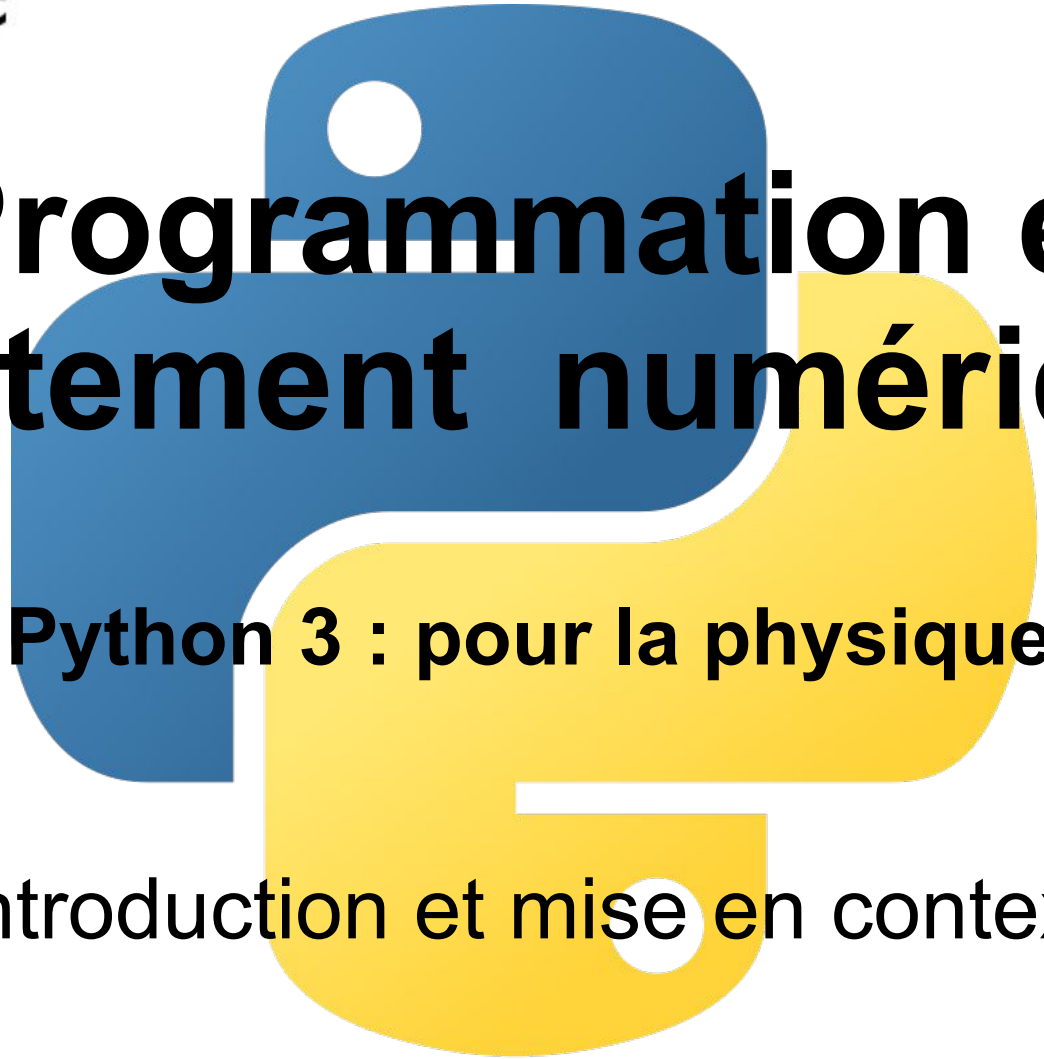
Université Paris Cité

2023/2024



Programmation et traitement numérique

Python 3 : pour la physique

Introduction et mise en contexte

FRERE Maxime

Objectif du cours

- Comprendre les clefs du scripting, codage et programmation.

