Lycée Les 3 Sources — **Bourg-Lès-Valence**
NSI 1re — Année **2025-26**

Devoir surveillé n°3 – correction

Exercice 1 : (1.5pt) Entourer la bonne réponse à chaque question.

1) Si `Décalage` s'écrit "`44 c3 a9 63 61 6c 61 67 65`" en UTF-8 en hexadécimal, quelle peut être l'écriture de `Glacé` ?
   a) `67 6c 61 63 a9 c3`
   b) `47 6c 61 63 c3 a9`
   c) `67 6c 61 63 c3 a9`
   d) `47 6c 61 63 a9 c3`

2) Déterminer la valeur de `score` après les instructions ci-contre :

```
>>> score = {}
>>> score['Alice'] = 4
>>> score['Bob'] = 7
>>> score['Alice'] = 15
```

   a) `{'Alice': 19, 'Bob': 7}`
   b) `{'Alice': 15, 'Bob': 7}`
   c) `{'Alice': 4, 'Bob': 7, 'Alice': 15}`
   d) `[4, 7, 15]`

3) On considère le code suivant :

```
def menagerie(animaux):
    reponse  = dict()
    for animal in animaux:
        if animal in reponse:
            reponse[animal] = reponse[animal] + 1
        else:
            reponse[animal] = 1
    return reponse

enclos = ['poule', 'poule', 'coq', 'poussin', 'poule', 'poussin']
```

   Quelle est la valeur de `menagerie(enclos)` ?

   a) `{'poule': 3, 'coq': 1, 'poussin': 2}`
   b) `{'poule': 1, 'coq': 1, 'poussin': 1}`
   c) `{'poule': 1, 'poule': 1, 'coq': 1, 'poussin': 1, 'poule': 1, 'poussin': 1}`
   d) `{3: 'poule', 1: 'coq', 2: 'poussin'}`

Exercice 2 : (2pt) On donne ci-contre l'architecture des dossiers de Samia sur l'ordinateur de la maison. Les fichiers ne sont pas affichés.

```
/
└── home
    └── samia
        ├── boulot
        │   ├── euro
        │   ├── maths
        │   └── NSI
        └── perso
            ├── photos
            └── vidéos
```

1) Donnez le chemin absolu du dossier `NSI`.
   `/home/samia/boulot/NSI`
2) Le dossier courant dans le terminal est `perso`. Donnez le chemin relatif permettant d'aller dans `NSI`.
   `../boulot/NSI`
3) Le dossier courant est maintenant `NSI`. Dans quel dossier va se trouver le fichier `memo.txt` après la commande :
   `touch ../../memo.txt` dans le dossier `samia`
4) Toujours depuis le dossier `NSI`, quelle commande permet de créer un dossier `Python` dans le dossier `boulot` ?
   `mkdir ../Python`

Exercice 3 : (8,5pt) Les 3 parties de cet exercice sont indépendantes.

Partie A

Voici une capture d'un terminal Linux :

```
alovelace@cerebro:~$ ls -l
total 21297
-rw-r--r--  8 root       www-data   1257  janv. 29 2019  concat.pls
drwxrwxr-x  2 alovelace  alovelace  4096  févr. 18 2023  graph
-rwxrwxr--  4 root       adminsys    344  juin  13 2021  todo.txt
-rwxrw-r--  1 alovelace  www-data    118  ocot. 05 2024  verify.sh
```

On rappelle que dans la partie gauche, le symbole précédent les 9 symboles de permissions est soit `-` pour un fichier, `d` pour un dossier et `l` pour un lien.

1) Quels sont les permissions dont disposent le propriétaire (`user`), le groupe (`group`) et les autres (`other`) sur le fichier `verify.sh` :
   Propriétaire : lire, écrire et exécuter
   Groupe : lire et écrire
   Autres : lire
2) Qui a le droit de supprimer le fichier `concat.pls` (il faut avoir le droit d'écriture) et avec quelle commande ?
   Solution : Seul `root` peut le supprimer en faisant `rm concat.pls`.
3) Donner la représentation à l'aide des 9 symboles des droits du fichier `todo.txt` après avoir exécuté la commande `chmod u-x,g-wx todo.txt`.
   Solution : `rw-r--r--`.

