Autotest n°3 – Correction

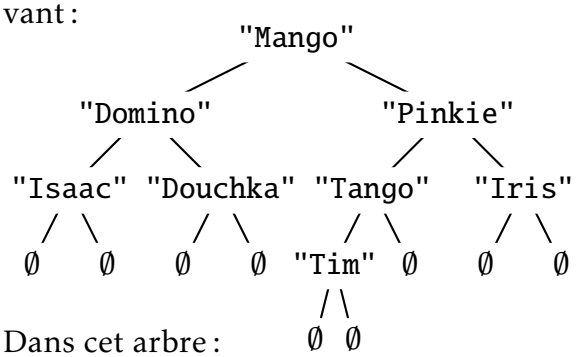
Exercice 1 : *Cet exercice porte sur les arbres binaires et leurs algorithmes associées.*

Un éleveur de chiens gère les informations sur ses animaux à l'aide d'un logiciel qui mémorise le pédigrée de chacun de ses chiens. Le pédigrée d'un chien correspond à son arbre généalogique.

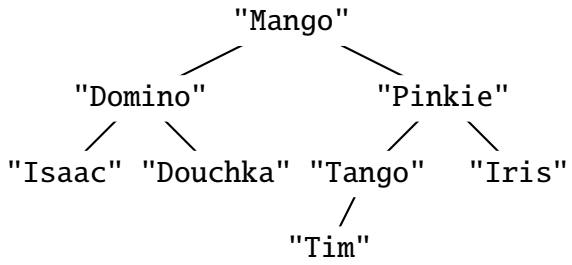
Une structure **arbre de pédigrée** est définie récursivement, soit par un arbre, noté ∅, soit par un arbre binaire où :

- la valeur du nœud est une chaîne de caractères qui représente le nom de l'animal ;
- le sous-arbre gauche est un l'arbre de pédigrée du père du chien ;
- le sous-arbre droit est un l'arbre de pédigrée de la mère du chien.

On représente donc graphiquement un arbre de pédigrée comme l'arbre `A` suivant :



Pour alléger la représentation d'un arbre de pédigrée, on ne notera pas les arbres vides. L'arbre précédent sera donc représenté comme ci-dessous.



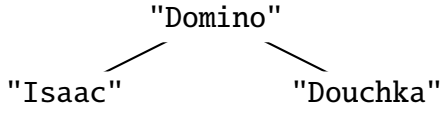
Dans cet arbre :

- le père de Mango est Domino et sa mère Pinkie ;
- les parents de Douchka ne sont pas connus ;
- Iris est la mère de Pinkie ;
- la mère de Tango n'est pas connue.

Pour manipuler les arbres de pédigrée, on dispose des quatre fonctions suivantes :

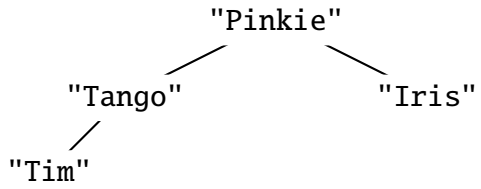
- la fonction `racine` qui prend en paramètre un arbre de pédigrée non vide et qui renvoie la valeur de la racine.
  Exemple : en reprenant l'exemple de l'arbre de pédigrée `A`, `racine(A)` vaut `"Mango"`.

- La fonction `gauche` qui prend en paramètre un arbre de pédigrée non vide et renvoie son sous-arbre gauche correspondant à l'arbre de pédigrée du père.

  Exemple : en reprenant l'exemple de l'arbre de pédigrée `A`, `gauche(A)` est l'arbre représenté graphiquement ci-contre :

- La fonction `droit` qui prend en paramètre un arbre de pédigrée non vide et renvoie son sous-arbre gauche correspondant à l'arbre de pédigrée de la mère.

