# Labo d'introduction à l'informatique

pour les mathématiques

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

# Semaine 9

*Écriture et lecture de fichiers*

# Un nouveau problème

- J'ai écrit un code pour trouver de grands nombres premiers.
- Le lundi, après 8 heures de calcul, j'arrête l'ordinateur.
- Comment reprendre là où j'en étais le mardi matin ?
- Solution : **écrire** les résultats sur une mémoire **persistante**, afin de pouvoir les **lire** une autre fois.
- Trois avantages :
  - Sauvegarde de données
  - Performance (pas besoin de tout recalculer)
  - Séparation entre données et logique du code

# Interagir avec d'autres fichiers

En programmation (scientifique ou pas), il est courant de séparer les tâches.

Par exemple :

- Regrouper des données brutes dans un fichier
- Décrire la logique du traitement des données dans un autre fichier (script)
- Écrire les résultats dans un autre fichier encore

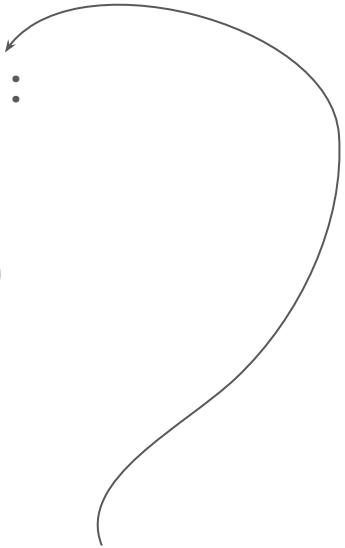
# Ouvrir un fichier en Python

- Nous montrons ici une façon d'ouvrir un fichier. Il en existe d'autres !
- Cette façon utilise deux nouveaux mots-clé (`with` et `as`) ainsi qu'une nouvelle fonction (`open`).
- L'utilisation de with permet de s'assurer que l'on ferme correctement le fichier quoi qu'il arrive durant l'exécution du script.
- Quand on ouvre un fichier, on indique si on veut le lire ou l'écrire (d'autres options existent, non traitées ici) : "r" → "**r**ead" ou "w" → "**w**rite".
- Exemple d'ouverture en lecture :

```
with open("mon_fichier.txt", "r") as f:
    ... #bloc de code où je décris ma logique
```

# Lire un fichier | **`read`**

```
with open("mon_fichier.txt", "r") as f:
    contenu = f.read() #str
    print("Voici ce qu'il y a dedans:")
    print(contenu )
```



Variable qui représentant le fichier

# Lire les lignes d'un fichier | **`readlines`**

```
with open("mon_fichier.txt", "r") as f:
    lignes = f.readlines() #liste de str
    print("Voici ce qu'il y a dedans:")
    print(lignes)
```

# Écrire un fichier | `write`

```
with open("mon_fichier.txt", "w") as f:
    f.write('Voici la première ligne\n')
```

Attention :

- `write` écrit par dessus l'ancien fichier s'il existe
- "\n" est le caractère spécial de fin de ligne (saut de ligne)

# Écrire un fichier | **`write` en mode append**

Ouverture en mode append ("a") pour ajouter à la suite du contenu :

```
with open("mon_fichier.txt", "a") as f:
    f.write('Et voici la seconde ligne\n')
```

# Lecture/écriture | **Commentaires finaux**

- On peut préciser l'**encodage** utilisé si besoin :
  `open("mon_fichier", "r", encoding="utf-8")`

- La lecture et l'écriture s'effectuent **relativement** à l'endroit du disque d'où le script est appelé.