On s'intéresse maintenant à la représentation octale des droits dont les valeurs sont résumées ci-contre.

	user			group			other		
droits	r	w	x	r	w	x	r	w	x
valeur	4	2	1	4	2	1	4	2	1

4) Quel est le code octal de `rwxr-xr--` ? **Solution :** 754.

Partie B

On souhaite écrire une fonction permettant de passer de la représentation à l'aide des 9 symboles à la représentation octale.
La fonction `bin_vers_oct` prend en paramètre un texte composé de 3 chiffres binaires et qui renvoie la valeur octale correspondante.

```
>>> bin_vers_oct('101')
5
>>> bin_vers_oct('011')
3
```

5) Compléter le code de la fonction `bin_vers_oct`.

```
def bin_vers_oct(binaire):
    octal = 0
    if binaire[0] == '1':
        octal += 4
    if binaire[1] == ...:
        octal += 2
    if binaire[2] == '1':
        octal += 1
    return octal
```

On considère la fonction `mystere` qui prend en paramètre une chaîne de 9 caractères et qui renvoie une chaîne de caractères :

```
def mystere(chaine):
    resultat = ''
    for c in chaine:
        if c == '-':
            resultat = resultat + '0'
        else:
            resultat = resultat + '1'
    return resultat
```

6) Déterminer ce que renvoie `mystere('rw-r-xr--')`. **Solution :** `110101100`.

On suppose qu'on dispose d'une fonction Python `symb_vers_bin` qui prend en paramètre une chaîne de 9 caractères correspondant à la notation symbolique des permissions et qui renvoie une chaîne de caractères correspondant à une version binaire de ces permissions.
La fonction `symb_vers_oct` prend en paramètre une chaîne de 9 caractères correspondant à la notation symbolique des permissions et qui renvoie la représentation octale, sous forme d'une chaîne de 3 caractères, de ces permissions. On rappelle que `str(5)` renvoie `'5'`.

```
>>> symb_vers_oct('rwxrw-r--')
'764'
>>> symb_vers_oct('rw-r--r--')
'644'
```

7) Compléter le code de la fonction `symb_vers_oct`. Vous pouvez rajouter plusieurs lignes

```
def symb_vers_oct(chaine):
    repr_bin = symb_vers_bin(chaine)
    user = repr_bin[0] + repr_bin[1] + repr_bin[2]
    group = repr_bin[3] + repr_bin[4] + repr_bin[5]
    other = repr_bin[6] + repr_bin[7] + repr_bin[8]
    u = bin_vers_oct(user)
    g = bin_vers_oct(group)
    o = bin_vers_oct(other)
    return str(u) + str(g) + str(o)
```

Partie C

On représente les fichiers à l'aide d'un dictionnaire. Par exemple le fichier `verify.sh` est représenté par le fichier :

```
verif = {'nom': 'verify.sh', 'proprio': 'alovelace', 'groupe': 'www-data',
         'permissions': 'rwxrw-r--'}
```

La fonction `a_le_droit_exec` prend en paramètre un dictionnaire `fichier` et deux chaînes de caractères `utilisateur` et `groupe`, et qui renvoie un booléen qui vaut `True` si l'utilisateur appartenant au groupe indiqué a le droit d'exécuter le fichier et `False` sinon.

```
>>> a_le_droit_exec(verif, "alovelace", "alovelace")
True
>>> a_le_droit_exec(verif, "samia", "www-data")
False
```

Pour rappel, un utilisateur a le droit d'exécuter un fichier si :

- il est le propriétaire et le propriétaire a le droit d'exéction :
- il n'est pas le propriétaire mais appartient au même groupe que le fichier, et le groupe a le droit d'exécution ;

- il n'est ni propriétaire, ni membre du groupe, et les autres ont le droit d'exécution.

8) Compléter le code de la fonction `a_le_droit_exec`.

```
def a_le_droit_exec(fichier, utilisateur, groupe):
    droits = fichier["permissions"]
    if utilisateur == fichier["proprio"] and droits[2] == 'x':
        return True
    elif groupe == fichier["groupe"] and droits[5] == 'x':
        return True
    elif droits[8] == 'x':
        return True
    return False
```

Exercice 4 : (7pt) *Cet exercice porte sur les structures de données (dictionnaires)*
Afin d'organiser les dossiers et les fichiers sur un disque dur, une structure arborescente est utilisée. Les fichiers sont dans des dossiers qui sont eux-mêmes dans d'autres dossiers, etc. Dans une arborescence, chaque dossier peut contenir des fichiers et des dossiers, qui sont identifiés par leur nom. Le contenu d'un dossier est modélisé par la structure de données **dictionnaire**. Les clés de ce dictionnaire sont des chaînes de caractères donnant le nom des fichiers et des dossiers contenus.

