Programmation orientée objet avec Python

Dany Siriphol – CODA - 2025

Présentation de Python

- Première version en 1991 par Guido van Rossum
- Utilisé dans de nombreux domaines
- Langage interprété
- Portable
- Multi paradigme
- Code plus concis
- Communauté active
- Grand nombre de librairies et framework :
  - Web : Flask, Django
  - Data Science : Pandas, numPy,
  - IA : TensorFlow ...
- Très utilisé en 2025

Nous utiliserons **Python 3** pour ce module

Objectifs du module

- Apprendre Python et ses spécificités
- Introduction à l’architecture logicielle
- Utiliser quelques librairies populaire de l’écosystème Python
- Devenir un meilleur programmeur.se

Format du cours

- 5 jours
- Cours le matin, Exos l’aprem (puis correction)
- Evaluation : Projet à faire en une semaine à rendre après la fin du module

Installation de Python

Python est un langage interprété donc il vous faut installer
un interpréteur pour pouvoir exécuter du code

Windows

- Télécharger l’installer de la dernière version de python :
  https://www.python.org/downloads/
- Vérifier que python se trouve dans votre variable d’environnement **PATH**

MacOS

```
brew install python
```

```
python -v
```

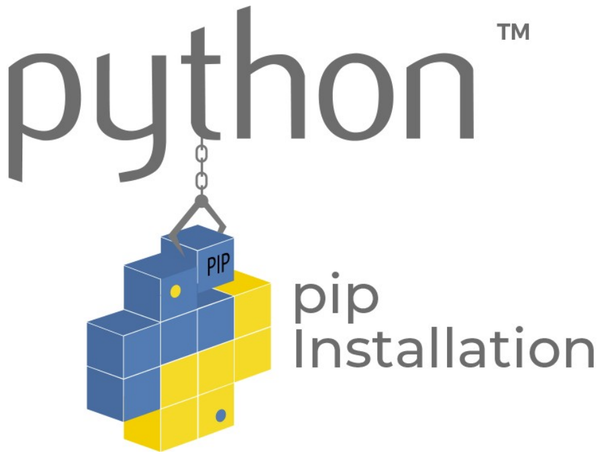
Linux

```
sudo apt update
sudo apt install python3
```

Installation de Python

Pour coder en Python, il est conseillé d’utiliser un IDE évolué comme Pycharm ou Visual studio code

Python dispose d’un gestionnaire de paquet nommé pip qu’il est recommandé d’installer *https://pypi.org/project/pip/*



Bases de Python

Dany Siriphol – CODA - 2025

Execution d’un script python

main.py

```
print("Hello World")
```

Terminal

```
python main.py
```

```
Hello World
```

Variables

```
age = 25
firstname = "John"
lastname = "Doe"
```

Affichage

```
print("prenom " + firstname)
```

Type hint

Python est un langage non typé. Cependant il est possible de déclarer les types des variables et des fonctions grace au **type hinting**

```python
answer: int = 42
PI: float = 3.14
language: str = "FR"
condition: bool = True # False
```

Mème si c’est facultatif, pour des gros projets il est recommandé de type hint le plus possible afin que votre IDE puisse autocompléter votre code et détecter les erreurs basiques plus facilement

Types en Python

Dany Siriphol – CODA - 2025

Types disponible en Python

- Texte : str
- Numérique : int, float
- Collections : list, tuple, dict
- bool
- None

...

Strings

```
text = 'Ceci est une string'
```

```
text = '''
Ceci est un texte
multiligne'''
```

```
print("prenom " + firstname)
```

Strings

```
print("prenom " + firstname)
print(f"prenom {firstname}")
```

```
text = f'''prenom : {firstname}
lastname : {lastname}'''

print(text)
```

Nombres

```
PI: float = 3.14
radius: float = 5.14

diameter = PI * radius * 2
```

- Opérateur : *, /, +, -, % ...
- Casting :

```
text = '5'
conversion = int(text)
```

```
PI: float = 3.14
number: int = int(PI)
```

Collections

List (array)

```
numbers = [5, 2, 10, 8]
```

```
modules = ['Python', 'Java', 'C']
```

