Introduction à Python

Chuan Xu

chuan.xu@univ-cotedazur.fr

Sept. 2024

Plan de cours

Moodle : SLEI505 - ECUE IA : Langage Python pour l'IA

1. 1: Python classic (Cours + TP)
2. 2: Conda, Python avancée
3. 3: Python Classes
4. 4: NEF
5. 5: Numpy/Panda/Matplotlib
6. 6-7: PyTorch
7. 8: Projet

Plan de cours

Moodle : SLEI505 - ECUE IA : Langage Python pour l'IA

1. 1: Python classic (Cours + TP)
2. 2: Conda, Python avancée
3. 3: Python Classes
4. 4: NEF
5. 5: Numpy/Panda/Matplotlib
6. 6-7: PyTorch
7. 8: Projet

Évaluation : Exam (50%) et Projet (50%)

Caractéristiques du langage Python

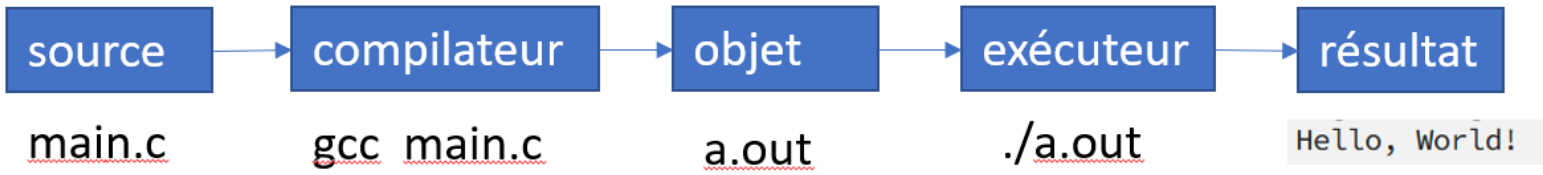
1. Langage Open Source (libre et gratuit)
2. Orientation objet
3. Simplicité : syntaxe claire (lisible), Gestion mémoire automatique, typage dynamique fort
4. Langage interprété rapide
5. Disponibilité de bibliothèques

Production des programme

- Deux techniques
  1. Compilation : la traduction du source en langage machine (binaire)

Production des programme

- Deux techniques
  1. Compilation : la traduction du source en langage machine (binaire)




```
#include <stdio.h>

int main(void)
{
      printf("Hello, World!\n");
      return (0);
}
```

Production des programme

- Deux techniques
  1. Compilation : la traduction du source en langage machine (binaire)

  2. Interprétation : chaque ligne du source analysé est traduite au fur et à mesure

Production des programme

- Deux techniques
  1. Compilation : la traduction du source en langage machine (binaire)

  2. Interprétation : chaque ligne du source analysé est traduite au fur et à mesure

Production des programme

- Deux techniques
  1. Compilation : la traduction du source en langage machine (binaire)

  2. Interprétation : chaque ligne du source analysé est traduite au fur et à mesure


```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

My first PHP page

Hello World!

Production des programme

- Deux techniques
  1. Compilation : la traduction du source en langage machine (binaire)

  2. Interprétation : chaque ligne du source analysé est traduite au fur et à mesure

	Avantage	Inconvénient
Compilation	Exécution rapide	Toute modification nécessite une nouvelle traduction
Interprétation	Développement simple	Exécution lente

Production des programme : Python 3

Une combinaison des deux techniques : l'interprétation du bytecode compilé.

- Charge le fichier *.py* en mémoire vive, produit le bytecode et enfin l'exécute
- Sauvegarde le bytecode produit (*.pyo* ou *.pyc*) et recharge simplement le ficher le plus récent
- Il n'est pas nécessaire de compiler explicitement un module

Python : mode interactif

1. Lancer l'interpréteur dans son "mode interactif"

```
C:\Users\chxu>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. Calculatrice, afficher des messages, obtenir de l'aide à propos des fonctions...
3. Exit() pour quitter

Python : mode programme

- Donner votre script Python

```
C:\Users\chxu>python mon_script.py
```

```
def my_function():
        print("Hello world!")
print("La fin de script")
```

Python : mode programme

- Donner votre script Python

```
C:\Users\chxu>python mon_script.py
```

```
def my_function():
        print("Hello world!")
print("La fin de script")
```

- Activer le mode interactif en même temps

```
C:\Users\chxu>python -i mon_script.py
La fin de script
>>> my_function()
Hello world!
>>>
```

Variables

- Noms : lettres, chiffres, tirets bas (mes_1, 1_mes)
  1. Aucun espace
  2. Évitez d'utiliser des mots clés et des noms de fonctions
  3. Court et descriptible
  4. Attention lettre l, O avec le chiffre 1, 0