Le dossier appelé `Téléchargements` contient deux fichiers `rapport.pdf` et `jingle.mp3`, et un dossier `Images` contenant simplement le fichier `logo.png`. Il est représenté ci-contre.
Ce dossier `Téléchargements` est modélisé en Python par le dictionnaire suivant :

```
Téléchargements
├── Images
│   └── logo.png
├── jingle.mp3
└── rapport.pdf
```

```
{"Images": {"logo.png": 36}, "rapport.pdf": 450, "jingle.mp3": 4800}
```

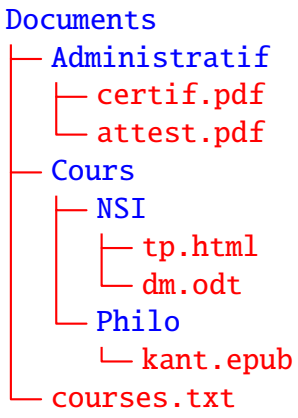
Les valeurs numériques sont exprimées en ko (kilo-octets). `"logo.png": 36` signifie que le fichier `logo.png` occupe un espace mémoire de 36 ko sur le disque dur.
On rappelle, ci-dessous, quelques commandes sur l'utilisation d'un dictionnaire :
- `dico = {}` ou `dico = dict()` crée un dictionnaire vide appelé `dico`,
- `dico[cle] = contenu` met la valeur contenu pour la clé `cle` dans le dictionnaire `dico`,
- `dico[cle]` renvoie la valeur associée à la clé `cle` dans le dictionnaire `dico`,
- `cle in dico` renvoie un booléen indiquant si la clé `cle` est présente dans le dictionnaire `dico`,
- `for cle in dico:` parcourt l'ensemble des clés du dictionnaire.

L'**adresse** d'un fichier ou d'un dossier correspond au nom de tous les dossiers à parcourir depuis la racine afin d'accéder au fichier ou au dossier. Cette adresse est modélisée en Python par la liste des noms de dossier à parcourir pour y accéder.

Exemple : L'adresse du dossier : `/home/pierre/Documents/` est modélisée par la liste `["home", "pierre", "Documents"]`.

1) Dessiner l'arbre donné par le dictionnaire suivant, qui correspond au dossier `Documents` donné ci-dessous. On ne demande pas d'indiquer les tailles des fichiersdans cet arbre.

```
Documents = {
  "Administratif":{
    "certif.pdf ": 1500,
    "attest.pdf ": 850
  },
  "Cours": {
    "NSI": {
      "tp.html": 60,
      "dm.odt": 345
    },
    "Philo": {"kant.epub": 2600}
  },
  "courses.txt ": 24
}
```



2) a) On reprend la variable `Documents` de la question précédente. On considère les instructions suivantes :

```
>>> dossier = Documents
>>> dossier = dossier["Cours"]
>>> dossier = dossier["NSI"]
```

Compléter le résultat de l'évaluation de la variable `dossier` après l'exécution des instructions ci-dessus :

```
>>> dossier
{'tp.html': 60, 'dm.odt': 345}
```

b) On donne la fonction `parcourir` suivante qui prend en paramètres un dossier racine et une liste représentant une adresse, et qui renvoie le contenu du dossier cible correspondant à l'adresse.

```
>>> parcourir(Documents, ["Cours", "Philo"])
{'kant.epub': 2600}
```

Compléter le code de la fonction.
Indication : Il faut vous inspirer de la question a) où la variable `dossier` prend successivement pour valeur les dossiers visités au cours de l'exploration.

```
def parcourir(racine, adr):
    dossier = racine
    for nom_dossier in adr:
        dossier = dossier[nom_dossier]
    return dossier
```

c) Soit la fonction suivante :

```
def afficher(racine, adr, nom_fichier):
    dossier = parcourir(racine, adr)
    print(dossier[nom_fichier])
```

Qu'affiche l'instruction `afficher(Documents, ["Cours", "NSI"], "tp.html")` sachant que la variable `Documents` contient le dictionnaire de la question 1 ?

