DAWAN Paris 11,rue Antoine Bourdelle, 75015 PARIS
DAWAN Nantes 32, Bd Vincent Gâche, 5e étage - 44200 NANTES
DAWAN Lyon Bt Banque Rhône Alpes, 2ème étage – 235 cours Lafayette 69006 LYON

Formation Python: fichiers et bases de données

Plus d'info sur http://www.dawan.fr ou 0810.001.917

Formateur: Matthieu LAMAMRA

Les fichiers en Python

- Ouvrir un fichier
  - Les fichiers forment un type en python
  - On les manipule avec la fonction de base **open**

```
open( path_to_filename, [ r || w || a ] b ? )
```

  - r « read » : Ouvre le fichier en lecture seule
  - w « write » : Ouvre le fichier en écriture, écrase le contenu si déjà créé, crée le fichier sinon
  - a « append » : Ouvre le fichier en écriture, écrit à la fin du fichier
  - b « binary » : Placé après les précédents, ouvre le fichier en mode binaire (01)

Les fichiers en Python

- Manipuler un fichier : read, write, close

```
fic = open("./test_fic.txt", "w") # création ou écrasement
fic.write("some content") # 12
fic.close()

fic = open("./test_fic.txt", "r") # lecture depuis le début
fic.read(4) # "some"
fic.read() # " content"
fic.close()

fic = open("./test_fic.txt", "a") # ajout à la fin
fic.write("\nother content") # 14
fic.close()
```

- **Toujours penser à refermer les fichiers quand on ne travaille plus avec**

Les fichiers en Python

- Manipuler un fichier : readlines, writelines

```
# lecture / écriture depuis le début
fic = open("test_fic.txt", "r+", encoding="utf8")
# déplace le curseur du fichier
print(fic.readlines())
fic.writelines(["\n3rd row", "\nquatrième ligne"])
fic.close()
```

Les fichiers en Python

- Manipuler le curseur d'un fichier

- **Pointeur sur la position de la dernière écriture ou du mode d'ouverture**
- **tell()** donne la position du curseur dans le fichier
- **seek(** *offset, [whence]***)** place le curseur à la position *offset* à partir de *whence avec* :

  os.SEEK_SET = 0 => début du fichier
  os.SEEK_CUR = 1 => position courante du curseur => en binaire pour python 3.2+
- os.SEEK_END = 2 => fin du fichier => en binaire pour python 3.2+

```
f =open("test_fic.txt", "r+", encoding="utf8")
f.read(13) # lit 13 caractères
print(f.tell()) # nb d'octets lus
f.seek(45) # déplacement de 44 octets
#UnicodeDecodeError
print(f.read(3)) # lit 3 caractères
f.close()
```

- **Attention, tell() et seek() sont des fonctions bas niveau, qui comptent des octets**

Les fichiers en Python

- With : le « context manager »

  - Ouvrir un fichier avec le mot-clé **with**
  - Permet d'écrre un bloc qui **gère la fermeture du fichier** à la fin de son exécution

```
with open("test_fic.txt", "r+", encoding="utf8") as f :
    f.read()
# fichier refermé
```

Les fichiers en Python

- Fichier comme itérateur

  - Un objet de type file possède un itérateur
  - **next() :** retourne la ligne suivante

```
for line in f :
```

Les fichiers en Python

- Manipuler les chemins : module os

- On utilise le module **os**
- **os.remove(***path_to_file***)** : supprime un fichier
- **os.mkdir(***path_to_dir, mode=0o777***)** : crée un répertoire dans un chemin existant
- **os.makedirs(***path_to_dir***)** : crée un répertoire et les répertoires intermédiaires
- **os.removedirs(***path_to_dir***)** : supprime récursivement les répertoires si vides
- **os.rename(***path_to_old, path_to_new***)** : renomme ou déplace le fichier ou le répertoire
- **os.renames(***path_to_old, path_to_new***)** : idem avec création des répertoires intermédiaires