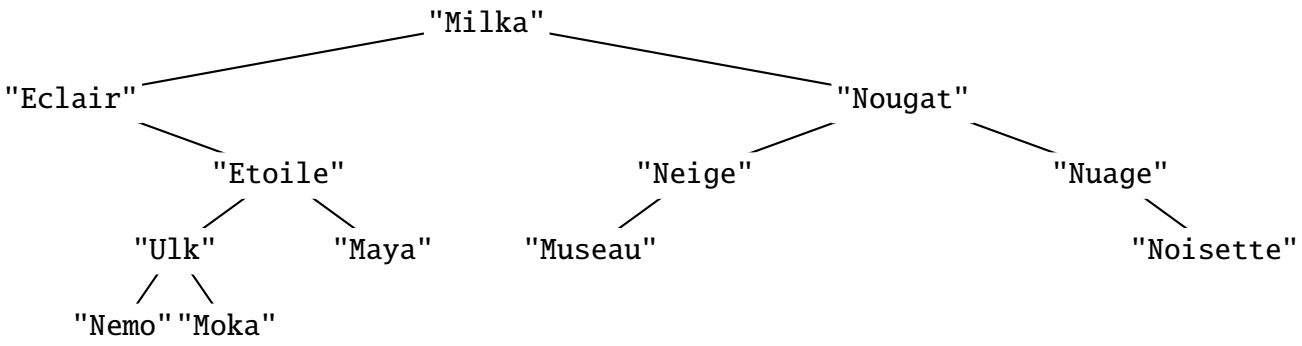
  

  Exemple : en reprenant l'exemple de l'arbre de pédigrée `A`, `droit(A)` est l'arbre représenté graphiquement ci-contre :

- La fonction `est_vide` qui prend en paramètre un arbre de pédigrée et renvoie `True` si l'arbre est vide ou `False` sinon.
  Exemple : en reprenant l'exemple de l'arbre de pédigrée `A`, `est_vide(A)` vaut `False`.

Pour toutes les questions de l'exercice, on suppose que tous les chiens d'un même pédigrée ont un nom différent.

1) On considère l'arbre de pédigrée `B` suivant :



a) Déterminer la valeur de la racine de cet arbre. **Solution :** `"Milka"`

b) On appelle feuille d'un arbre de pédigrée, un nœud dont les sous-arbres gauche et droit sont vides.
Déterminer l'ensemble des valeurs des feuilles de cet arbre.
Solution : `"Nemo"`, `"Moka"`, `"Maya"`, `"Museau"` et `"Noisette"`.

c) Déterminer si `"Nuage"` est un mâle ou une femelle.
Solution : `"Nuage"` est la racine du sous-arbre droit de `"Nougat"`, c'est donc une femelle.

d) Déterminer le père et la mère de `"Etoile"`.
Solution : Le père est `"Ulk"` et la mère `"Maya"`.

2) a) Recopier et compléter la fonction récursive Python `present` ayant pour paramètres un arbre de pédigrée `arb` et le nom d'un chien `nom` et qui renvoie **`True`** si ce nom est présent dans l'arbre de pédigrée ou **`False`** sinon.

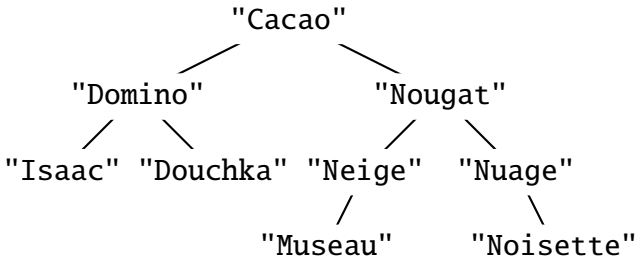
```
def present(arb, nom):
    if est_vide(arb):
        return False
    elif racine(arb) == nom:
        return True
    else:
        return present(gauche(arb), nom) or present(droit(arb), nom)
```

Pour toutes la suite de l'exercice, on pourra utiliser la fonction `present` même si la question 2)*a*) n'a pas été traitée.

b) Écrire une fonction Python `parents` ayant pour paramètres un arbre de pédigrée `arbre` d'un chien et qui renvoie le p-uplet `parents` des deux parents de ce chien dans l'ordre père, mère. Si un des parents est inconnu, il sera noté `""`.
Exemple : `parents(B)` vaut `("Eclair", "Nougat")`.

```
def parents(arb):
    p_pere = gauche(arb)
    p_mere = droit(arb)
    if est_vide(p_pere):
        pere = ""
    else:
        pere = racine(p_pere)
    if est_vide(p_mere):
        mere = ""
    else:
        mere = racine(p_mere)
    return (pere, mere)
```

3) a) On dit que deux chiens sont frères et sœurs s'ils ont le même père ou la même mère.
On considère les trois arbres de pédigrée `A`, `B` et `C` donné ci-contre.
Parmi les trois chiens Mango, Milka et Cacao, déterminer les liens de fratrie.