Tuple

```
tuple = (1, 2, 3)
```

Set

```
set = {"apple", "banana", "cherry"}
```

Dictionnaire
(HashMap)

```
dict = {"firstname": "John", "age": 25}
```

Opération sur les listes

```
numbers = [5, 2, 10, 8]
```

Accéder à un élément

```
print(numbers[0])
```

Ajouter un élément

```
numbers.append(4)
```

Index négatif

```
numbers = [5, 2, 10, 8]

print(numbers[-1])  # 8
print(numbers[-2])  # 10
```

Obtenir un intervalle de la liste

```
print(numbers[1:3])  # 2, 10
```

Opération sur les listes

Concaténer 2 listes

```
list1 = [1, 2]
list2 = [3, 4]
print(list1 + list2)
```

Taille d’une liste

```
print(len(numbers))  # 4
```

Tester la présence d’un élément dans une liste

```
print(5 in numbers)
print(100 not in numbers)
```

Modules et import

Un module est un fichier du code python et qui peut être reutilisé dans un autre programme

- Il existe des modules python disponible par défaut : math, os, random
- Grace à **pip ou miniconda**, il est aussi possible de charger des modules venant de la communauté : numPy, pandas
- Il est aussi possible définir ses propres modules. Il suffit de créer un fichier .py avec des fonctions ou des classes à partager et de les mettre dans un dossier contenant un fichier nommé **__init__.py**

```python
def greet(name: str) -> str:
    return f"Hello, {name}!"
```

```python
import my_module


from my_module import greet
```

Structures de contrôle

Dany Siriphol – CODA - 2025

Structure de contrôle if

La structure de contrôle if évalue si une condition est **True** ou **False**. En fonction du résultat de la condition, un bloc de code spécifique est exécuté

```python
age: int = 25

if age >= 18:
    print("vous etes majeur")
else:
    print("vous etes mineur")
```

En python, **il est obligatoire d'indenter d'un cran les blocs de code** appartenant à une structure de contrôle

Les opérateurs de comparaison

Opérateur	*Description*
==	*Egal*
!=	Différent
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égal

```
int a = 10;

a > 10; // False
```

```
int a = 10;

a < 10; // False
```

Opérateurs logiques

Il est possible de combiner plusieurs conditions grâce aux opérateurs logiques **OR** et **AND**

```python
age: int = 25

if age >= 20 and age < 30:
    print("vous avez la vingtaine")
```

```python
language: str = "FR"

if language == 'FR' or language == 'EN':
    ...
```

OR gate truth table		
Input		Output
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

AND gate truth table		
Input		Output
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

if imbriqués

La structure de contrôle if évalue si une condition est True ou False. En fonction du résultat de la condition, un bloc de code spécifique est exécuté

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

```
if expression1:
    statement(s)
    if expression2:
        statement(s)
    else
        statement(s)
else:
    if expression3:
        statement(s)
    else:
        statement(s)
```

Les boucles

Dany Siriphol – CODA - 2025

Boucle while

Une boucle while permet de répéter un bloc de code tant qu’une condition est **True**

```
i: int = 0

while i < 5:
    print(str(i) + ' ')
    i += 1
```

Boucle for

Une boucle for permet de parcourir une collection et d’exécuter un traitement pour chacun des éléments

```
numbers = [5, 2, 10, 8]
sum = 0

for number in numbers:
    sum += number
```

```
set = {"apple", "banana", "cherry"}

for fruit in set:
    print(fruit)
```

```
for key, value in dict.items():
    print(f"{key}: {value}")
```

Boucle for

Contrairement à d’autre langage de programmation, la boucle for ne permet pas naturellement de répéter une instruction n fois.
Cependant, il est possible de reproduire ce comportement grâce à la fonction **range** qui génère une suite de chiffre dans une liste

```
for num in range(5):
    print(num)
```

```
for num in range(10, 20):
    print(num)
```

Fonctions

Dany Siriphol – CODA - 2025

Fonctions

Une fonction est un bloc de code réutilisable qui peut prendre des paramètres et peut retourner une valeur

```python
def say_hello() -> None:
    print("hello")
```

```python
def greet(name: str) -> str:
    return f"Hello, {name}!"
```

Fonctions

Une fonction est un bloc de code réutilisable qui peut prendre des paramètres et peut retourner une valeur

```python
def say_hello() -> None:
    print("hello")
```

```python
def greet(name: str) -> str:
    return f"Hello, {name}!"
```

