Notions essentielles python

Les types de données de base en Python: Ces types comprennent les nombres (entiers et flottants), les chaînes de caractères, les listes, les tuples et les dictionnaires:

```
# Exemple avec des nombres
x = 10  # un entier
y = 3.14  # un flottant

# Exemple avec des chaînes de caractères
s = "Bonjour le monde"

# Exemple avec des listes
lst = [1, 2, 3, 4, 5]

# Exemple avec des tuples
tup = (1, 2, 3)

# Exemple avec des dictionnaires
dct = {"clé": "valeur", "nom": "Dupont"}
```

Manipulations:

Les manipulations essentielles des types de données de base en Python comprennent :

1. **Nombres (entiers et flottants)**:

```
# Exemple avec des nombres
x = 10  # un entier
y = 3.14  # un flottant

# Opérations de base
print(x + y)  # addition
print(x - y)  # soustraction
print(x * y)  # multiplication
print(x / y)  # division
print(x ** y)  # exponentiation

# Utilisation du module math
import math
```

```
print(math.sqrt(x))  # racine carrée
```

2. Chaînes de caractères:

```
# Exemple avec des chaînes de caractères
s = "Bonjour le monde"

# Concaténation
print(s + ", comment ça va?")

# Majuscules et minuscules
print(s.upper())
print(s.lower())

# Remplacement
print(s.replace("Bonjour", "Salut"))

# Division
print(s.split(" "))
```

3. Listes

```
# Exemple avec des listes
lst = [1, 2, 3, 4, 5]

# Ajout et retrait d'éléments
lst.append(6)  # ajouter à la fin
lst.insert(0, 0)  # ajouter à une position spécifique
lst.remove(3)  # retirer un élément spécifique

# Accès à des éléments spécifiques
print(lst[0])  # premier élément
print(lst[-1])  # dernier élément

# Trier une liste
lst.sort()

# Parcourir une liste
for element in lst:
    print(element)
```

4. tuplets

```python
# Exemple avec des tuples
tup = (1, 2, 3)

# Accès à des éléments spécifiques
print(tup[0])  # premier élément
print(tup[-1])  # dernier élément

# Parcourir un tuple
for element in tup:
    print(element)
```

5. Dictionnaires

```python
# Exemple avec des dictionnaires
dct = {"clé": "valeur", "nom": "Dupont"}

# Ajout et retrait de paires clé-valeur
dct["age"] = 30  # ajout
del dct["clé"]  # retrait

# Accès à des valeurs spécifiques à partir de clés
print(dct["nom"])

# Parcourir un dictionnaire
for key, value in dct.items():
    print(f"Clé: {key}, Valeur: {value}")
```

6. Contrôle de flux (boucles et instructions conditionnelles)

```python
# Exemple avec une boucle for
for i in range(5):
    print(i)

# Exemple avec une boucle while
i = 0
while i < 5:
    print(i)
    i += 1

# Exemple avec des instructions conditionnelles
x = 10
if x > 0:
    print("x est positif")
```

```
elif x < 0:
    print("x est négatif")
else:
    print("x est zéro")
```

7. Fonctions:

```
# Exemple de définition de fonction
def saluer(nom):
    print(f"Bonjour {nom}")

# Exemple d'utilisation de la fonction
saluer("Pierre")
```

8. Gestion des erreurs et des exceptions

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("On ne peut pas diviser par zéro!")
```

Bibliothèques Pandas

DataFrame et Series: Les deux structures de données clés dans pandas sont les DataFrames et les Series. Un DataFrame est une table de données bidimensionnelle, tandis qu'une Series est une colonne de données unidimensionnelle.

```
import pandas as pd

# Création d'un DataFrame
df = pd.DataFrame({
    "A": [1, 2, 3],
    "B": ["a", "b", "c"]
})

# Création d'une Series
s = pd.Series([1, 2, 3], name="A")
```

Lecture et écriture de données:

```
# Lecture d'un fichier Excel
df = pd.read_excel('fichier.xlsx', sheet_name='Feuil1', header=None)

# Écriture dans un fichier Excel
df.to_excel('nouveau_fichier.xlsx', index=False)

# Lecture d'un fichier CSV
df = pd.read_csv('fichier.csv')

# Écriture dans un fichier CSV
df.to_csv('nouveau_fichier.csv', index=False)
```

Indexation et sélection de données: savoir comment sélectionner des données spécifiques à partir d'un DataFrame en utilisant l'indexation par étiquette (`.loc`) et l'indexation par position (`.iloc`):