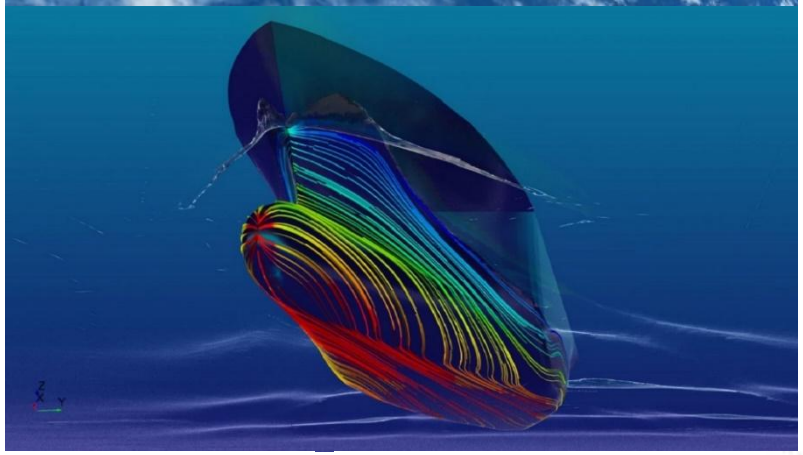
- Appliquer ces clefs aux résolutions de problèmes physiques :
  - Méthode d'intégration : Rectangle, Trapèze, Simpson.
  - Équation différentiel ordinaire (EDO): Euler, Runge-Kutta.
  - Analyse de courbe : Interpolation et déterminer la fonction d'une courbe.

- Pratiquer les outils : Projet de programmation

Modalités : 3 x 3 heures de Cours/TD et 4 x 3 heures consacré au projet.
Notation : (Examen intermédiaire + Examen final)/2 x 0,5 + Note Projet x 0,5
Projet : Projet à choisir, parmi la liste qui sera proposée vendredi. Code à rendre + rapport de projet.

Pourquoi la programmation ?

- Automatiser des acquisitions et traitement :
  - Communication avec capteur et stockage
  - Intégration de courbe
  - Traitement et transformation de données
  - Profiter des temps de « réactions » des machines
- Résoudre des problèmes non analytiques :
  - Equation différentielle
  - Equation différentielle avec propagation d'onde
- Effectuer des simulations multiparamétriques :
  - Simulation thermique
  - Equation de Navier-Stockes



Rappel sur comment compte les machines

Binaire :
Utilisateur -> Machines Actuelles

Comptage base deux pour des raisons de conception, les ordinateurs n'ayant que les états fondamentaux 0 et 1.

Décimal :
Utilisateur -> Civilisation actuelle

Comptage en base dix, l'humain ayant dix doigts.

Quaternaire :
Utilisateur -> Shadocks

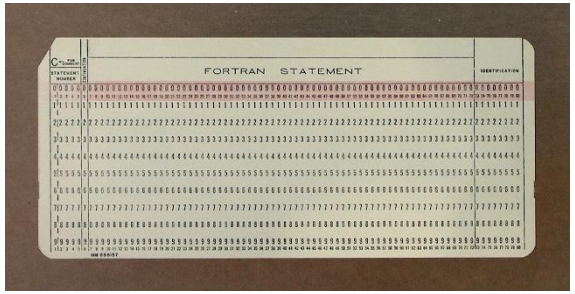
Les Shadocks n'ayant que 4 mots : Ga, Bu, Zo et Meu. Ces derniers sont forcés d'utiliser cette base pour compter au-delà de quatre.

Dozénale :
Utilisateur –> Civilisation Grec antique

Comptage à base des phalange de chaque doigts pouce exclu. Plus pratique pour effectuer des divisions et multiplications de tête (2, 3, 6…).

Table des bases de comptages (2 à 20): https://fr.wikipedia.org/wiki/Table_des_bases