60

3) a) La fonction `ajouter_fichier(racine, adr, nom_fichier, taille)` suivante ajoute au dictionnaire `racine`, à l'adresse `adr`, la clé `nom_fichier` associée à la valeur `taille`.

```
>>> ajouter_fichier(Documents, ["Cours", "NSI"], "ds3.py", 23)
>>> parcourir(Documents, ["Cours", "NSI"])
{"tp.html": 60, "dm.odt": 345, "ds3.py": 23}
```

Compléter le code de la fonction. Il faut utiliser `parcourir`.

```
def ajouter_fichier(racine, adr, nom_fichier, taille):
    dossier =  parcourir(racine, adr)
    dossier[nom_fichier] = taille
```

b) Écrire une fonction `ajouter_dossier(racine, adr, nom_dossier)` pour créer un dictionnaire représentant un dossier vide appelé `nom_dossier` dans le dictionnaire racine à l'adresse `adr`. On utilisera `parcourir`.

```
>>> ajouter_dossier(Documents, ["Administratif"], "Contrats")
>>> parcourir(Documents, ["Administratif"])
{"certif.pdf ": 1500, "attest.pdf ": 850, "Contrats": {}}
```

```
def ajouter_dossier(racine, adr, nom_dossier):
    dossier = parcourir(racine, adr)
    dossier[mon_dossier] = dict()
```

4) Écrire une fonction `calcule_taille` qui prend en paramètre un dictionnaire `dossier` modélisant le contenu du répertoire `dossier` et qui renvoie le total de l'espace mémoire occupé par les fichiers contenus dans le dossier. On considère que le répertoire `dossier` ne contient que des fichiers et **aucun sous-dossier**.

```
>>> calcule_taille({"tp.html": 60, "dm.odt": 345, "ds3.py": 23})
428
```

```
def calcule_taille(dossier):
    taille = 0
    for fichier in dossier:
        taille = taille + dossier[fichier]
    return taille
```

Lycée Les 3 Sources — **Bourg-Lès-Valence**
NSI 1re — Année **2025-26**

Devoir surveillé n°3 – correction

Exercice 1 : (1.5pt) Entourer la bonne réponse à chaque question.

1) Si `Décalage` s'écrit "`44 c3 a9 63 61 6c 61 67 65`" en UTF-8 en hexadécimal, quelle peut être l'écriture de `Glacé` ?
   a) `67 6c 61 63 c3 a9`
   b) `47 6c 61 63 a9 c3`
   c) `67 6c 61 63 a9 c3`
   d) `47 6c 61 63 c3 a9`

2) Déterminer la valeur de `score` après les instructions ci-contre :

```
>>> score = {}
>>> score['Alice'] = 4
>>> score['Bob'] = 7
>>> score['Alice'] = 15
```

   a) `{'Alice': 4, 'Bob': 7, 'Alice': 15}`
   b) `{'Alice': 19, 'Bob': 7}`
   c) `{'Alice': 15, 'Bob': 7}`
   d) `[4, 7, 15]`

3) On considère le code suivant :

```
def menagerie(animaux):
    reponse  = dict()
    for animal in animaux:
        if animal in reponse:
            reponse[animal] = reponse[animal] + 1
        else:
            reponse[animal] = 1
    return reponse

enclos = ['poule', 'poule', 'coq', 'poussin', 'poule', 'poussin']
```

Quelle est la valeur de `menagerie(enclos)` ?

   a) `{'poule': 1, 'coq': 1, 'poussin': 1}`
   b) `{'poule': 3, 'coq': 1, 'poussin': 2}`
   c) `{3: 'poule', 1: 'coq', 2: 'poussin'}`
   d) `{'poule': 1, 'poule': 1, 'coq': 1, 'poussin': 1, 'poule': 1, 'poussin': 1}`

Exercice 2 : (2pt) On donne ci-contre l'architecture des dossiers de Samia sur l'ordinateur de la maison. Les fichiers ne sont pas affichés.

```
/
└── home
    └── samia
        ├── boulot
        │   ├── euro
        │   ├── maths
        │   └── NSI
        └── perso
            ├── photos
            └── vidéos
```

