ENG-209

Data science pour ingénieurs avec Python

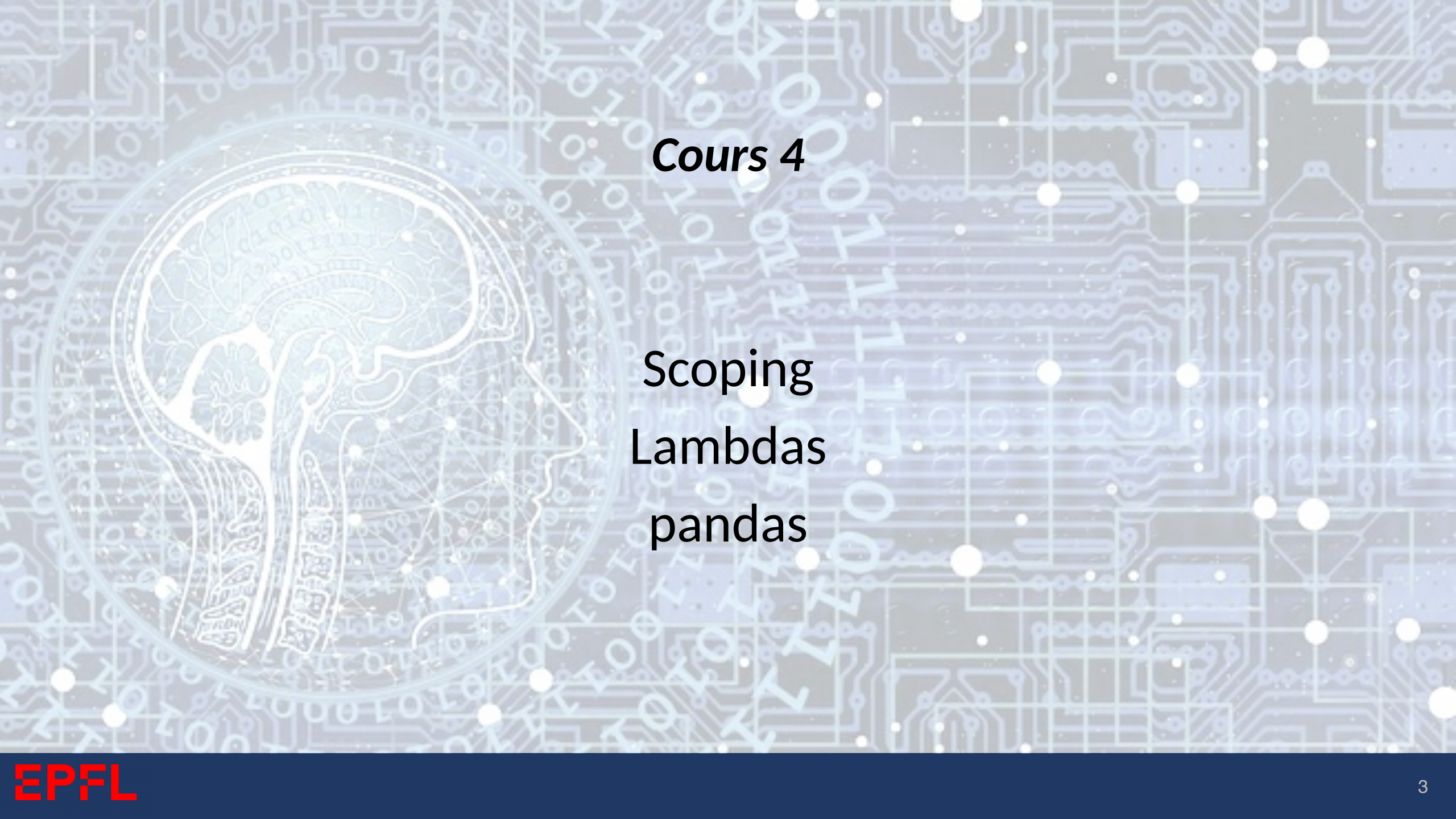
Cours 4:
Scopes; Lambdas; pandas

Jean-Philippe Pellet, Eric Bouillet, Olivier Verscheure

6 octobre 2025

Organisation

- **13.10.2025: Midterm, 10h15, INF3, 120 minutes** (160 si temps suppl.)
  - Partie QCM (questions plutôt théoriques)
  - Partie programmation (compléter/corriger/écrire du code dans un Jupyter notebook, à soumettre par Moodle) — proche de ce que l'on fait comme style dans les exercices, sans pandas
  - Travail sur machine virtuelle uniquement depuis poste INF3
    - *Assurez-vous aujourd'hui que tout fonctionne sur votre VM!*
  - Matériel autorisé: résumé personnel au format d'une feuille A4 recto-verso
    - ***Pandas** ne fait **pas** partie de l'évaluation (**numpy si**)*