Les fichiers en Python

- Manipuler les chemins : module os

- **os.chdir(***path_to_dir***)** : change le répertoire de travail « cd »
- **os.getcwd()** : renvoie le chemin du répertoire de travail « pwd »
- **os.path.exists(***path***)** : True si le chemin existe, False sinon
- **os.path.isdir(***path***)** : True si le chemin est celui d' un répertoire, False sinon
- **os.path.isfile(***path***)** : Idem pour un fichier
- **os.listdir(***path_to_dir***)** : retourne la liste du contenu du répertoire
- **os.chmod(***path, mode***)** : change les permissions d'un fichier ou d'un répertoire (en octale)

Les fichiers en Python

- Manipuler les chemins : module shutil

- **shutil.move(***src, dest***)** : déplace ou renomme un fichier ou un répertoire « mv »
- **shutil.copy(***src, dest***)** : copie un fichier ou un répertoire
- **shutil.copy2(***src, dest***)** : Idem avec les métadonnées
- **shutil.rmtree(***path***)** : supprime un dossier et son contenu (rm -rf)

Les fichiers en Python

- Manipuler les noms de fichiers

- **os.path.dirname(*path*)** : Retourne le chemin du répertoire contenant le chemin en paramètre
- **os.path.basename(*path*)** : Retourne le nom du fichier ou du répertoire sans le chemin
- **os.path.split(*path*)** : Retourne un tuple des deux précédents
- **os.path.splitext(*path*)** : Retourne un tuple pour obtenir l'extension

Les fichiers en Python

- Le format CSV

  - **Comma Separated Values** : fichiers structurés en lignes / colonnes dits « à plat »

```
name;position;symbol;weight
Hydrogen;1;H;1.0079
Helium;2;He;4.0026
Lithium;3;Li;6.941
Beryllium;4;Be;9.0122
Boron;5;B;10.811
Carbon;6;C;12.0107
Nitrogen;7;N;14.0067
Oxygen;8;O;15.9994
Fluorine;9;F;18.9984
Neon;10;Ne;20.1797
Sodium;11;Na;22.9897
Magnesium;12;Mg;24.305
Aluminum;13;Al;26.9815
Silicon;14;Si;28.0855
Phosphorus;15;P;30.9738
Sulfur;16;S;32.065
Chlorine;17;Cl;35.453
Argon;18;Ar;39.948
Potassium;19;K;39.0983
Calcium;20;Ca;40.078
```

	A	B	C	D
1	name	position	symbol	weight
2	Hydrogen	1	H	1.0079
3	Helium	2	He	4.0026
4	Lithium	3	Li	6.941
5	Beryllium	4	Be	9.0122
6	Boron	5	B	10.811
7	Carbon	6	C	12.0107
8	Nitrogen	7	N	14.0067
9	Oxygen	8	O	15.9994
10	Fluorine	9	F	18.9984
11	Neon	10	Ne	20.1797
12	Sodium	11	Na	22.9897
13	Magnesium	12	Mg	24.305
14	Aluminum	13	Al	26.9815
15	Silicon	14	Si	28.0855
16	Phosphorus	15	P	30.9738
17	Sulfur	16	S	32.065
18	Chlorine	17	Cl	35.453
19	Argon	18	Ar	39.948
20	Potassium	19	K	39.0983
21	Calcium	20	Ca	40.078

Les fichiers en Python

- Le format CSV

➢ On utilise le module csv et les classes **csv.reader et csv.writer**

```
import csv

with open("elements.csv", "r", encoding="utf8") as f :
    reader = csv.reader(f, delimiter=";", quotechar="'")
    for row in reader :
        print(", ".join(row))
```

```
new_row = ["Scandium", 21, "Sc", 44.956]

with open("elements.csv", "a", encoding="utf8", newline="") as f :
    writer = csv.writer(f, delimiter=";", quotechar="'", quoting=csv.QUOTE_MINIMAL)
    writer.writerow(new_row)
```

