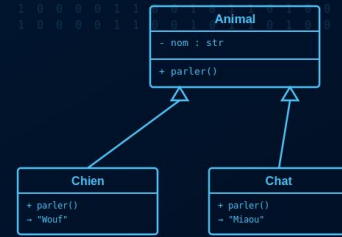


I. Introduction



héritage (POO)



```

class Animal:
    def __init__(self, nom):
        self.nom = nom
    def parler(self):
        raise NotImplementedError

class Chien(Animal):
    def parler(self):
        return "Wouf !"
    
```

M. Martin

1

1

Objectif de ce cours

S'adresse aux apprentis en GEII 2A

12h de Cours, 12h de TD, 8h de TP

Évaluations : Compte rendu de TP (30%) et Examen pratique (70%)



Compétences à acquérir :

- 1- Connaissances de bases et fonctionnalités avancées en Python,
- 2- Comprendre le fonctionnement des tests logiciels,
- 3- Utilisation de bibliothèques pour le domaine scientifique,
- 4- Analyser un problème avec les notions d'objets et de classes avec leur association

2

2



3



- Dong, S., Wang, P., Abbas, K.. A survey on deep learning and its applications. Computer Science Review 40, 100379 (2021)
- Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning : concepts, CNN architectures, challenges, applications, future directions. J. Big Data 8, 53 (2021)
- Dargan, S., Kumar, M., Ayyagari, M.R. et al. A Survey of Deep Learning and Its Applications : A New Paradigm to Machine Learning. Arch Computat Methods Eng 27, 1071–1092 (2020)



4



Blogs

- <https://mml-book.com>
- <https://sebastianraschka.com/blog>
- <https://machinelearningmastery.com>

Chaînes Youtube

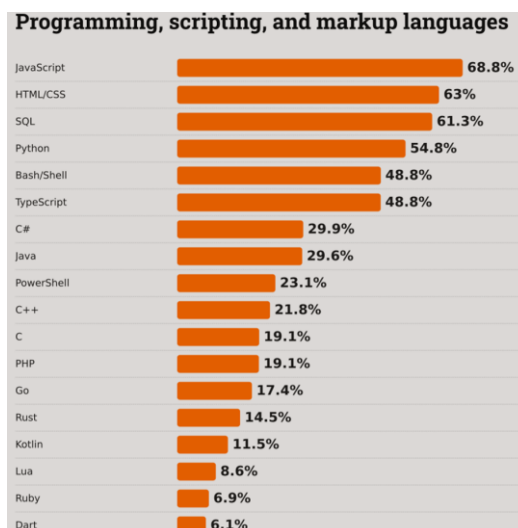
- Machine Learnia : <https://www.youtube.com/@MachineLearnia>
- AI for beginners : <https://www.youtube.com/@EasyAIForAll>
- Une vidéo à part : https://youtu.be/Z_hKt_kYNOU?si=Jsm794ele79pVJI1

Jeux de données

- <https://www.kaggle.com>



5



6

5 raisons d'utiliser Python

1. **Lisible et concis** : favorise **productivité** et **maintenabilité**
2. **Langage interprété** : pas de compilation lourde, le code peut être exécuté immédiatement
3. **Typage dynamique** : flexibilité pour coder rapidement sans déclarer les types
4. **Polyvalent** :
 - intelligence artificielle et apprentissage automatique
 - traitement de données massives (big data)
 - simulation numérique en sciences et ingénierie
 - développement Web
 - cybersécurité
 - automatisation de tâches
5. **Écosystème riche** : des milliers de bibliothèques (NumPy, Matplotlib, etc.)

7

7

5 raisons de ne pas utiliser Python

1. **Temps réel et embarqué** : peu adapté
2. **Typage dynamique** : les erreurs de type sont détectées à l'exécution, ce qui peut provoquer des bogues difficiles à anticiper
3. **Performances limitées** : plus lent que C/C++/Fortran/Rust pour les calculs lourds, même si des bibliothèques optimisées existent (*NumPy*)
4. **Langage non-système** : pas d'accès direct à la mémoire ou au matériel comme en C ou Rust
5. **Parallélisme limité** : le GIL (Global Interpreter Lock) empêche l'exécution simultanée de plusieurs threads sur plusieurs cœurs

8

8



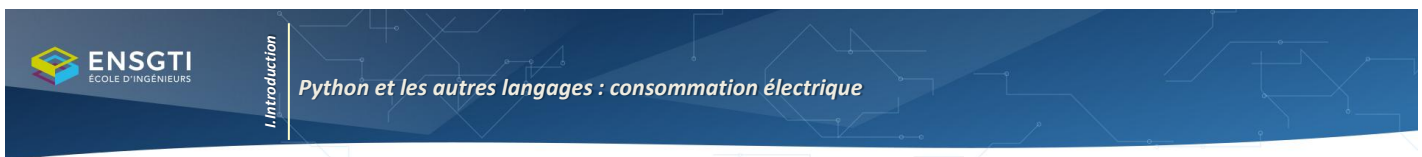
Les tableaux suivants sont extraits de l'article :

R. Pereira et al. Energy Efficiency across Programming Languages. In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, p. 256-267, 2017.