Cours 4

Scoping

Lambdas

pandas

Cours 4

Scoping

Lambdas

pandas

Déclarations imbriquées

En règle générale, un bloc de code a accès aux déclaration de ses «contextes parents»

```
friendships: dict[str, set[str]] = {}

def add_friends(name1: str, name2: str):
    def add(name1: str, name2: str):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}
    add(name1, name2)
    add(name2, name1)
```

Ici, une fonction a accès sans problème à friendships*, même si cette variable n'a pas été déclarée/affectée dans cette fonction et qu'elle n'a pas été passée comme paramètre*

On peut déclarer une fonction dans une autre fonction, ce qui est très pratique pour, par exemple, rapidement et localement extraire et nommer un bout de code sans «polluer» le reste du code avec une fonction de premier niveau en plus

Scoping et affectations

```
application = "VS Code"

def read_preferences():
    global application
    application = "PyCharm"

read_preferences()


print(f"Open {application} and solve the exercise")
```

S'il s'agit d'affecter depuis une fonction une variable extérieure à cette fonction, il faut l'indiquer explicitement.

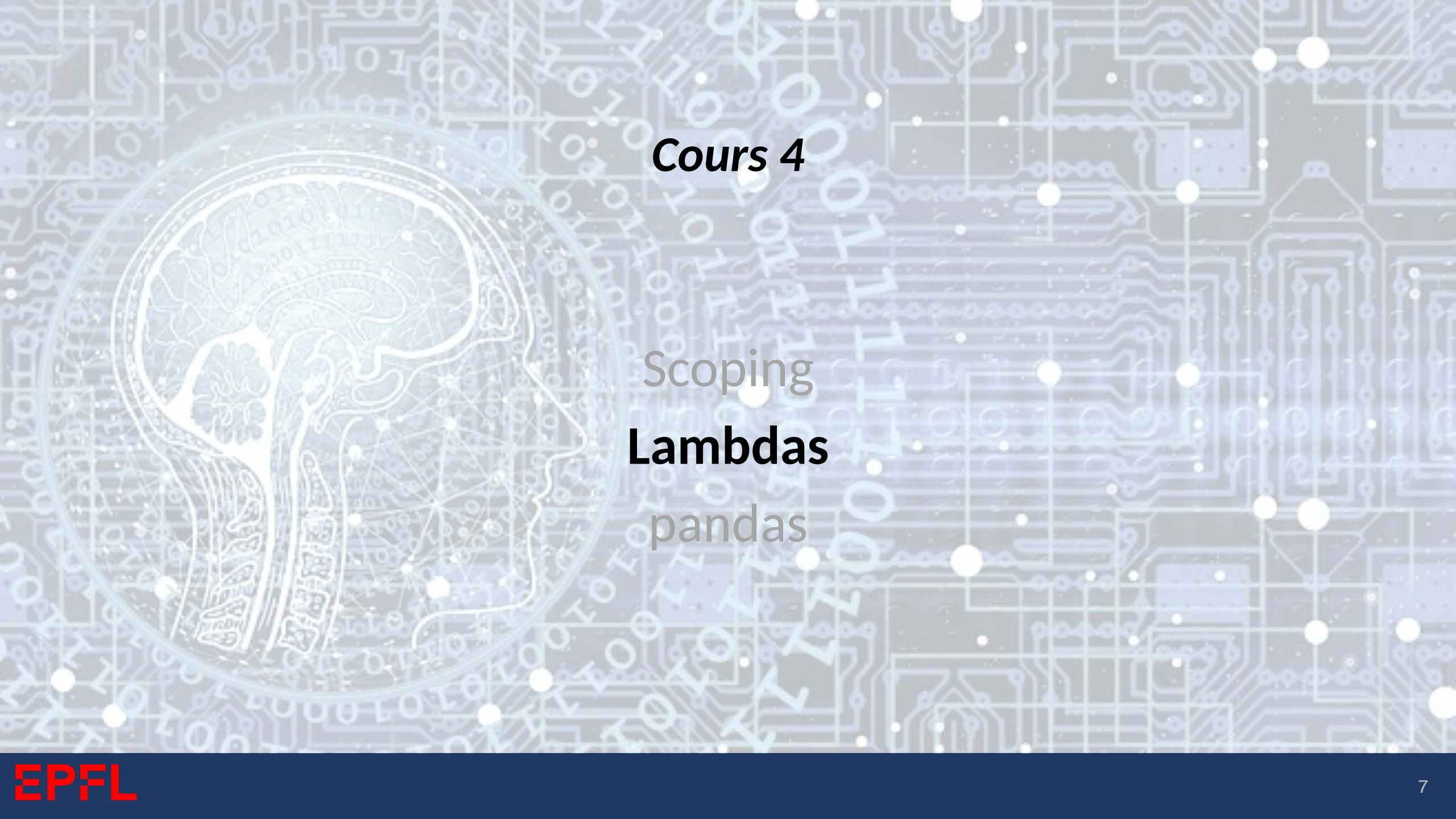
Lire une variable extérieure à la fonction fonctionne, mais par défaut, l'affectation est locale