Solution : Mango et Cacao ont le même père. Milka et Cacao ont la même mère.

b) Écrire une fonction Python `frere_soeur` ayant pour paramètres deux arbres de pédigrée `arbre1` et `arbre2` correspondant à deux chiens. Cette fonction renvoie **`True`** si les deux chiens ont le même père ou la même mère ou **`False`** sinon.

```
def frere_soeur(arbre1, arbre2):
    pere1, mere1 = parents(arbre1)
    pere2, mere2 = parents(arbre2)
    return ((pere1 == pere2 and pere1 != "")
         or (mere1 == mere2 and mere1 != ""))
# ou
def frere_soeur(arbre1, arbre2):
    return (parents(arbre1) == parents(arbre2)
            and parents(arbre1) != ("", ""))
```

4) Étant donné l'arbre de pédigrée d'un chien, on considère que :

- le niveau 0 est le niveau de la racine contenant le nom du chien ;
- le niveau 1 est le niveau des parents du chien ;
- le niveau 2 est le niveau des grands-parents du chien ;
- etc.

Proposer une fonction Python `nombre_chiens` ayant pour paramètres un arbre de pédigrée `arb` et un entier `n` et qui renvoie le nombre de noms connus dans l'arbre de pédigrée `arb` au niveau `n`.

Exemple : Dans l'arbre de pédigrée `B`, `nombre_chiens(B, 3)` vaut 4 car les noms des chiens mentionnés dans l'arbre de pédigrée au niveau 3 sont `"Ulk"`, `"Maya"`, `"Museau"` et `"Noisette"`.

```python
def nombre_chiens(arb, n):
    if est_vide(arb):
        return 0
    elif n == 0:
        return 1
    else:
        nb_gauche = nombre_chiens(gauche(arb), n-1)
        nb_droite = nombre_chiens(droit(arb), n-1)
        return nb_gauche + nb_droite
```

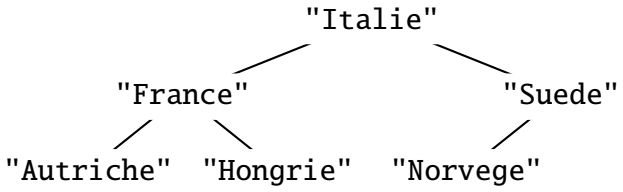
Exercice 2 : *Cet exercice porte sur les arbres binaires de recherche et les algorithmes associés*

Les arbres binaires de recherche considérés ici sont des arbres binaires où les nœuds désignent des chaînes de caractères et pour lesquelles la valeur de chaque nœud est supérieure à celles des nœuds de son enfant gauche, et inférieure à celles des nœuds de sont enfant droit. La relation d'ordre notée < est ici la relation d'ordre alphabétique.

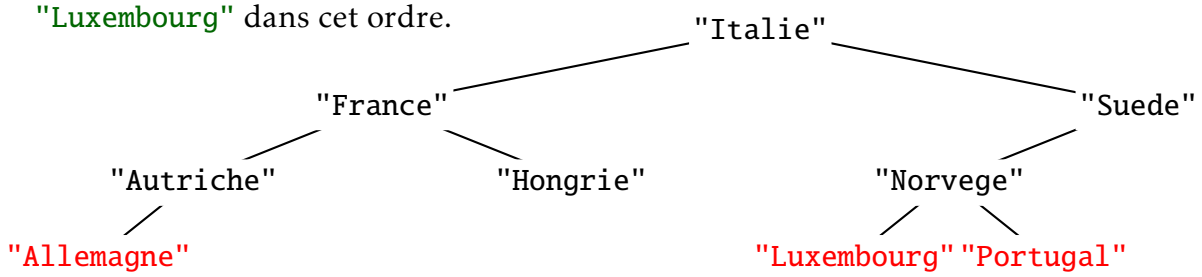
Dans cet exercice, on utilisera la convention suivantes : la hauteur d'un arbre binaire ne comportant qu'un nœud est 1.