1) Donnez le chemin absolu du dossier `NSI`.
   `/home/samia/boulot/NSI`
2) Le dossier courant dans le terminal est `perso`. Donnez le chemin relatif permettant d'aller dans `NSI`.
   `../boulot/NSI`
3) Le dossier courant est maintenant `NSI`. Dans quel dossier va se trouver le fichier `memo.txt` après la commande :
   `touch ../../memo.txt` dans le dossier `samia`
4) Toujours depuis le dossier `NSI`, quelle commande permet de créer un dossier `Python` dans le dossier `boulot` ?
   `mkdir ../Python`

Exercice 3 : (8,5pt) Les 3 parties de cet exercice sont indépendantes.

Partie A

Voici une capture d'un terminal Linux :

```
alovelace@cerebro:~$ ls -l
total 21297
-rw-r--r--  8 root       www-data   1257  janv. 29 2019  concat.pls
drwxrwxr-x  2 alovelace  alovelace  4096  févr. 18 2023  graph
-rwxrwxr--  4 root       adminsys    344  juin  13 2021  todo.txt
-rwxrw-r--  1 alovelace  www-data    118  ocot. 05 2024  verify.sh
```

On rappelle que dans la partie gauche, le symbole précédent les 9 symboles de permissions est soit `-` pour un fichier, `d` pour un dossier et `l` pour un lien.

1) Quels sont les permissions dont disposent le propriétaire (`user`), le groupe (`group`) et les autres (`other`) sur le fichier `verify.sh` :
   Propriétaire : lire, écrire et exécuter
   Groupe : lire et écrire
   Autres : lire
2) Qui a le droit de supprimer le fichier `concat.pls` (il faut avoir le droit d'écriture) et avec quelle commande ?
   Solution : Seul `root` peut le supprimer en faisant `rm concat.pls`.
3) Donner la représentation à l'aide des 9 symboles des droits du fichier `todo.txt` après avoir exécuté la commande `chmod u-x,g-wx todo.txt`.
   Solution : `rw-r--r--`.

On s'intéresse maintenant à la représentation octale des droits dont les valeurs sont résumées ci-contre.

	user			group			other		
droits	r	w	x	r	w	x	r	w	x
valeur	4	2	1	4	2	1	4	2	1

4) Quel est le code octal de `rwxr--r-x` ? **Solution :** 745.

Partie B

On souhaite écrire une fonction permettant de passer de la représentation à l'aide des 9 symboles à la représentation octale.
La fonction `bin_vers_oct` prend en paramètre un texte composé de 3 chiffres binaires et qui renvoie la valeur octale correspondante.

```
>>> bin_vers_oct('101')
5
>>> bin_vers_oct('011')
3
```

5) Compléter le code de la fonction `bin_vers_oct`.

```
def bin_vers_oct(binaire):
    octal = 0
    if binaire[0] == '1':
        octal += 4
    if binaire[1] == ...:
        octal += 2
    if binaire[2] == '1':
        octal += 1
    return octal
```

On considère la fonction `mystere` qui prend en paramètre une chaîne de 9 caractères et qui renvoie une chaîne de caractères :

```
def mystere(chaine):
    resultat = ''
    for c in chaine:
        if c == '-':
            resultat = resultat + '0'
        else:
            resultat = resultat + '1'
    return resultat
```

6) Déterminer ce que renvoie `mystere('r-xrw-r--')`. **Solution :** `101110100`.

On suppose qu'on dispose d'une fonction Python `symb_vers_bin` qui prend en paramètre une chaîne de 9 caractères correspondant à la notation symbolique des permissions et qui renvoie une chaîne de caractères correspondant à une version binaire de ces permissions.
La fonction `symb_vers_oct` prend en paramètre une chaîne de 9 caractères correspondant à la notation symbolique des permissions et qui renvoie la représentation octale, sous forme d'une chaîne de 3 caractères, de ces permissions. On rappelle que `str(5)` renvoie `'5'`.

```
>>> symb_vers_oct('rwxrw-r--')
'764'
>>> symb_vers_oct('rw-r--r--')
'644'
```