# Partie 2 - Fichiers, dossiers et chemins

# Qu’écrit-on sur la mémoire auxiliaire de l’ordinateur ?

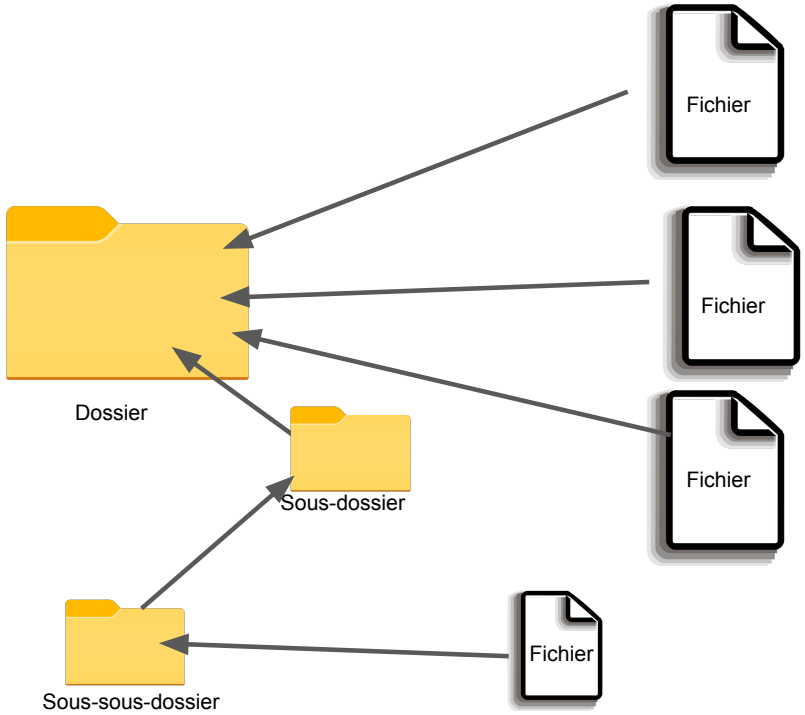
Toutes les informations (ou "**données**") que l’on veut garder :

- Photos
- Documents
- Musiques
- Jeux-vidéos
- …

Ces données sont organisées sous la forme de **fichiers** informatiques.

# Fichiers et dossiers | **Différence entre dossier et fichier**

- Un fichier ne peut pas contenir d'autres dossiers ou fichiers.
- Un dossier peut contenir d'autres dossiers ou fichiers.



# Qu’y a-t-il dans un fichier ?

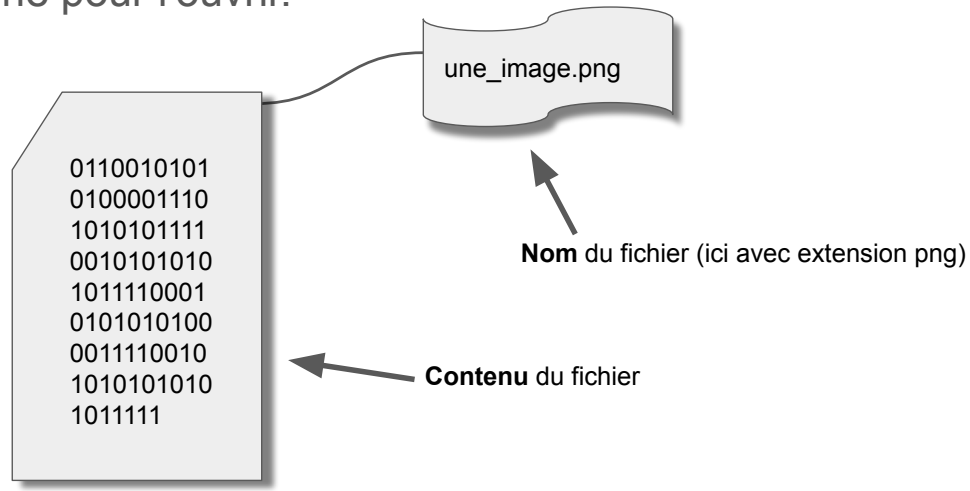
Comme on l’a vu, l’ordinateur traite automatiquement des nombres. Un fichier est constitué d’une longue suite de nombres qui servent à **coder l’information**.

Cette suite de nombres peut être très longue (par exemple pour stocker une vidéo) ou très courte (par exemple un fichier texte avec un seul mot).

Une image ne s’interprète pas de la même façon qu’une musique. Ce sont les **logiciels** de l’ordinateur qui décident comment **interpréter** la suite de nombres.

