : `255.255.224.0`. Ces deux adresses dénotent elles des machines sur le même sous-réseau ? On justifiera en donnant l'adresse réseau correspondant à chaque IP.

   Remarque : 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.

> **Réponse:** On rappelle que (255) est constitué uniquemende 1 en base 2 (toutes les $2^n - 1$ ont cette propriété) et donc faire le « et » bit à bit entre un octet et 255 est l'identité. De façon duale faire le « et » bit à bit avec 0 renvoie 0. On a donc que l'adresse réseau de la première IP est 192.168.(128&224).0 et celle de la seconde est 192.168.(159&224).0. Il suffit donc de calculer la valeur du troisième octet dans les deux cas, si elles sont égales les deux IP correspondent au même sous réseau. On calcule donc :
> — $128_{10} = 10000000_2$
> — $224_{10} = 11100000_2$
> — $159_{10} = 10011111_2$ (voir la note sur la page du cours pour savoir comment faire les convertions).
> Ainsi, $128_{10} \& 224_{10} = 10000000_2 \& 11100000_2 = 10000000_2 = 128_{10}$ et $159_{10} \& 224_{10} = 10011111_2 \& 11100000_2 = 10000000_2 = 128_{10}$. Les deux IPs sont sur le même sous-reseau.

2. (1 point) Qu'est-ce qu'un routeur ?

> **Réponse:** Un routeur est une machine avec au moins deux cartes réseaux servant à retransmettre les paquets entre deux sous-reseaux physiques.

4 Programmation en Python (11 points)

On suppose qu'un site de gestion d'évènements enregistre des évènements dans des fichiers sous la forme suivante :

```
1641225600:300:Acheter du pain
1641559500:7200:Exam L1
1641240000:3600:Répetition piano
1641283200:10800:Cours M1
```

Ce fichier est composé de lignes. Chaque ligne contient trois champs, séparés par des « : ». Le premier champ est la date de l'évènement donné sous le *format d'heure Unix*. Ce dernier représente le nombre de secondes écoulées depuis le 1er janvier 1970, 00h00min0s UTC. En effet, il s'agit d'un moyen commode de stocker le temps de façon universelle sans se soucier de problèmes annexes (fuseaux horaires, heure d'été ou d'hiver). Le second champ représente la durée de l'évènement en secondes. Le troisième champ est une chaîne de caractères ne contenant pas de caractère « : » et décrivant l'évènement.

Dans l'exemple ci-dessus, les dates correspondent respectivement au 3 janvier 2022 17h00, 7 janvier 2022 13h30, 3 janvier 2022 21h et 4 janvier 2022 9h00. Les durées étant données en secondes, elles valent respectivement 5 minutes, 2h, 1h, 3h. On remarque que ces évènements ne sont pas triés par date de début, mais peuvent apparaitre dans n'importe quel ordre dans le fichier.

Le but de cet exercice est d'écrire des fonctions Python permettant de manipuler de telles listes d'évènements.

1. (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 triplets `(`t`, `d`, `s`)` où t est un entier correspondant au premier champ, d un entier correspondant au second champ et s une chaîne de caractères correspondant au troisième champ.

   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 « : ». On n'oubliera pas de convertir les champs qui le nécessitent en entier. Attention, on n'oubliera pas non plus de supprimer le `\n` se trouvant en fin de ligne. Pour le fichier donné en exemple, la fonction doit renvoyer le tableau :

```
[
  (1641225600, 300, "Acheter du pain"),
  (1641559500, 7200, "Exam L1"),
  (1641240000, 3600, "Répetition piano"),
  (1641283200, 10800, "Cours M1")
]
```

Dans la suite, on appelle un tel tableau un **tableau d'évènements**. A priori ce tableau n'est pas trié (comme dans notre exemple, le second évènement du tableau se déroule chronologiquement après les autres).

Réponse:

```
1  def charge_fichier (f):
2      desc = open(f)
3      res = list(desc.readlines())
4      for i in range(len(res)):
5        l = res[i].strip().split(":") #retire le \n et découpe selon :
6        res[i] = (int(l[0]), int(l[1]), l[2])
7      close(desc)
8      return res
```

2. (1.5 point) Écrire une fonction `min_max_duree(tab)` qui prend en argument un tableau d'évènements et qui renvoie un couple composé de l'évènement le plus court et du plus long (dans cet ordre). Votre fonction doit renvoyer une paire de triplets.

Réponse:

```
1    def min_max_duree (tab):
2        assert len(tab) >= 1 #la fonction n'est pas
3                             #définie pour les tableaux vides
4
5        #initialise le min et le max par
6        #le premier element du tableau
7        min_e = tab[0]
8        max_e = tab[0]
9
10       for i in range(1,len(tab)):
11         e = tab[i]
12         if e[1] < min_e[1]:
13           min_e = e
14         if e[1] > max_e[1]:
15           max_e = e
16
17       return (min_e, max_e)
```

Remarque : l'énoncé ne dit pas quoi faire en cas de tableau vide, on peut donc choisir comme on veut. Ici on fait le choix d'un assert.

3. (1.5 point) Étant donné une date t au format d'heure Unix et en sachant que le premier janvier 1970 était un jeudi, écrire une fonction `jour_date(t)` qui renvoie le jour correspondant à la date Unix `t`

donnée en argument. On suppose que vous disposez d'une variable globale
`JOUR = ["lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"]`
Indication il y a $3600 \times 24 = 86400$ secondes dans un jour. On fera ensuite une utilisation judicieuse de la division entière et du modulo pour savoir combien de jours se sont écoulés depuis le 1er janvier 1970.