7) Compléter le code de la fonction `symb_vers_oct`. Vous pouvez rajouter plusieurs lignes

```
def symb_vers_oct(chaine):
    repr_bin = symb_vers_bin(chaine)
    user = repr_bin[0] + repr_bin[1] + repr_bin[2]
    group = repr_bin[3] + repr_bin[4] + repr_bin[5]
    other = repr_bin[6] + repr_bin[7] + repr_bin[8]
    u = bin_vers_oct(user)
    g = bin_vers_oct(group)
    o = bin_vers_oct(other)
    return str(u) + str(g) + str(o)
```

Partie C

On représente les fichiers à l'aide d'un dictionnaire. Par exemple le fichier `verify.sh` est représenté par le fichier :

```
verif = {'nom': 'verify.sh', 'proprio': 'alovelace', 'groupe': 'www-data',
         'permissions': 'rwxrw-r--'}
```

La fonction `a_le_droit_exec` prend en paramètre un dictionnaire `fichier` et deux chaînes de caractères `utilisateur` et `groupe`, et qui renvoie un booléen qui vaut `True` si l'utilisateur appartenant au groupe indiqué a le droit d'exécuter le fichier et `False` sinon.

```
>>> a_le_droit_exec(verif, "alovelace", "alovelace")
True
>>> a_le_droit_exec(verif, "samia", "www-data")
False
```

Pour rappel, un utilisateur a le droit d'exécuter un fichier si :

- il est le propriétaire et le propriétaire a le droit d'exéction :
- il n'est pas le propriétaire mais appartient au même groupe que le fichier, et le groupe a le droit d'exécution ;

- il n'est ni propriétaire, ni membre du groupe, et les autres ont le droit d'exécution.

8) Compléter le code de la fonction `a_le_droit_exec`.

```
def a_le_droit_exec(fichier, utilisateur, groupe):
    droits = fichier["permissions"]
    if utilisateur == fichier["proprio"] and droits[2] == 'x':
        return True
    elif groupe == fichier["groupe"] and droits[5] == 'x':
        return True
    elif droits[8] == 'x':
        return True
    return False
```

Exercice 4 : (7pt) *Cet exercice porte sur les structures de données (dictionnaires)*
Afin d'organiser les dossiers et les fichiers sur un disque dur, une structure arborescente est utilisée. Les fichiers sont dans des dossiers qui sont eux-mêmes dans d'autres dossiers, etc. Dans une arborescence, chaque dossier peut contenir des fichiers et des dossiers, qui sont identifiés par leur nom. Le contenu d'un dossier est modélisé par la structure de données **dictionnaire**. Les clés de ce dictionnaire sont des chaînes de caractères donnant le nom des fichiers et des dossiers contenus.

Le dossier appelé `Téléchargements` contient deux fichiers `rapport.pdf` et `jingle.mp3`, et un dossier `Images` contenant simplement le fichier `logo.png`. Il est représenté ci-contre.
Ce dossier `Téléchargements` est modélisé en Python par le dictionnaire suivant :

```
Téléchargements
├── Images
│   └── logo.png
├── jingle.mp3
└── rapport.pdf
```

```
{"Images": {"logo.png": 36}, "rapport.pdf": 450, "jingle.mp3": 4800}
```

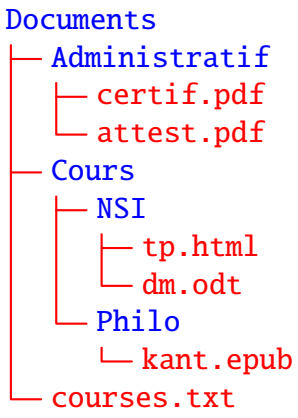
Les valeurs numériques sont exprimées en ko (kilo-octets). `"logo.png": 36` signifie que le fichier `logo.png` occupe un espace mémoire de 36 ko sur le disque dur.
On rappelle, ci-dessous, quelques commandes sur l'utilisation d'un dictionnaire :
- `dico = {}` ou `dico = dict()` crée un dictionnaire vide appelé `dico`,
- `dico[cle] = contenu` met la valeur contenu pour la clé `cle` dans le dictionnaire `dico`,
- `dico[cle]` renvoie la valeur associée à la clé `cle` dans le dictionnaire `dico`,
- `cle in dico` renvoie un booléen indiquant si la clé `cle` est présente dans le dictionnaire `dico`,
- `for cle in dico:` parcourt l'ensemble des clés du dictionnaire.