Il faut ainsi le mot clé global *pour signaler qu'on ne veut pas créer une nouvelle variable locale avec cette affectation*

Sans global*: contient* "VS Code".
Avec global*: contient* "PyCharm".

Ceci dit, attention aux dangers des variables globales…

Quand plusieurs scopes imbriquées: nonlocal *à la place de* global



Cours 4

Scoping

Lambdas

pandas

Fonctions et lambdas

```python
def square(x: int) -> int:
    return x * x

print(square(4))

from typing import Callable

au_carré: Callable[[int], int] = square

print(au_carré(4))

square2 = lambda x: x ** 2

lambda x, y: x * y

lambda: 42

lambda x:
    y = x + 2
    return y * y
```

Ceci est une fonction normale

Elle est manipulable comme n'importe quelle autre valeur, elle peut notamment être assignée à une variable. Son type est Callable[[…], …]

Les lambdas *sont un moyen de créer rapidement une fonction anonyme sur une ligne sans* def *ou* return

Fonction qui accepte deux paramètres (non typés)

Fonction qui n'accepte aucun paramètre et retourne 42

Impossible de faire un lambda sur plusieurs ligne, il faut déclarer une fonction normale à la place

Les fonctions lambdas sont très pratiques lorsqu'on passe une petite fonction à une autre fonction

Utilisation des lambdas: defaultdict

```
from collections import defaultdict
from dataclasses import dataclass

@dataclass
class Person:
    name: str

people_by_id: dict[int, Person] = \
       defaultdict(lambda: Person("<unknown>"))

people_by_id[4] = Person("Jean-Philippe")


print(people_by_id[4])


print(people_by_id[5])
```

defaultdict(…) *crée un nouveau dictionnaire avec une valeur par défaut. Mais en fait son argument doit être une fonction à aucun paramètre qui donne une valeur par défaut*

C'est l'occasion de faire un lamba à aucun paramètre (rien entre lambda *et le deux-points) qui retourne une instance d'une nouvelle personne avec un nom inconnu*

On insère une personne liée à l'ID 4, par exemple

Affiche Person(name='Jean-Philippe'), *la clé est présente*

Affiche Person(name='unkown'), *en ayant appelé la fonction anonyme*

Utilisation des lambdas: *sorted*

```
data = ["une", "liste", "de", "mots"]

data_sorted1 = sorted(data)
# ['de', 'liste', 'mots', 'une']


data_sorted2 = \
    sorted(data, key=lambda s: len(s))
# ['de', 'une', 'mots', 'liste']

# idem:
data_sorted2 = sorted(data, key=len)


data_sorted3 = \
    sorted(data, key=lambda s: s.find("e"))
# ['mots', 'de', 'une', 'liste']
```

La fonction sorted() *utilise l'ordre naturel des éléments qu'elle trie; ici, via une comparaison lexicographique*

Mais elle a un paramètre optionnel, key*, qui est une fonction à un paramètre qui livre, pour chaque élément, le critère selon lequel le trier. les lambdas sont pratique pour implémenter des critères simples; ici, la longueur*

On aurait aussi simplement pu faire ainsi, car len *est elle-même une fonction à un paramètre…*

Ceci trie selon la position du premier «e» (ce qui donne –1 quand aucun «e» n'est présent)

Utilisation des lambdas: valeur par défaut d'un champ

```python
from dataclasses import dataclass, field

@dataclass
class Person:
    name: str
    friends: list['Person'] = []

p1 = Person("Arthur")
p2 = Person("Bérénice")
p1.friends.append(p2)
p2.friends.append(p1)
```

C'est impossible d'écrire ceci, car la valeur par défaut est partagée entre toutes les instances. On n'a pas le droit d'utiliser un type modifiable (comme une liste) ici

Ici, p1.friends *et* p2.friends *seraient en fait la même liste! Ce n'est pas ce qu'on veut*

```python
@dataclass
class Person:
    name: str
    friends: list["Person"] = \
        field(default_factory=lambda: [])
```

À la place, on indique une default_factory*, qui est une fonction à aucun paramètre qui retourne la valeur initiale du champ si non spécifié. Une bonne occasion pour un lambda*