- Ces résultats rappellent que Python n'est pas conçu pour la performance brute, mais pour la rapidité de développement
- En pratique, les ingénieurs combinent Python avec des bibliothèques optimisées en C, C++, voire Rust



9



Langage	Consommation électrique
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) JavaScript	4.45
(i) PHP	29.30
(v) Erlang	42.23
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58



10

Python et les autres langages : temps d'exécution

Langage	Temps d'exécution
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(i) JavaScript	6.52
(i) PHP	27.64
(v) Erlang	36.71
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90

11

11

Python et les autres langages : consommation mémoire

Langage	Consommation mémoire
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Ruby	3.97
(i) JavaScript	4.59
(v) Java	6.01
(i) Perl	6.62
(v) Erlang	7.20

12

12



1. Interpréteur interactif

- Lancer depuis un terminal :

```
python
```

- Une instruction est immédiatement évaluée et le résultat est affiché
- Utile pour tester rapidement une idée, vérifier un calcul ou explorer une fonction



13



2. Scripts exécutés dans un environnement virtuel

- Le code est écrit dans un fichier texte avec une extension « .py »

- On exécute le script depuis le terminal :

```
python mon_script.py
```

- Pour isoler les dépendances et travailler de manière reproductible, on utilise un **environnement virtuel** (module **venv** par exemple)
- Chaque projet est compartimenté et possède ainsi ses propres versions de bibliothèques



14

3. Bloc-notes Jupyter

- Interface interactive dans le navigateur Web
- Mélange code, résultats, graphiques et texte explicatif dans le même document
- Très utilisé en recherche et enseignement

- Lancement :

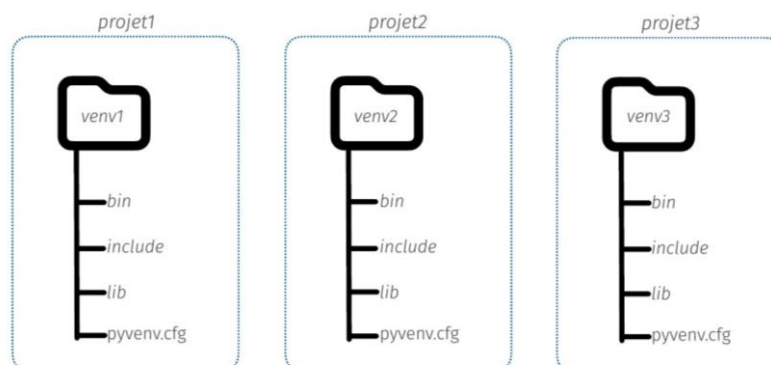
[jupyter notebook](#)

- Chaque bloc-notes contient des cellules de code et des cellules de texte (Markdown)

15

15

- **Bonne pratique** : structurer les projets avec des dossiers dédiés et des environnements isolés



16

16

1. Créer un projet

```
mkdir projet1
cd projet1
mkdir src data results
```

2. Créer un environnement virtuel

```
python -m venv env1
source env1/bin/activate # sous Linux / macOS
env1\Scripts\activate    # sous Windows
```

3. Installer des bibliothèques

```
pip install numpy matplotlib
```

4. Sauvegarder les dépendances dans « requirements.txt »

```
pip freeze > requirements.txt
```

17

17

5. Partager ou reproduire l'environnement

```
pip install -r requirements.txt
```

6. Désactiver un environnement virtuel

```
deactivate
```

Exemple de fichier « requirements.txt »

```
matplotlib == 3.9.2
numpy == 1.26.4
pandas == 2.2.2
scikit-image == 0.24.0
scikit-learn == 1.7.0
scipy == 1.14.1
* * *
```

18

18

```
mon_projet/  
|-- src/           # scripts et modules Python  
|-- data/          # données d'entrée  
|-- results/       # résultats générés (graphiques, rapports)  
|-- env/           # environnement virtuel (non versionné)  
|-- tests/         # tests  
|-- requirements.txt # dépendances du projet  
|-- README.md      # documentation du projet
```

- src/ : contient vos scripts et modules
- data/ : fichiers de données brutes
- results/ : sorties (graphiques, fichiers traités)
- requirements.txt : liste des bibliothèques installées
- env/ : environnement virtuel (à recréer à l'aide de venv car ne doit jamais être versionné)
- tests : fichiers de test

Séparer code, données et résultats est une habitude professionnelle