L’histoire de la programmation



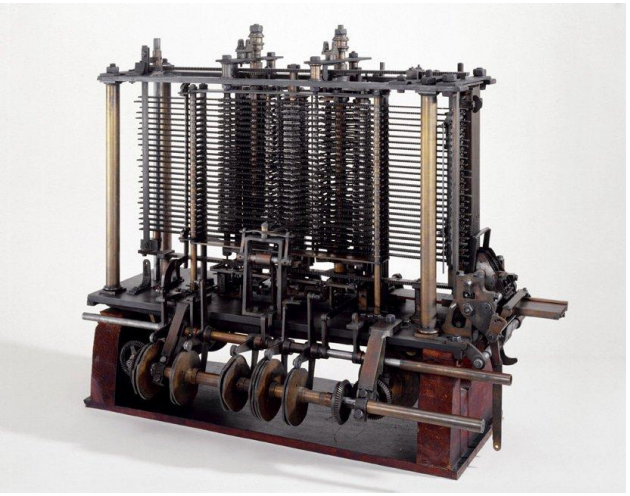
Naissance de la programmation
1836

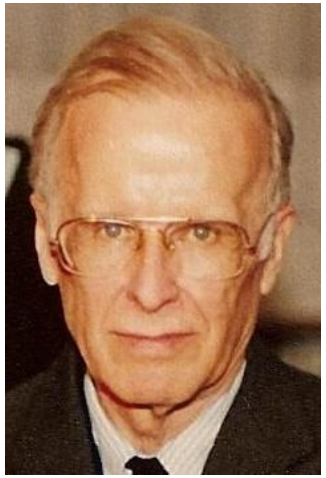
Les balbutiements
1940-1960

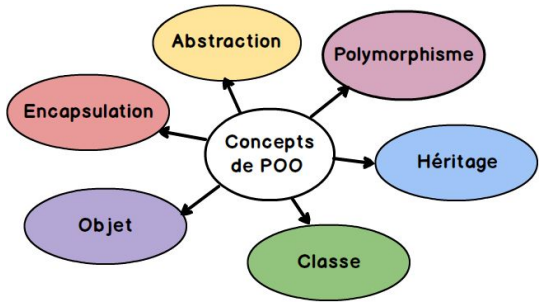
Structuration et paradigme
1960-1980

L’ère d’internet
1980-2000

L’ère de la sécurisation
2000-Aujourd’hui







Architecture de la programmation

Les bas niveaux :

- Basic
- C/C#/C++

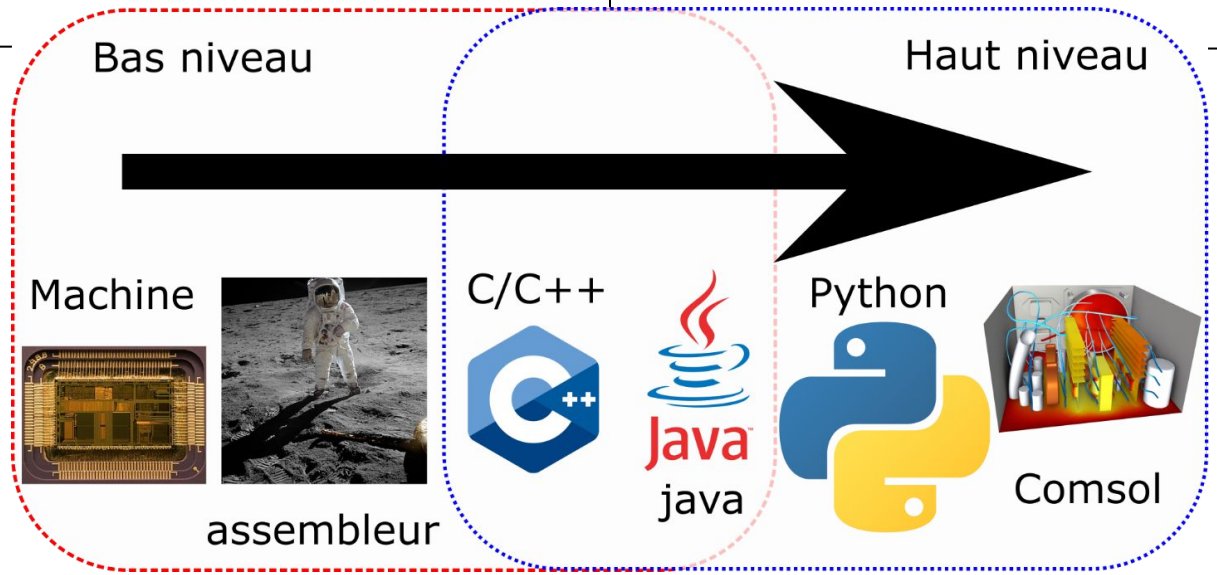
- **Spécificités :**
  - Allocation manuelle des espaces mémoires et thread d'exécution
  - Peu de surcouche, proche du langage des machines

Les hauts niveaux :

- Python, matlab, octave
- Wolfram

- **Spécificités :**
  - Proche du langage humain
  - Beaucoup de surcouches, loin du langage des machines. Besoin de machines puissantes.