Lire des données depuis le terminal

Dany Siriphol – CODA - 2025

Lire des données depuis le terminal

En mode interactif avec la fonction input

```
age: int = int(input("Enter your age: "))
print(f"You are {age} years old.")
```

En lisant les paramètres de la commande d’exécution

```
python main.py John Doe
```

```
import sys

for arg in sys.argv:
    print arg
```

Convention de codage python : PEP 8

Dany Siriphol – CODA - 2025

Norme PEP 8

- https://peps.python.org/pep-0008/

Quelques points clés :

- Indentation avec 4 espaces
- Ligne de code de 80 caractères max

- 2 sauts de lignes entre chaque définition de classe
- 1 saut de ligne entre chaque méthode

Programmation orientée objet avec Python 3 (POO)

Dany Siriphol – CODA - 2025

Programmation orientée objet

- Lisibilité
- Modularité
- Maintenabilité

POO : Concepts

Classe

Une classe est un type défini par le développeur pour représenter un concept et qui est composé d’un ensemble de variables (**attributs**) et de fonctions (**méthodes**)

```
class Person:
    ...
```

POO : Concepts

Attributs ou propriétés

Les classes sont composé d’attributs ou propriétés qui servent à représenter les caractéristiques d’une classe

En python, les attributs doivent être défini dans la méthode __init__ appelé le constructeur

```python
class Person:

    def __init__(self, firstname: str, age: int) -> None:
        self.firstname = firstname
        self.age = age
```

POO : Concepts

Instances

Une fois la classe définie, il faut créer des objets de cette classe pour pouvoir les utiliser. On appelle ces objets des **instances** d'une classe

L'instanciation d'une classe appelle la fonction __init qui est son **constructeur**. Son rôle est en général de définir les valeurs par défaut d'une instance

Depuis ces instances, il est possible d'utiliser leurs propriétés dans les algorithmes

```python
class Person:

    def __init__(self, firstname: str, age: int):
        self.firstname = firstname
        self.age = age

person = Person("John", 25)
print(person.age)
```

POO : Concepts

Méthodes

Une classe peut contenir des fonctions appelées méthodes : Leur but est d'attacher des fonctionnalités aux classes.

Pour utiliser une méthode, **il faut les appeler depuis une instance de la classe**

```python
class Person:

    def __init__(self, firstname: str, age: int) -> None:
        self.firstname = firstname
        self.age = age

    def increase_age(self) -> None:
        self.age += 1

person = Person("John", 25)
person.increase_age()
print(person.age) # 26
```

POO : Concepts

Mot clé self

Les méthodes doivent toujours avoir **en premier paramètre la variable self**.

self permet depuis le code d'une classe d'obtenir l'**instance actuelle.**

Il permet par exemple d'accéder aux propriétés ou aux méthodes de l'instance courante

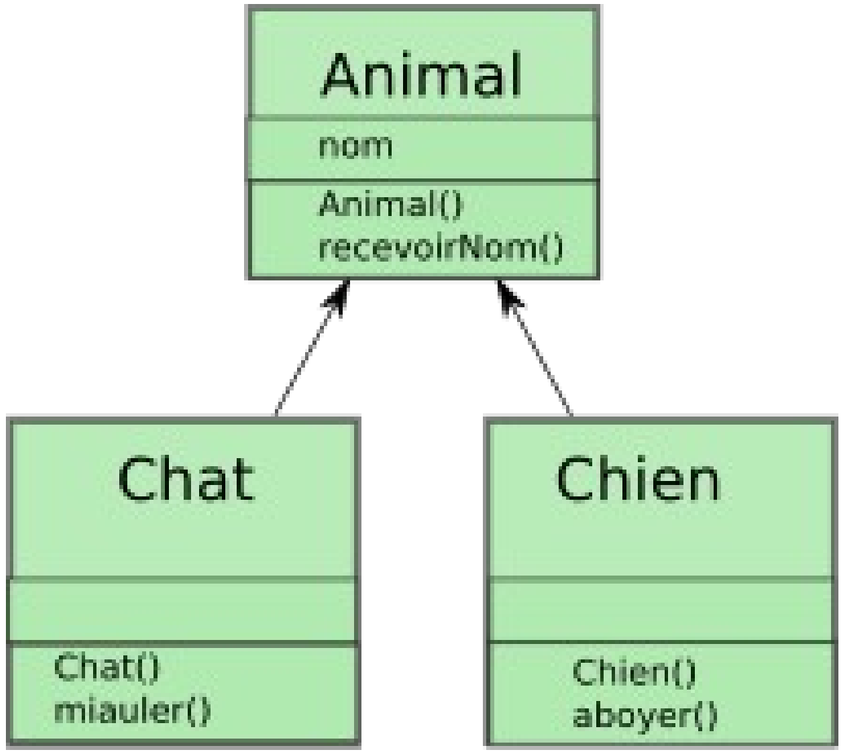
```python
class Person:

    def __init__(self, firstname: str, age: int) -> None:
        self.firstname = firstname
        self.age = age

    def increase_age(self) -> None:
        self.age += 1

person = Person("John", 25)
person.increase_age()
print(person.age) # 26
```