Réponse:

```
1  def jour_date (t):
2    j = t // 86400  #nombre de jours écoulés depuis le 1/1/1970
3    d = j mod 7     #le nombre de jour manquant pour faire
4                    #une semaine complète
5    #à ce stade, on sait que t est d jours de la semaine après un jeudi.
6    #la position du jeudi dans le tableau JOUR est 3.
7    i = (d + 3) mod 7
8    return JOUR[i]
```

4. (1 point) Écrire une fonction `temps_occupé(tab)` prenant en argument un tableau d'évèmements `tab` et qui renvoie le nombre de secondes occupées par tous les évènements mis bout à bout. On suppose pour cette question qu'il n'y a pas d'évènement qui se chevauchent.

Réponse: Sans difficulté, on fait la somme des durées (composante 1 du triplet) de chaque entrée du tableau.

```
1  def temps_occupe (tab):
2    total = 0
3    for e in tab:
4      total += e[1]
5    return total
```

5. (2 points) Écrire une fonction `conflit(tab)` prenant en argument un tableau d'évèmements `tab` et qui renvoie `True` si deux évènements se chevauchent dans le tableau et `False` sinon.
Indication On pourra commencer par trier le tableau par date de début d'évènement (c.f. Annexe page 8). On pourra ensuite parcourir le tableau et regarder si deux évènement successifs dans ce tableau sont tels que le second commence avant que le premier termine.

Réponse: Algorithme classique. Les triplets étant ordonnés d'abord par première composante, puis par les suivantes en cas d'égalité, il suffit de trier le tableau des évènements puis de comparer un événement et le suivant pour savoir s'ils se chevauchent.

```
1  def conflit (tab):
2    stab = sorted(tab)
3    for i in range(len(stab) - 1): #on ne va pas jusqu'au dernier
4        cur = stab[i]
5        suiv = stab[i+1]
6        if cur[0] + cur[1] > suiv[0]:
7          #si l'évènement courant se termine après le début
8          #du suivant
9          return True #on a trouvé un conflit, on sort
10
11   return False #en fin de boucle, on n'a trouvé aucun conflit.
```

6. (1.5 points) Écrire une fonction `temps_par_activite(tab)` prenant en argument un tableau d'évèmements `tab` et qui renvoie un dictionnaire dont les clés sont les textes des différents évènements et les valeurs les sommes des durées de tous les évèmenent ayant ce texte.

```
1  def conv_html(tab, t):
2      print ("""<!DOCTYPE html>
3  <html>
4      <head>
5      <title>Mon calendrier</title>
6      <style>
7      ul li.passe { color : gray; text-decoration: italic; }
8      ul li.future { color : blue; font-weight: bold; }
9      </style>
10     </head>
11     <body>
12     """)
13
14     ### À Compléter à partir d'ici
15
16     print ("""    </body>
17 </html>""")
```

FIGURE 1 – Code de la fonction `conv_html`

Réponse: Utilisation d'un dictionnaire comme vu en cours. Pour chaque évènement, si la description est déjà dans le dictionnaire, on y ajoute le temps de cette évènement. Si elle n'y est pas on l'initialise avec la durée de l'évènement.

```
1  def temps_par_activite (tab):
2        res = {}
3        for e in tab:
4          if e[2] in res:
5            res[e[2]] += e[1]
6          else:
7            res[e[2]] = e[1]
8
9        return res
```

7. (1.5 point) Compléter la fonction **`conv_html(tab, t)`** dont le code est donné à la figure 1, page 6. Cette fonction prend en argument un tableau d'évènements et une date t au format d'heure Unix et produit sur la sortie standard un fichier HTML dans lequel les évènements sont présentés sous forme d'une liste non énumérée (balise `<ul>/</ul>` contenant une suite de balises `<li> ... </li>`). Les évènements doivent être ordonnés par date de début. Les éléments `<li >` correspondant à un élément avant `t` doivent avoir l'attribut `class` valant `"passe"` et les autres l'attribut `class` valant `"futur"`.

   Remarque On demande uniquement d'écrire le code à insérer à partir de la ligne 14. Vous pouvez bien sûr utiliser plusieurs lignes, définir des variables, utiliser des boucles (on ne demande pas à ce que tout le code tienne sur une ligne).

Réponse: L'énoncé ne précise pas quoi faire si un événement commence exactement à la date `t`, on choisit de considérer ça comme le passé.

```
1  ...
2  ### À Compléter à partir d'ici
3  print("<ul>")
4  for e in sorted(tab):
5        print("<li ")
6        if e[0] <= t:
7          print("class='passe'">)
```

```
        else:
          print("class='futur'">)
        print(e[0], e[1], e[2])
        print("</li>")

  print("</ul>)
  ...
```

Aide-ménoire Unix

Expansion de la ligne de commande

On rappelle que lorsqu'une commande de la forme

$$\texttt{com}\ f_1\ \ \ldots\ f_n$$

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* $f_1\ \ \ldots\ f_n$ 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

$$\texttt{com}\ f_1\ \ \ldots\ f_n > o$$

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` f_1 f_2 : si f_2 est un nom de fichier ou un fichier n'existant pas : copie f_1 sous le nom f_2. Si f_2 est un nom de répertoire existant, copie f_1 dans le répertoire f_2.

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

`ls` $f_1\ \ \ldots f_n$: affiche les fichiers f_i (ou une erreur si les f_i ne correspond pas à des noms de fichiers existants).

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

`mv` $p_1\ \ \ldots\ \ p_n\ \ d$: déplace les fichier ou répertoires p_i 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 $\texttt{i}^{\text{ème}}$ 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 cararctère 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`$^{\text{ème}}$ 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`$^{\text{ème}}$ élément du tableau `t`.

`t[i] = v` permet de remplacer le `i`$^{\text{ème}}$ 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).