Avantages :
Optimisation avancée possible
Inconvénients :
Plus complexe et long à développer



Avantages :
Simple pour l'humain
Inconvénients :
Complexe à optimiser

Pourquoi python ?

- Installable sur toutes les machines (PC, Mac et Linux)
- Langage de haut niveau :
  - Adapté à la sciences et au traitement de données
  - Tout en étant flexible et polyvalent
  - Très tolérant sur les manières d’écrire
  - Orienté Objet
- Communauté active et varié (Particulièrement en science)
- Moins lourd que Java à lancer
- Gratuit et toujours en développement

Les librairies

- Les librairies que vous allez rencontrer :
  - **Numpy :** Base du traitement de data et de matrice
  - **Matplotlib :** Base pour le tracer de courbe
  - **Scipy :** Boite à outils complète pour quasiment tous les domaines de mathématique, physique et chimie.
  - **Pandas :** Outils de gestion et de traitement de données temporelle (Station météo) et autre matrice complexe
- Une infinité de boites à outils déjà existante :
  - Avant de programmer une fonction/méthode, toujours vérifier si une solution n'a pas déjà été réalisée en cherchant une librairie dédiée.

Installer python et ces indispensables

Gestionnaires de paquet/librairies :

- ***Windows* :** Anaconda
- ***Linux :*** apt-get ou pip
- ***Mac :*** IDLE

Les IDE :

- ***Pycharm*** : IDE python
- ***Spyder*** : IDE physique/python
- ***Visual Studio Code :*** IDE tout langage

Suivi de version :

- ***Git :*** Programme de gestion de suivie de version. Exécutable localement.
- ***GitHub :*** serveur distant utilisant Git
- ***GitLab :*** serveur distant utilisant Git

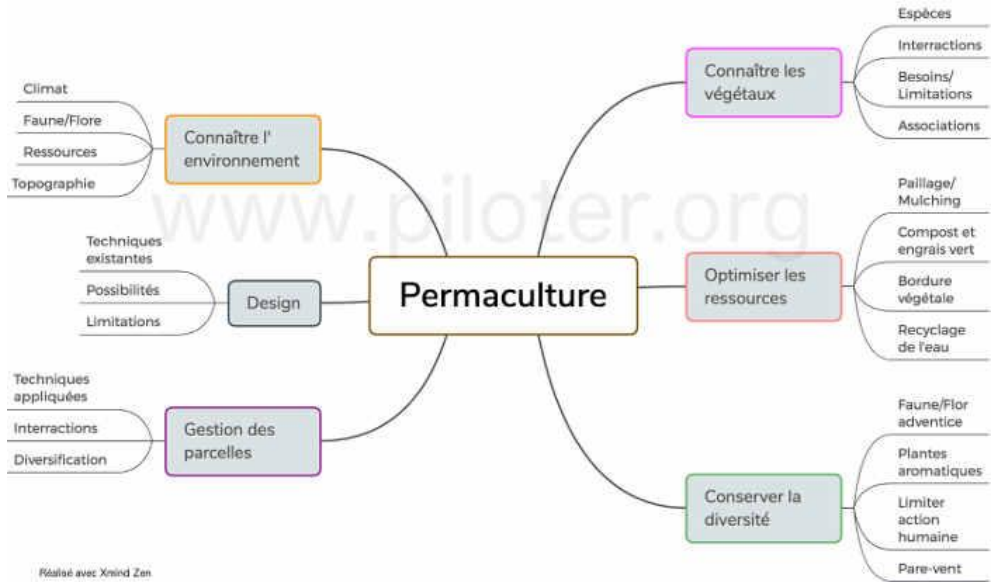
Autre :