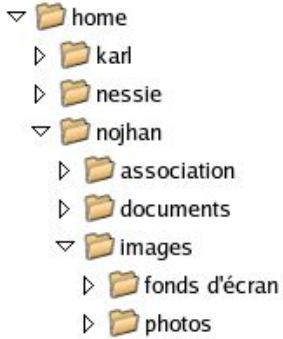
# Fichier informatique

Un fichier possède un **nom**. Souvent, la fin du nom termine par un point suivi de quelques lettre : c'est ce que l'on nomme l'**extension de fichier**. Néanmoins, ce n'est qu'une convention ! Cela ne garantit rien à propos du **contenu** du fichier, mais aide juste à choisir un logiciel approprié pour l'ouvrir.



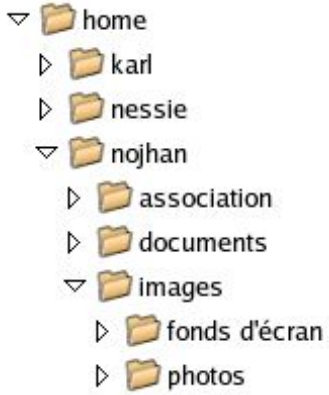
# Fichiers et dossiers

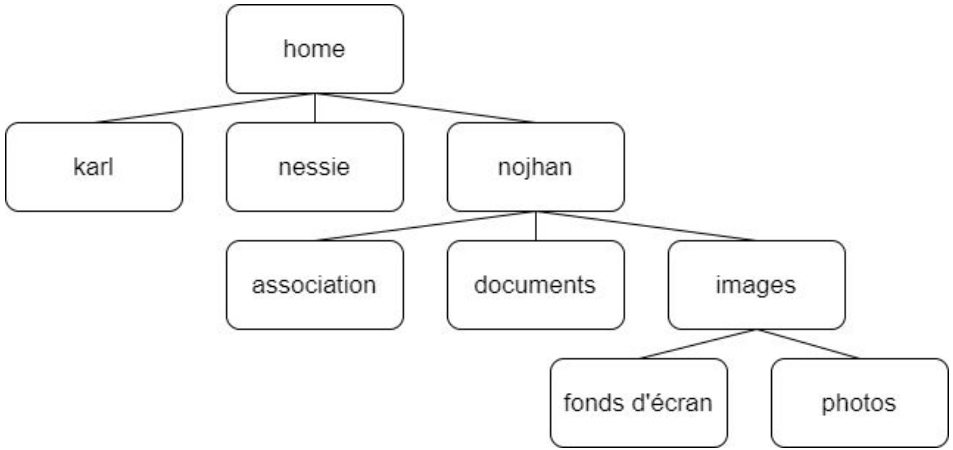
- Les OS permettent d'organiser les fichiers dans des **dossiers**.
- Un dossier peut contenir des fichiers ou être vide.
- Un dossier peut également contenir d'autres dossiers.
- Cela permet d'établir une **hiérarchie arborescente** entre les dossiers/fichiers.



# Hiérarchie de dossiers et fichiers

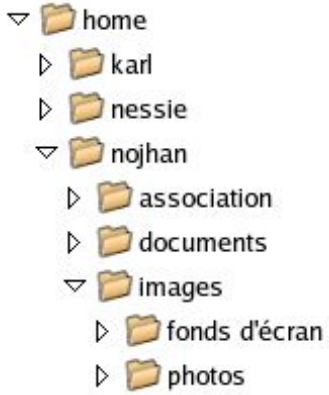
Sur cette image, le dossier *home* contient trois autres dossiers nommés *karl*, *nessie* et *nojhan*. On voit que nojhan contient lui même trois autres dossiers, etc.
La hiérarchie entre ces dossiers peut être résumée comme un arbre.