Les fichiers en Python

- Le format JSON

  - JavaScript Object Notation : syntaxe utile au stockage et à l'échange de données

```
{"elements":[
  { "position": 1, "name": "Hydrogen", "weight": 1.0079, "symbol": "H" },
  { "position": 2, "name": "Helium", "weight": 4.0026, "symbol": "He" },
  { "position": 3, "name": "Lithium", "weight": 6.941, "symbol": "Li" },
  { "position": 4, "name": "Beryllium", "weight": 9.0122, "symbol": "Be" },
  { "position": 5, "name": "Boron", "weight": 10.811, "symbol": "B" },
  { "position": 6, "name": "Carbon", "weight": 12.0107, "symbol": "C" },
  { "position": 7, "name": "Nitrogen", "weight": 14.0067, "symbol": "N" },
  { "position": 8, "name": "Oxygen", "weight": 15.9994, "symbol": "O" },
  { "position": 9, "name": "Fluorine", "weight": 18.9984, "symbol": "F" },
  { "position": 10, "name": "Neon", "weight": 20.1797, "symbol": "Ne" },
  { "position": 11, "name": "Sodium", "weight": 22.9897, "symbol": "Na" },
  { "position": 12, "name": "Magnesium", "weight": 24.305, "symbol": "Mg" },
  { "position": 13, "name": "Aluminum", "weight": 26.9815, "symbol": "Al" },
  { "position": 14, "name": "Silicon", "weight": 28.0855, "symbol": "Si" },
  { "position": 15, "name": "Phosphorus", "weight": 30.9738, "symbol": "P" },
  { "position": 16, "name": "Sulfur", "weight": 32.065, "symbol": "S" },
  { "position": 17, "name": "Chlorine", "weight": 35.453, "symbol": "Cl" },
  { "position": 18, "name": "Argon", "weight": 39.948, "symbol": "Ar" },
  { "position": 19, "name": "Potassium", "weight": 39.0983, "symbol": "K" },
  { "position": 20, "name": "Calcium", "weight": 40.078, "symbol": "Ca" }
]}
```

Les fichiers en Python

- Le format JSON

- On ultilise le module **json, les fonctions**
  - **loads et dumps** pour travailler avec str
  - **load et dump** pour travailler avec des fichiers

- Ces fonctions exploitent la similitude entre objet Json et dictionnaire python

```
import json

with open("elements.json", "r", encoding="utf8") as f:
    elements = json.load(f))
    # elements = json.loads(f.read())
    elements["elements"][0]["name"]
```

```
with open("elements.json", "w", encoding="utf8") as f:
    json.dump(data, f, indent=2)
    # f.write(json.dumps(data))
```

Les fichiers en Python

- Le format XML

  - **eXtended Markup Language** : langage de balise similaire à HTML

  - Utile pour stocker et requêter des ensembles de données de moyenne taille sans SQL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <elements>
    <element>
      <name>Hydrogen</name>
      <position>1</position>
      <symbol>H</symbol>
      <weight>1.0079</weight>
    </element>
    <element type="rare-gas">
      <name>Helium</name>
      <position>2</position>
      <symbol>He</symbol>
      <weight>4.0026</weight>
    </element>
    <element>
      <name>Lithium</name>
      <position>3</position>
      <symbol>Li</symbol>
```

Les fichiers en Python

- Le format XML
  - On ultilise le module **xml.etree.ElementTree**
  - On charge d'abord le fichier comme arborescence
  - Puis on selectionne le nœud racine pour traverser ou modifier l'arborescence

```
import xml.etree.ElementTree as ET

tree = ET.parse("elements.xml")
root = tree.getroot()

tag_name, attributes = root.tag, root[0][1].attrib
text = root[0][2][0].text
```

```
for elem in root:
   print( elem.tag )

for elem in root.iter("element"):
    print( elem.find("name").text )
```

Les fichiers en Python

- Le format XML
  - La mise à jour de l'arbre se fait par fabrications et attachements de balise

```
new_elem = ET.Element("element")

for tag, val in {"name": "Scandium", "position": "21", "weight": "44.9" , "symbol": "Sc"}.items():
    new_tag = ET.Element(tag)
    new_tag.text = val
    new_elem.append(new_tag)