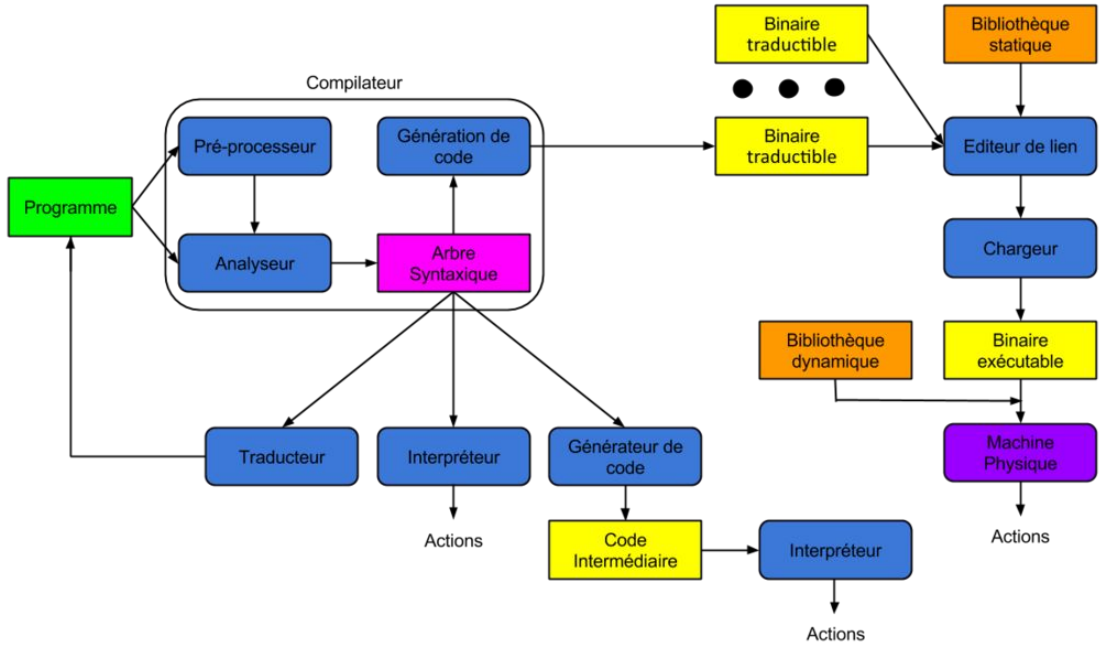
- ***Jupyter :*** éditeur dynamique de code, avec insertion de texte. Pratique pour expérimenter, expliquer et apprendre le langage.

Bien conduire un projet [1/2]

Avant de coder :

- Comprendre la problématique et la segmenter en tâches simples successives.
- Poser à l'écrit cette compréhension et listes de tâches.
- Rendre le problème adimensionnel et unitaire.
- Choisir son langage en fonction des contraintes du projet.
- Trouver les librairies susceptibles de répondre aux taches segmentées.





Bien conduire un projet [2/2]

Pendant le codage :

- Se fixer des règles de structure et d'écriture, sans en changer. (Voir PEP 8)
- Nommer intelligemment les variables, fonctions, objets et méthode. (Voir PEP 8)
- Commenter son code en détail. (Voir PEP 8)
- Utiliser un gestionnaire de version (Git …)
- Traduire les taches en termes compréhensibles pour la machine.

Pendant/Après Codage :

- Tester et vérifier la fiabilité des résultats.
- Déboguer et corriger.
- Optimiser le programme et les rendus.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Short descriptive sentance.
"""

# == Import Space == #
import numpy as np
import matplotlib.pyplot as plt

# == Global Var/stat == #
SIGMA = 5.67037e-8  # W/m²K**4


# == Function/Class Space == #
def flux_radiative_from_temperature(temperature, emissivity=1):
    """Will compute the radiative flux from black body law

    Args:
        temperature (array): The given temperature in Kelvin
        emissivity (array): The emissivity of the surface

    Returns:
        flux (array): The resulting radiative flux (W)
    """