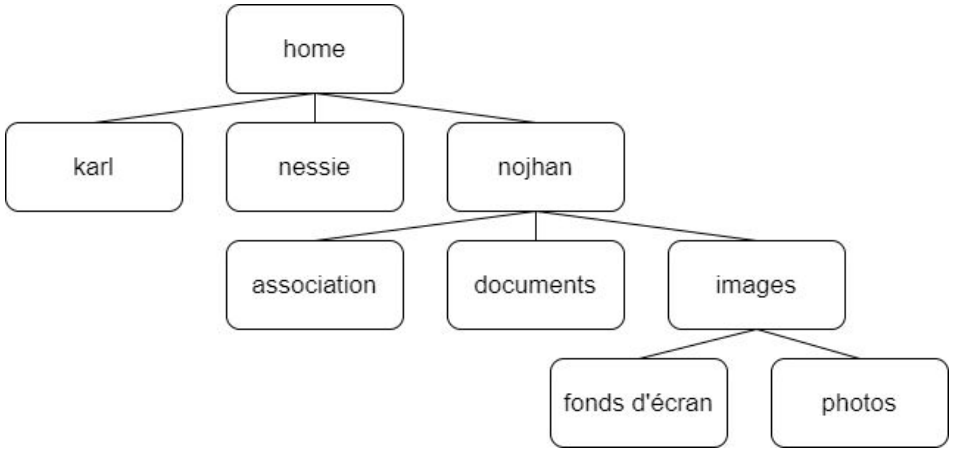


# Hiérarchie de dossiers et fichiers

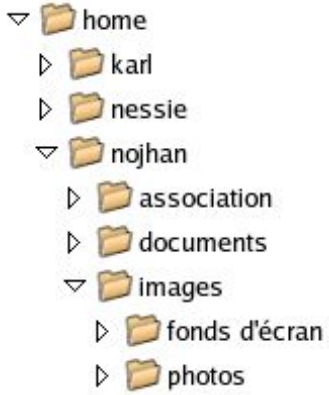
On dit que *nojhan* est le dossier **parent** de *association*, *documents* et *images*. *Image* est un dossier **enfant** de *nojhan*. Ici, *home* est la **racine** de la hiérarchie.

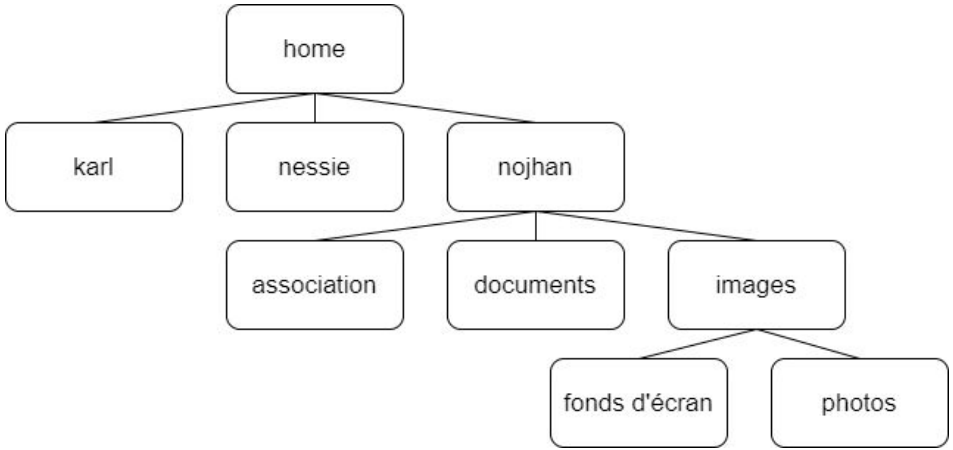


# Chemins de dossiers et fichiers

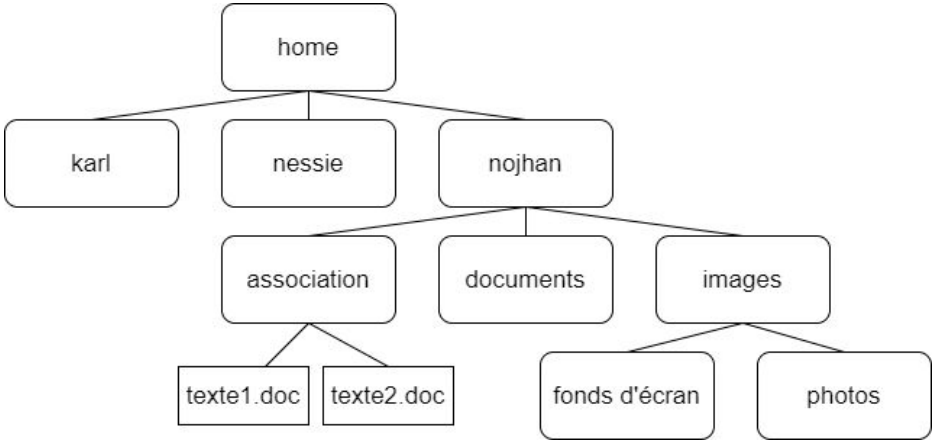
Le **chemin** menant au dossier association est : *`home/nojhan/association`*