Dans cet exercice les arbres binaire de recherche ne contiennent que des noms de pays tous distincts.

On considère l'arbre binaire de recherche ci-contre :

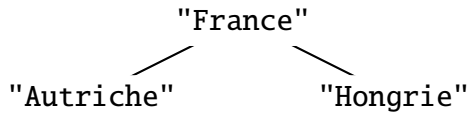


1) a) Donner sans justification la hauteur de cet arbre.
   Solution : La hauteur est de 3.
   b) Donner sans justification la valeur booléenne de l'expression `"Allemagne" < "Portugal"`.
   Solution : `"Allemagne" < "Portugal"` vaut **True**.
   c) Compléter l'arbre ci-dessous après l'ajout de `"Allemagne"`, de `"Portugal"` et de `"Luxembourg"` dans cet ordre.

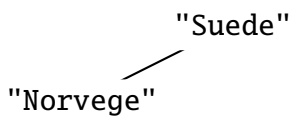


Pour les questions 2, 3 et 4 on traite de l'arbre initial, donc sans l'ajout de `"Allemagne"`, `"Portugal"` et `"Luxembourg"`.

2) On souhaite parcourir l'arbre. Indiquer l'ordre de visite des nœuds lors d'un parcours en largeur. **Solution :** `"Italie"`, `"France"`, `"Suede"`, `"Autriche"`, `"Hongrie"`, `"Norvege"`
3) On souhaite écrire une fonction pour déterminer si le nom d'un pays est dans l'arbre.
   On dispose pour cela de :
   - la fonction `est_vide` qui prend en paramètre un arbre `arb`. Cette fonction renvoie **True** si l'arbre `arb` est vide, **False** sinon ;
   - la fonction `gauche` qui prend en paramètre un arbre `arb` et renvoie son sous-arbre gauche.
     Exemple : si `A` est notre arbre initial, `gauche(A)` renvoie l'arbre ci-contre.



- la fonction `droite` qui prend en paramètre un arbre `arb` et renvoie son sous-arbre droit.
  Exemple : si `A` est notre arbre initial, `droite(A)` renvoie l'arbre ci-contre.



- la fonction `racine` qui prend en paramètre un arbre `arb` et renvoie la valeur de la racine de l'arbre.
  Exemple : `racine(A)` renvoie `"Italie"`.

Recopier, en complétant les lignes 2, 6, 7 et 10, la fonction `recherche` donnée ci-dessous et écrite en Python. Cette fonction prend en paramètre un arbre `arb` et une valeur `val`. L'appel `recherche(arb, val)` renvoie un booléen (`True` si la valeur `val` est dans l'arbre `arb`, `False` sinon).

```
1  def recherche(arb, val):
2      """ renvoie un booléen indiquant si val est dans arb """
3      if est_vide(arb):
4          return False
5      if val == racine(arb):
6          return True
7      if val < racine(arb):
8          return recherche(gauche(arb), val)
9      else:
10         return recherche(droite(arb), val)
```

4) Écrire une fonction récursive `taille` permettant de déterminer le nombre de pays présents dans un arbre.
   Cette fonction prendra en paramètre un arbre `arb` et renverra un entier.

```
def taille(arb):
    if est_vide(arb):
        return 0
    else:
        return 1 + taille(gauche(arb)) + taille(droite(arb))
```

Exercice 3 : *Cet exercice porte sur la gestion des processus et des ressources par un système d'exploitation.*

Les parties A et B peuvent être traitées indépendamment.

Partie A : Ordonnancement des processus

Dans le laboratoire d'analyse médicale d'un hôpital, plusieurs processus peuvent demander l'allocation du processeur simultanément.