```
# Sélection par étiquette
print(df.loc[0, 'A'])  # valeur à la première ligne et colonne 'A'

# Sélection par position
print(df.iloc[0, 0])  # valeur à la première ligne et première colonne
```

Application iloc

```
import pandas as pd

# Lecture du fichier Excel
df = pd.read_excel('fichier.xlsx', header=None)

# Obtenir les années et les mois à partir des 3ème et 4ème lignes
years = df.iloc[2, 1:].astype(int).values
months = df.iloc[3, 1:].astype(int).values

# Créer un index de type datetime
date_index = pd.to_datetime({
    'year': years,
    'month': months,
    'day': [1]*len(years)
})

# Obtenir les noms des séries à partir de la première colonne
series_names = df.iloc[4:, 0]
```

```
# Obtenir les valeurs à partir du reste du DataFrame
values = df.iloc[4:, 1:].values

# Créer un nouveau DataFrame avec les séries comme colonnes et l'index datetime
df_new = pd.DataFrame(data=values, columns=series_names, index=date_index)
```

Manipulation de données:

```
# Trier les données
df.sort_values('A', ascending=False)

# Grouper les données
df.groupby('A').mean()

# Fusionner les données
df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})
df2 = pd.DataFrame({'A': ['A0', 'A2'], 'C': ['C0', 'C2']})
df_merged = pd.merge(df1, df2, on='A', how='outer')
```

`how='outer'` : C'est le type de fusion à effectuer. Il existe quatre types de fusion : 'left', 'right', 'outer' et 'inner'.

- 'left' : utilise uniquement les clés de gauche (c'est-à-dire de `df1`), similaire à SQL LEFT OUTER JOIN.
- 'right' : utilise uniquement les clés de droite (c'est-à-dire de `df2`), similaire à SQL RIGHT OUTER JOIN.
- 'outer' : utilise l'union des clés des deux DataFrame, similaire à SQL FULL OUTER JOIN.
- 'inner' : utilise l'intersection des clés des deux DataFrame, similaire à SQL INNER JOIN.

Calculs statistiques

```
# Calculs statistiques
print(df['A'].mean())  # moyenne
print(df['A'].std())  # écart type
print(df.corr())  # corrélation entre les colonnes
```

Bibliothèques Numpy

NumPy (Numerical Python) est une bibliothèque Python qui est très utile pour le calcul numérique, en particulier pour les opérations sur les tableaux à plusieurs dimensions (aussi appelés ndarrays). En fait, Pandas est construit sur NumPy, ce qui signifie que beaucoup de la fonctionnalité de Pandas dépend de NumPy.

NumPy est conçu pour effectuer des opérations mathématiques sur des tableaux entiers de manière efficace. Pandas, en revanche, offre une gamme beaucoup plus large de fonctionnalités pour manipuler et analyser des données, y compris le traitement de données manquantes, la manipulation de dates et de chaînes de caractères, la fusion et le pivotement de données.

```
import pandas as pd
import numpy as np

# Créer un DataFrame
df = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50],
    'C': [100, 200, 300, 400, 500]
})

# Utiliser une fonction Numpy sur une colonne
df['A'] = np.sqrt(df['A'])

# Ou utiliser une fonction Numpy sur l'ensemble du DataFrame
df = np.sqrt(df)

# Appliquer la fonction np.sqrt à chaque élément de la colonne 'A'
df['sqrt_A'] = df['A'].apply(np.sqrt)

# Sélectionner les lignes où la colonne 'A' est supérieure à 2
condition = df['A'].values > 2
selected_rows = df[condition]
```

1. **Opérations arithmétiques**: addition (`np.add`), soustraction (`np.subtract`), multiplication (`np.multiply`), division (`np.divide`), racine carrée (`np.sqrt`), log (`np.log`), etc.
2. **Opérations statistiques**: minimum (`np.min`), maximum (`np.max`), moyenne (`np.mean`), médiane (`np.median`), écart type (`np.std`), etc.
3. **Opérations trigonométriques**: sinus (`np.sin`), cosinus (`np.cos`), tangente (`np.tan`), etc.
4. **Opérations de comparaison**: égal (`np.equal`), supérieur (`np.greater`), inférieur (`np.less`), etc.

5. **Opérations booléennes**: et (`np.logical_and`), ou (`np.logical_or`), etc.