# Chemins de dossiers et fichiers

Imaginons maintenant que le dossier association contienne deux fichiers nommés *texte1.doc* et *texte2.doc*



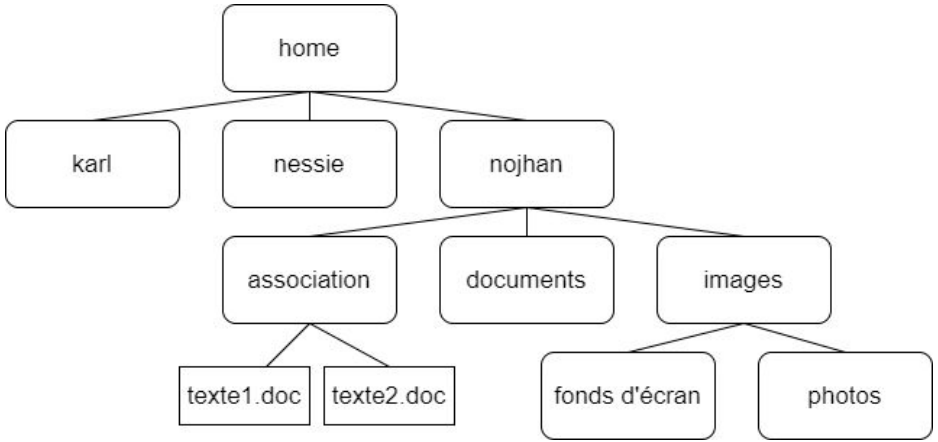
# Chemins de dossiers et fichiers

Quel est le chemin complet du fichier *texte2.doc* ?

home/nojhan/association/texte2.doc

Dans quel dossier le fichier *texte2.doc* se trouve-t-il ?

Il se trouve dans le dossier *association*.

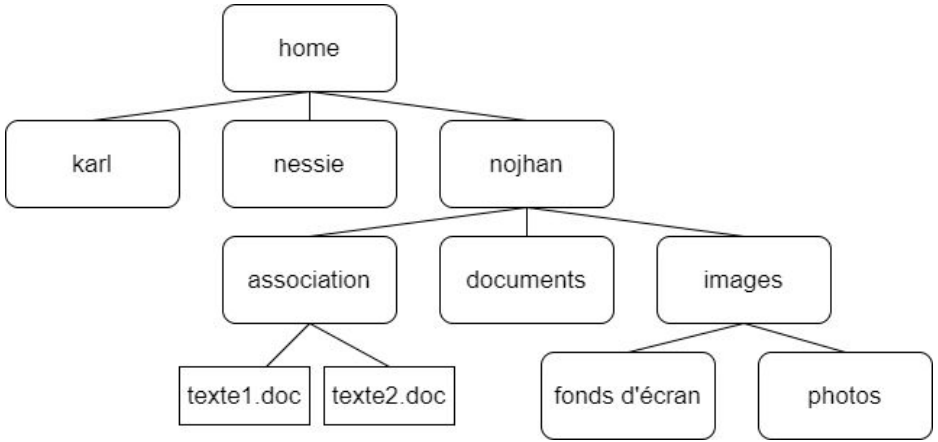


# Chemins relatifs

Le double point ".." sert à désigner le dossier parent.

Par exemple, le dossier *nojhan* est le parent du parent de *texte2.doc*. **Relativement** à *texte2.doc*, on le désigne donc par `../../`

Comment indiquer le chemin du dossier photos relativement au fichier *texte2.doc* ?
Réponse : `../../images/photos`



# Chemins absolus

Comment indiquer le chemin du dossier photos relativement au fichier *texte2.doc* ?
Réponse : `../../images/photos`

Le même chemin en absolu :
`home/nohjan/association/texte2.doc`