Héritage

Dany Siriphol – CODA - 2025

Héritage

Si certaines classes partagent un ensemble de fonctionnalités,
il est possible de les réutiliser en se servant de l’héritage



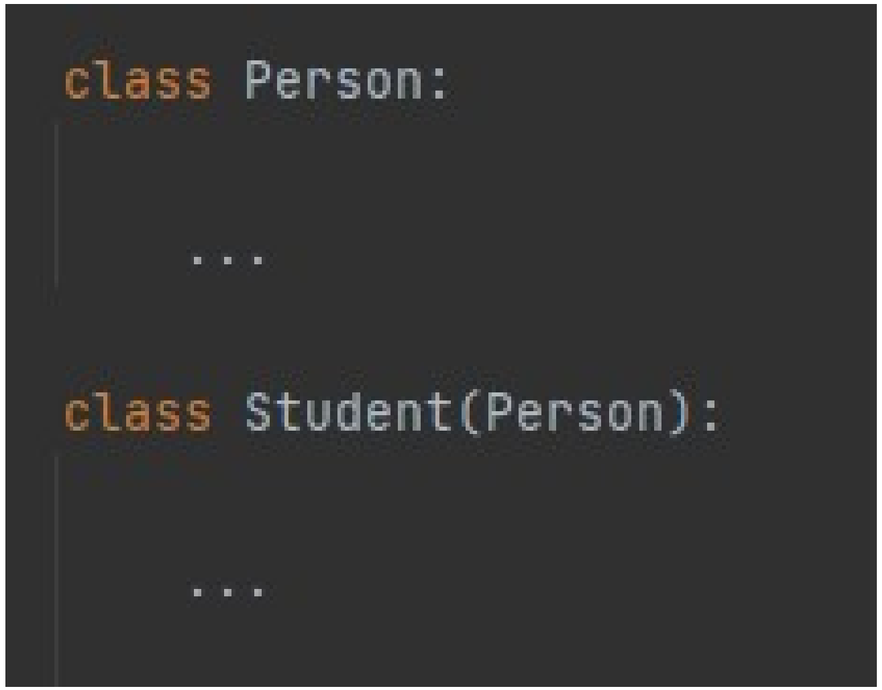
Héritage

Pour qu’une classe hérite d’une autre, il faut indiquer le **nom de la classe herité au niveau de la déclaration**

Ici, la classe Student hérite de la classe Person et va **avoir accès à toutes les méthodes et propriété de la classe Person**

Student est un **enfant** de la classe Person

Person est la classe **parent** de Student



Héritage

En héritant d’une classe, il est possible de lui rajouter des méthodes et des propriétés.

Il est courant dans le constructeur de la classe enfant de réutiliser celui du parent. La fonction **super()** permet d’accéder à la classe parent et d’appeler son constructeur

```python
class Person:

    def __init__(self, firstname: str, age: int) -> None:
        self.firstname = firstname
        self.age = age

class Student(Person):

    def __init(self, firstname: str, age: int, school: str) -> None:
        super().__init__(firstname, age)
        self.school = school
```

Héritage

```python
class Person:

    def __init__(self, firstname: str, age: int) -> None:
        self.firstname = firstname
        self.age = age

    def increase_age(self) -> None:
        self.age += 1

class Student(Person):

    def __init__(self, firstname: str, age: int, school: str) -> None:
        super().__init__(firstname, age)
        self.school = school

student = Student('John', 25, "Ecole multimedia")
student.increase_age()
```