ET.tostring(new_elem)

root[0].append(new_elem)
tree.write("elements.xml")
```

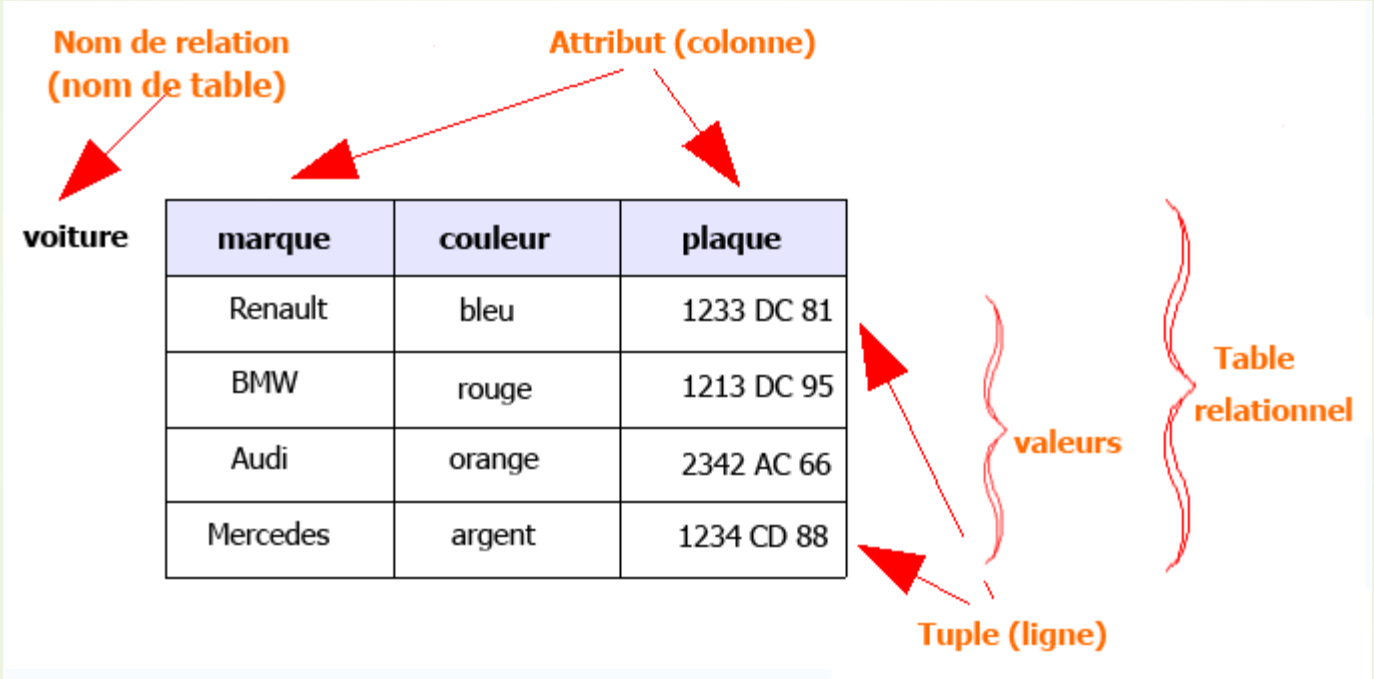
Python : bases de données

- Bases de données relationnelles

- Une base de donnée relationnelle est un **ensemble de fichiers**
- Ces fichiers organisent des données dans des tableaux à deux dimensions : **les tables**
- Les tables contiennent des **enregistrements (tuples!)** divisés en **champs nommés**
- Les champs forment des **colonnes** par nom de champs
- Les tables peuvent être apparentées par des **relations entre champs**
- Les bases de données relationnelles sont accessibles par des **requêtes SQL sur un serveur**
- Ex : **mysql/Mariadb, postgresql, oracle, IBM db2, sqlite3**
- En python sqlite3 est disponible dans la librairie standard, les autres par pip install

Python : bases de données

- Bases de données relationnelles



Python : bases de données

- SQL : création de tables

```
CREATE TABLE IF NOT EXISTS proprietaires (
    plaque CHAR(10) NOT NULL,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    PRIMARY KEY (plaque)
)  ENGINE=INNODB;