Le tableau ci-dessous donne les demandes d'exécution de quatre processus et indique :

- le temps d'exécution du processus (en unité de temps) ;
- l'instant d'arrivée du processus sur le processeur (en unité de temps) ;
- le numéro de priorité du processus (classé de 1 à 10).

Plus la priorité est grande plus le numéro de priorité est petit.

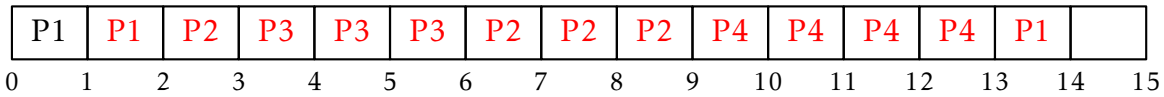
Ainsi le processus P3, du tableau ci-dessous, est plus prioritaire que le processus P1.

L'ordonnancement est de type préemptif, ce qui signifie qu'à chaque unité de temps, le processeur choisit d'exécuter le processus ayant le plus petit numéro de priorité (un seul processus à la fois). Ceci peut provoquer la suspension d'un autre processus qui reprendra

lorsqu'il deviendra le plus prioritaire dans la file d'attente.

Processus	Temps d'exécution	Instant d'arrivée	Numéro de priorité
P1	3	0	4
P2	4	2	2
P3	3	3	1
P4	4	5	3

1) Compléter le diagramme ci-dessous et indiquer dans chacune des cases le processus exécuté par le processeur entre deux unités de temps (il peut y avoir des cases vides).



2) Compléter les temps de séjour ainsi que les temps d'attente de chacun des processus (toujours en unités de temps).

Temps de séjour = instant de terminaison – instant d'arrivée

Temps d'attente = temps de séjour – temps d'exécution

Processus	Temps d'exécution	Instant d'arrivée	Numéro de priorité	Temps de séjour	Temps d'attente
P1	3	0	4	14 – 0 = 14	14 – 3 = 11
P2	4	2	2	9 – 2 = 7	7 – 4 = 3
P3	3	3	1	6 – 3 = 3	3 – 3 = 0
P4	4	5	3	13 – 5 = 8	8 – 4 = 4

3) À quelles conditions le temps d'attente d'un processus peut-il être nul?
Solution : Il faut que ce soit le processus le plus prioritaire pendant toute son exécution.

Partie B : Processus et ressources

Dans ce laboratoire d'analyse médicale de l'hôpital, le laborantin en charge du traitement des différents prélèvements (sanguins, urinaires et biopsiques) utilise simultanément quatre logiciels :

- Logiciel d'analyse d'échantillons (connecté à l'analyseur)
- Logiciel d'accès à la base de données des patients (SGBD)
- Traitement de texte
- Tableur

Le tableau ci-dessous donne l'état à un instant donné des différents processus (instances des programmes) qui peuvent soit mobiliser (M) des données (D1, D2, D3, D4 et D5), soit être en attente des données (A) ou ne pas les solliciter (-).

Une donnée ne peut être mobilisée que par un seul processus à la fois. Si un autre processus demande une donnée déjà mobilisée, il passe en attente.

Exemple : le SGBD mobilise la donnée D4 et est en attente de la donnée D5

	D1	D2	D3	D4	D5
Analyseur échantillon	M	-	-	A	-
SGBD	-	-	-	M	A
Traitement de texte	-	M	A	-	-
Tableur	A	-	M	-	M

1) À partir du tableau ci-dessus, démontrer que, à cet instant, les processus s'attendent mutuellement.
Solution : Toutes les ressources sont mobilisées et tous les processus sont en attente d'une autre ressource. Aucun de ces processus ne peut avancer.

2) Comment s'appelle cette situation ?
Solution : C'est une situation d'interblocage.

3) On suppose que l'analyseur d'échantillon libère la ressource D1. Donner un ordre possible d'exécution des processus.
Solution :
- Le tableur peut terminer son exécution et libérer D1, D3 et D5.
- Le traitement de texte peut terminer son exécution et libérer D2 et D3.
- Le SGBD peut terminer son exécution et libérer D4 et D5.
- L'analyseur d'échantillon peut terminer son exécution.