Cours 4

Scoping

Lambdas

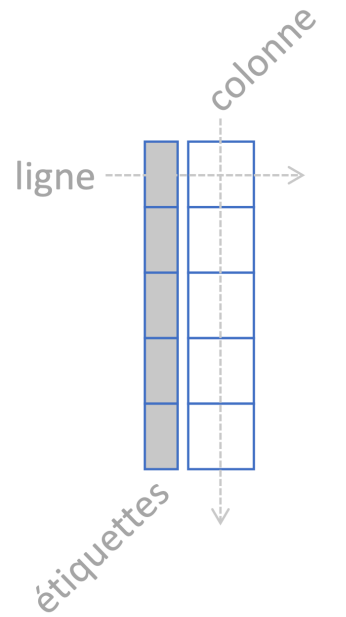
pandas

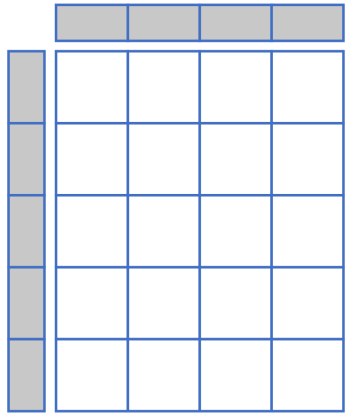
pandas — Manipulation et analyse de données

- **Pandas est une bibliothèque Python qui permet de manipuler facilement des données à analyser**
  - Utilise les arrays de NumPy pour être efficace *(ou autre implémentation native)*
  - *«Structures de données rapides, flexibles et expressives conçues pour rendre le travail avec des données "relationnelles" ou "étiquetées" à la fois facile et intuitif.»*
    - Étiquetées: on donne des noms aux colonnes, on utilise des index pour les lignes
    - Relationnelles: on facilite le traitement de plusieurs tables qui ont des données liées
- **Deux structures de données Pandas incontournables:**
  - Series (1D); DataFrame (2D)
- **Doc: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html**

pandas: *Series* et *DataFrame*

- **Series:** structure **unidimensionnelle**
  - ✦ Un array, mais en plus puissant
  - ✦ Comparaison tableur: une seule colonne dont les lignes sont étiquetée avec un *index*
- **DataFrame:** structure **bidimensionnelle**
  - ✦ Une série de colonnes, potentiellement de différents types
  - ✦ Les colonnes *et* les lignes sont étiquetées





pandas: *démo*

(cours04.ipynb dans votre repository)

pandas, quelques fonctions de base de la démo

Lecture	pd.read_csv() pd.read_excel() pd.read_html(), etc.	Lecture d'un fichier pour créer un dataframe depuis divers formats
Écriture	df.to_csv() df.to_excel() df.to_html, etc.	Écriture d'un dataframe *df* vers un fichier, divers formats
Check	df.head() df.info() df.describe() df.memory_usage()	Les 5 premières lignes Infos de base sur les colonnes et les types mean, min, max, quantiles, etc. pour chaque colonne Utilisation mémoire par colonne
Sélection	df.loc df.iloc df[…]	Sélection des lignes via valeur de l'index Sélection des lignes via leur index 0..*n*–1 Sélection des colonnes
Agrégation	df.sum() df.min() df.max(), etc.	Agrégation (par ligne ou colonnes)
Tri	df.sort_values(), df.sort_index()	Tri
Traitement	df.drop(…) df.assign(…) df.corr()	Suppression de lignes et colonnes Insertion de lignes et colonnes Matrice de corrélation

Résumé du cours d'aujourd'hui

- **Python est flexible dans ce qui peut être déclaré où et accessible depuis où**
  - ✦ Ne pas abuser de cette possibilité…
- **Les lambdas sont un moyen de déclarer facilement de petites fonctions**
  - ✦ Pratique pour les transformations dans un style fonctionnel, notamment
- ***pandas* est une bibliothèque de manipulation et analyse de données très puissante et flexible**
  - ✦ S'intègre bien aux notebooks

Autoévaluation — objectifs

- **Je suis capable de/d'…**
  - déterminer quelles variables sont accessibles depuis où, en lecture ou en écriture
  - reconnaître les endroits où l'usage de lambdas plutôt que de fonctions normales est possible et écrire des lambdas
  - charger des données CSV avec pandas
  - afficher des statistiques sommaires
  - sélectionner des lignes et colonnes de données chargées de différentes façons
  - calculer des valeurs dérivées (lignes et colonnes)
  - insérer et supprimer des lignes et des colonnes
  - appliquer des fonctions d'agrégation (*sum*, …) aux lignes et aux colonnes