CREATE TABLE IF NOT EXISTS voitures (
    voiture_id INT AUTO_INCREMENT,
    marque VARCHAR(100),
    couleur VARCHAR(100) NOT NULL,
    plaque CHAR(10) NOT NULL,
    create_date DATE CURRENT_TIMESTAMP,
    description TEXT,
    PRIMARY KEY (voiture_id)
    FOREIGN KEY proprietaires_plaque (plaque)
REFERENCES proprietaires(plaque)
)  ENGINE=INNODB;
```

Python : bases de données

- SQL : insertion de données

```sql
INSERT INTO proprietaires (plaque, nom, prenom) VALUES
  ('1233 DC 81', 'Martin', 'Rémy'),
  ('1213 FC 98', 'Benhamicha', 'Hakim'),
  ('2342 AC 43', 'Alloun', 'Jérémie')

INSERT INTO voitures (voiture_id, marque, couleur, plaque) VALUES
  (1, 'Renault', 'bleu', '1233 DC 81'),
  (2, 'BMW', 'rouge', '1233 DC 81'),
  (3, 'Audi', 'gris', '1233 DC 81'),
```

Python : bases de données

- SQL : requêtes sur les données

```sql
SELECT voiture_id, marque FROM voitures WHERE couleur = 'bleu'

UPDATE voitures SET couleur = 'rouge' WHERE marque = 'Renault'

SELECT p.nom, p.prenom, v.marque, v.plaque
FROM proprietaires p JOIN voitures v ON p.plaque = v.plaque
ORDER BY v.plaque DESC
```

Python : bases de données

- DBAPI : Principe

  - Interface d'accès à la plupart des serveurs de base de données selon les étapes suivantes :
  - **Établir une connexion**
  - **Créer un curseur** et lui attribuer une ou plusieurs requêtes
  - **Exécuter** la ou les requêtes sur le curseur
  - **Itérer** sur les éléments retournés : **FETCH**
  - **Fermer la connexion**

Python : bases de données

- Sqlite3

  - base de données **dans un unique fichier sur le disque**, sans accès serveur :
  - **Pas d'utilisateur**
  - **Disponible dans la lib standard**
  - Lors de la connexion, si la base n'existe pas, elle est créée
  - **On peut également travailler en mémoire : « :memory: »**

Python : bases de données

- Sqlite3 : connection / requête / retour
  - On peut changer la forme des résultats sur l'objet de connection ou le curseur (hydration)

```
import sqlite3

with sqlite3.connect("database.db") as conn:
    # conn.row_factory = sqlite3.Row
    cur = conn.cursor()
    cur.execute("SELECT SQLITE_VERSION() as version")
    tup = cur.fetchone()
    print(tup[0])
    # dct = dict(cur.fetchone())
    # print(dct["version"])
```

Python : bases de données

- Pymysql
  - base de données de production open source la plus utilisée
  - **Disponible via pip install**

```
db = pymysql.connect("host", "username", "passwd", "database")
with db.cursor() as cursor:
    cursor.execute("""CREATE TABLE IF NOT EXISTS pays (
        iso2 CHAR(2) NOT NULL,
        name VARCHAR(100) NOT NULL,
        PRIMARY KEY (iso2)
    )""")
```

Python : bases de données

- dbapi : requêtes paramétrées

```
db = pymysql.connect("host", "username", "passwd", "database")
    with db.cursor() as cursor:
        req = """INSERT INTO voitures (plaque, marque) VALUES
        (%s, %s)"""
        values = ("AZEF6Y7F54", "Peugeot")
        cursor.execute(req, values)
```

Python : bases de données

- dbapi : « fetch » des enregistrements

- **fetchone :** récupération de l'enregistrement suivant
- **fetchmany([size]) :** récupération de size enregistrements suivants
- **fetchall() :** récupération de tous les enregistrements

```
with conn:
    cur = conn.cursor()
    cur.execute("SELECT * FROM propietaires")
    records = cur.fetchall()
```

Python : bases de données

- dbapi : les transactions

➢ **commit :** valide les requêtes au bout d'un bloc **try : automatique dans un bloc with**

```
try:
    with conn:
        cur = conn.cursor()
        cur.execute("""INSERT INTO proprietaires
        (plaque, nom, prenom) VALUES ('1233DC43FR', 'smith', 'bob')
        """)
        print(cur.rowcount)
except sqlite3.OperationalError as e:
    conn.rollback()
```