L'**adresse** d'un fichier ou d'un dossier correspond au nom de tous les dossiers à parcourir depuis la racine afin d'accéder au fichier ou au dossier. Cette adresse est modélisée en Python par la liste des noms de dossier à parcourir pour y accéder.

Exemple : L'adresse du dossier : `/home/pierre/Documents/` est modélisée par la liste `["home", "pierre", "Documents"]`.

1) Dessiner l'arbre donné par le dictionnaire suivant, qui correspond au dossier `Documents` donné ci-dessous. On ne demande pas d'indiquer les tailles des fichiersdans cet arbre.

```
Documents = {
  "Administratif":{
    "certif.pdf ": 1500,
    "attest.pdf ": 850
  },
  "Cours": {
    "NSI": {
      "tp.html": 60,
      "dm.odt": 345
    },
    "Philo": {"kant.epub": 2600}
  },
  "courses.txt ": 24
}
```



2) a) On reprend la variable `Documents` de la question précédente. On considère les instructions suivantes :

```
>>> dossier = Documents
>>> dossier = dossier["Cours"]
>>> dossier = dossier["NSI"]
```

Compléter le résultat de l'évaluation de la variable `dossier` après l'exécution des instructions ci-dessus :

```
>>> dossier
{'tp.html': 60, 'dm.odt': 345}
```

b) On donne la fonction `parcourir` suivante qui prend en paramètres un dossier racine et une liste représentant une adresse, et qui renvoie le contenu du dossier cible correspondant à l'adresse.

```
>>> parcourir(Documents, ["Cours", "Philo"])
{'kant.epub': 2600}
```

Compléter le code de la fonction.
Indication : Il faut vous inspirer de la question a) où la variable `dossier` prend successivement pour valeur les dossiers visités au cours de l'exploration.

```
def parcourir(racine, adr):
    dossier = racine
    for nom_dossier in adr:
        dossier = dossier[nom_dossier]
    return dossier
```

c) Soit la fonction suivante :

```
def afficher(racine, adr, nom_fichier):
    dossier = parcourir(racine, adr)
    print(dossier[nom_fichier])
```

Qu'affiche l'instruction `afficher(Documents, ["Cours", "NSI"], "tp.html")` sachant que la variable `Documents` contient le dictionnaire de la question 1 ?

60

3) a) La fonction `ajouter_fichier(racine, adr, nom_fichier, taille)` suivante ajoute au dictionnaire `racine`, à l'adresse `adr`, la clé `nom_fichier` associée à la valeur `taille`.

```
>>> ajouter_fichier(Documents, ["Cours", "NSI"], "ds3.py", 23)
>>> parcourir(Documents, ["Cours", "NSI"])
{"tp.html": 60, "dm.odt": 345, "ds3.py": 23}
```

Compléter le code de la fonction. Il faut utiliser `parcourir`.

```
def ajouter_fichier(racine, adr, nom_fichier, taille):
    dossier =  parcourir(racine, adr)
    dossier[nom_fichier] = taille
```

b) Écrire une fonction `ajouter_dossier(racine, adr, nom_dossier)` pour créer un dictionnaire représentant un dossier vide appelé `nom_dossier` dans le dictionnaire racine à l'adresse `adr`. On utilisera `parcourir`.

```
>>> ajouter_dossier(Documents, ["Administratif"], "Contrats")
>>> parcourir(Documents, ["Administratif"])
{"certif.pdf ": 1500, "attest.pdf ": 850, "Contrats": {}}
```

```
def ajouter_dossier(racine, adr, nom_dossier):
    dossier = parcourir(racine, adr)
    dossier[mon_dossier] = dict()
```

4) Écrire une fonction `calcule_taille` qui prend en paramètre un dictionnaire `dossier` modélisant le contenu du répertoire `dossier` et qui renvoie le total de l'espace mémoire occupé par les fichiers contenus dans le dossier. On considère que le répertoire `dossier` ne contient que des fichiers et **aucun sous-dossier**.

```
>>> calcule_taille({"tp.html": 60, "dm.odt": 345, "ds3.py": 23})
428
```

```
def calcule_taille(dossier):
    taille = 0
    for fichier in dossier:
        taille = taille + dossier[fichier]
    return taille
```