Chiffre et commentaire

1. Affectation multiple : `x,y,z = 0,0,0`
2. Divisez deux nombres, retournez un float
3. Regroupez les chiffres en utilisant tiret bas : a=140_000_000
4. Le dièse # avant l'instruction pour commenter la ligne
5. Guillemets triples """ pour les commentaires multilignes

Chaîne de caractères

1. "chaîne" ou 'chaîne'
2. On peut concaténer (ajouter bout à bout) deux chaînes avec l'opérateur +
3. F-string : insérer des variables ou expressions à l'intérieur d'une chaîne de caractères (Python >3.6) Préfixez la string avec la lettre `f`.

```
prenom, nom = "Patrick", "David" #Chaîne de caratères
nom_complete = prenom + " " + nom #Concaténation de chaînes
print(f"Bonjour {nom_complete}") #Insérer les variables
print(f"Bonjour {prenom + ' ' + nom}") #Insérer les expression
print(f"Le résultat de 5 + 10 est égal à {5 + 10}") #Insérer les expression

#Précision flattante dans f-string  Syntaxe: {variable :.{précision}{type}}
pi = 3.1415926
print(f'Le valeur de pi est {val:.3f}')
print(f'Le valeur de pi est {val:.4f}')
```

Suite arithmétique

la fonction intégrée de Python
`range(debut, fin, pas)`:

1. debut: le premier élément dans la séquence
2. fin: l'entier jusqu'auquel on va boucler mais **sans** inclure
3. pas: combien on va incrémenter entre chaque élément dans la séquence

Suite arithmétique

la fonction intégrée de Python
`range(debut, fin, pas)`:

1. debut: le premier élément dans la séquence
2. fin: l'entier jusqu'auquel on va boucler mais **sans** inclure
3. pas: combien on va incrémenter entre chaque élément dans la séquence

- `range(0,10,2)` $\rightarrow$ [0,2,4,6,8]
- `range(10)` : pareil que `range(0,10,1)`
- `range(2,8)` : pareil que `range(2,8,1)`

Liste

Collecte des variables dans un ordre particulier :
`labels=[1,2,3]`

Liste

Collecte des variables dans un ordre particulier : `labels=[1,2,3]`

```
# Longueur de la liste
len(labels)

# Premier et dernier élément dans une liste
labels[0], labels[-1]

# Ajouter et insérer un élément dans une liste
labels.append(4)
labels.insert(1, 5)

# Prendre une sous-liste
labels[:3], labels[:-2], labels[2:4]

# Minimum, maximum, somme de la liste
min(labels), max(labels), sum(labels)

# Conversion de suite arithmétique à liste
labels_conv = list(range(3))
```

Tuples

Un tuple est une collection ordonnée et non modifiable d'éléments éventuellement hétérogènes.

```
# Séparés par des virgules, et entourés de parenthèses
mon_tuple = ('a', [1,2], 3)

# Non modifiables !
mon_tuple.append(4)
"""
Traceback (most recent call last) :
File "<stdin>", line 1, in <module>
AttributeError : 'tuple' object has no attribute 'append'
"""
```

Opérations communes pour chaîne, liste et tuple

l'opération	son effet
`x in s`	`True` si `s` contient `x`, `False` sinon
`x not in s`	`True` si `s` ne contient pas `x`, `False` sinon
`s + t`	concaténation de `s` et `t`
`s * n, n * s`	`n` copies (superficielles) concaténées de `s`
`s[i]`	i^e élément de `s` (à partir de 0)
`s[i:j]`	tranche de `s` de `i` (inclus) à `j` (exclu)
`s[i:j:k]`	tranche de `s` de `i` à `j` avec un pas de `k`
`len(s)`	longueur de `s`
`max(s), min(s)`	plus grand, plus petit élément de `s`
`s.index(i)`	indice de la 1^{re} occurrence de `i` dans `s`
`s.count(i)`	nombre d'occurrences de `i` dans `s`

Dictionnaire

Une collection de paires clé-valeur, modifiables mais non-ordonnés

- Enveloppé dans des accolades
- Les paires clé-valeur sont séparées par des virgules

```
mon_dict = {"nom": "Gayerie", "prenom": "David"}
nom = mon_dict["nom"] # Pour accéder le valeur,
# utiliser les crochets et préciser la valeur de la clé
prenom = mon_dict["prenom"]
size_dict = len(mon_dict)
#Créer un dictionnaire vide
a = {}
#Ajouter des valeurs
a["nom"] = "Wayne"
a["prenom"] = "Bruce"
#Vérifier si un clé existe déjà dans un dictionnaire
if "nom" in a: print(f"Oui, le clé existe")
```

Ensemble

Une collection non ordonnée d'éléments hachables uniques

```
X = {1,2,3} # Créer un ensemble d'entiers
V = set() # Créer un ensemble vide

l = [1,2,"s",2,1]
Y = set(l) # Convertir une liste au l'ensemble

Z = set('pass') # Convertir une châine au l'ensemble
W = set('spam')

W - Z # ensemble des éléments de W qui ne sont pas dans Z
W & Z # l'intersection des deux ensembles
W | Z # union des deux ensembles

X.add(4)
X.remove(3)
```

If condition

- `if condition: ...votre code`
  `else: ... votre code`
- *La condition a un type Boolean (`True, False`)*
- `if condition: ...votre code`
  `elif: ... votre code`
  `else: ... votre code`
- Comparateur : `==, !=, >=, <=, >, <`
- Logique booléenne : **and**, **or**, **not**

```
num = 3
if num >= 0:
    #Espace indentation (soit touch TAB soit 4 espaces)
    print(num, "est un nombre positive")
else:
    print(num, "est un numbre négative")

flag = True
if flag:
   print("Positive")

if flag is False:
   print("Négative")
```

Itérables

Un objet qui contient une séquence d'éléments sur lesquels on peut itérer (c.f. chaîne de caractères, suite arithmétique, tuple, liste, dictionnaire, set...).

```
# Parcourir une suite arithmétique
for i in range(10):
        # Block d'instructions avec espace indentation
    print(i)
for j in range(3, 10, 3):
    print(j)
```

Parcourir une liste

```python
# Parcourir une liste
nombres = [1, 2, 3]
for num in nombres:
    print(num)

# Parcourir deux listes en même temps
lettres = ["a", "b", "c"]
for num, lettre in zip(nombres, lettres):
    print(num, lettre)

# Parcourir une liste avec les indices
for ind in range(len(lettres)):
    print(ind, lettres[ind])

for ind, lettre in zip(range(len(lettres)), lettres):
    print(ind, lettre)
```

Compréhension de liste python

Générer une liste en une seule ligne de code

1. Executer une fonction sur chaque élément dans la liste :

```
new_list = [f(e) for e in list]
```

2. Filtrer une liste :

```
new_list = [f(e) for e in list if condition(e)]
```

Compréhension de liste python

Générer une liste en une seule ligne de code

1. Executer une fonction sur chaque élément dans la liste :

```
new_list = [f(e) for e in list]
```

2. Filtrer une liste :

```
new_list = [f(e) for e in list if condition(e)]
```

```
l = [1, 2, 3]
l1 = []
for j in l:                    <==>      l1 = [2*j for j in l]
    l1.append(2*j)

l2 = []
for j in l:
    if j % 2 != 0:             <==>      l2 = [2*j for j in l if j%2!=0]
        l2.append(2*j)
```

Parcourir un dictionnaire

```
civilite = {'M':'Monsieur', 'Mme':'Madame'}
```

- On parcourt les clés du dictionnaire :

```
for k in civilite.keys():
    print(k)
```

- On parcourt les valeurs du dictionnaire :

```
for k in civilite.values():
    print(k)
```

- On parcourt les clés et les valeurs du dictionnaire :

```
for k, v in civilite.items():
    print(k, v)
```

Les fonctions

Une fonction est une suite d'instructions que l'on peut appeler avec un nom (lettres minuscules et tirets bas).

```
# on utiliser le mot clé def suivi d'un nom puis de parenthèses
# et ensuite d'un double point.
def indique():
        return 10 # Espace indentation (soit touch TAB soit 4 espaces)
num = indique() # Appeler la fonction et sauvegarder le retourne

def somme(a,b):
        return a+b
res = somme(2,3)

def multiplication(a=1,b=2): # fonction avec valeurs par défaut
        return a*b
res = multiplication()
res = multiplication(3)
res = multiplication(b=3)

"""En combinant les valeurs par défaut et le nommage des paramètres,
les valeurs par défaut doient être regroupés à la fin de la liste des paramètres"""
```

Les fonctions

```
# Exemple avec utilisation d'un return multiple
import math
def surfaceVolumeSphere(r):
        surf = 4.0 * math.pi * r**2
        vol = surf * r/3
        return surf, vol

# Exemple passage d'une fonction en paramètre
def f(x):
        return 2*x+1
def h(fonc, x):
        return fonc(x)
h(f, 3)

# Exemple : Nombre d'arguments arbitraire
def somme(*args) :
"""Renvoie la somme du tuple <args>."""
        resultat = 0
        for nombre in args :
        resultat += nombre
        return resultat
print(somme(23, 42, 13)) # 78
```

Temps d'exercices !
MOODLE TP: Utilisation de python basique I