    flux = emissivity * SIGMA * temperature ** 4
    return flux


# == Running Script Space == #
if __name__ == '__main__':
    array_temperature = np.arange(300, 500, 1.0)

    figure = plt.figure()
    plt.title("Flux d'émission d'un corps noir")
    plt.plot(array_temperature,
             flux_radiative_from_temperature(array_temperature),
             label="Corps noir")
    plt.xlabel("Température (K)")
    plt.ylabel("Flux radiatif (W)")
    plt.grid()
    plt.show()
```

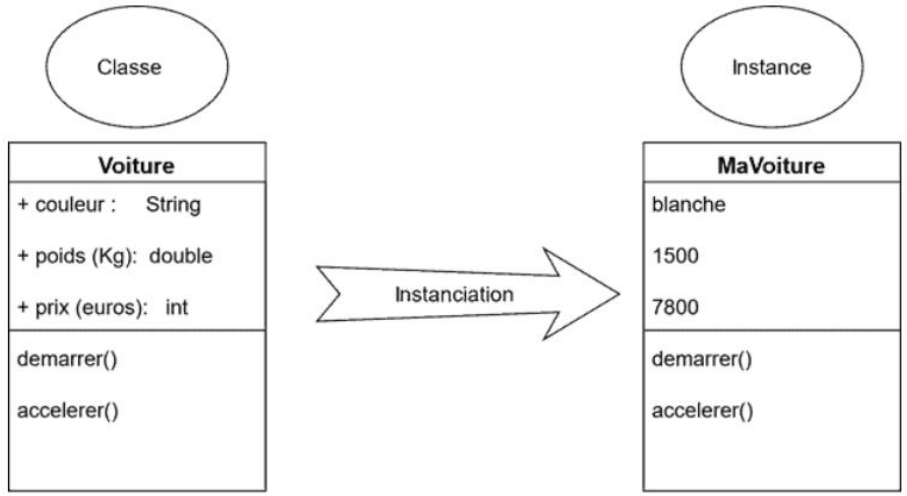
Comprendre la pensée machine : les objets

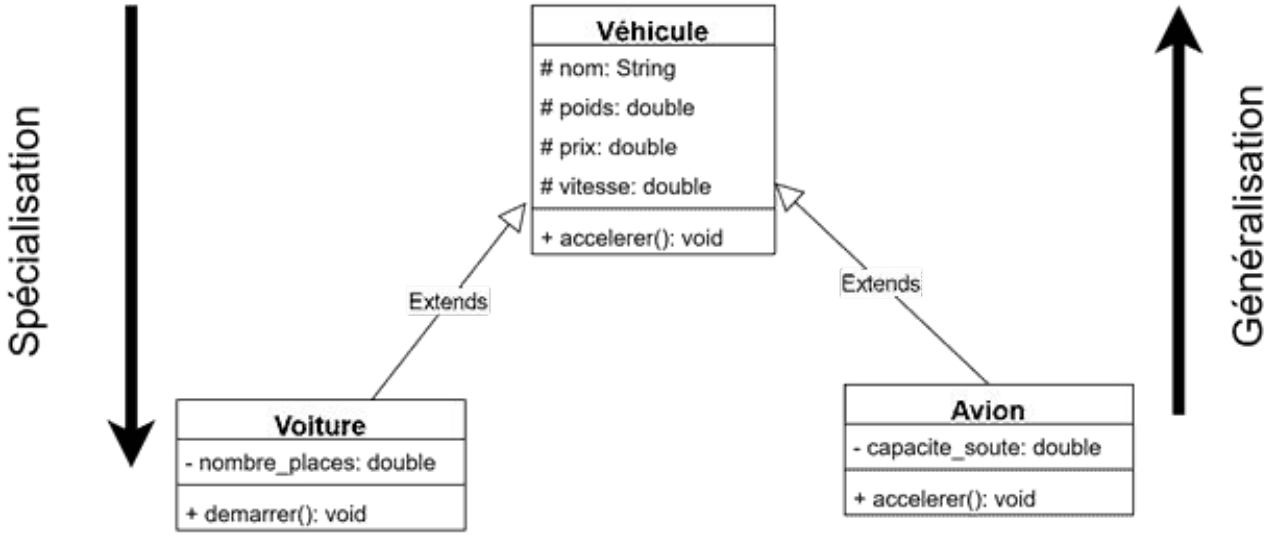
La nature des objets natifs :

- **Bool :** 0 ou 1, False ou True
- **Int :** Entier
- **Float :** Nombres décimaux
- **Str :** Chaines de caractère
- **Tuple :** Couple de données
- **List :** Liste, index par ordre
- **Dict :** Dictionnaire, index par Key

Les objets de librairies :

- **Array :** Tableau mathématique
- **DataFrame :** Tableau gestion de données
- **Figure :** Objet de graphiques
- **Axes :** Objet des axes du graphique
- Bien d'autres…





Comprendre la pensée machine : les opérateurs

Chaque opérateur va avoir un comportement **différent** en **fonction de la nature des objets** qu'il traite.

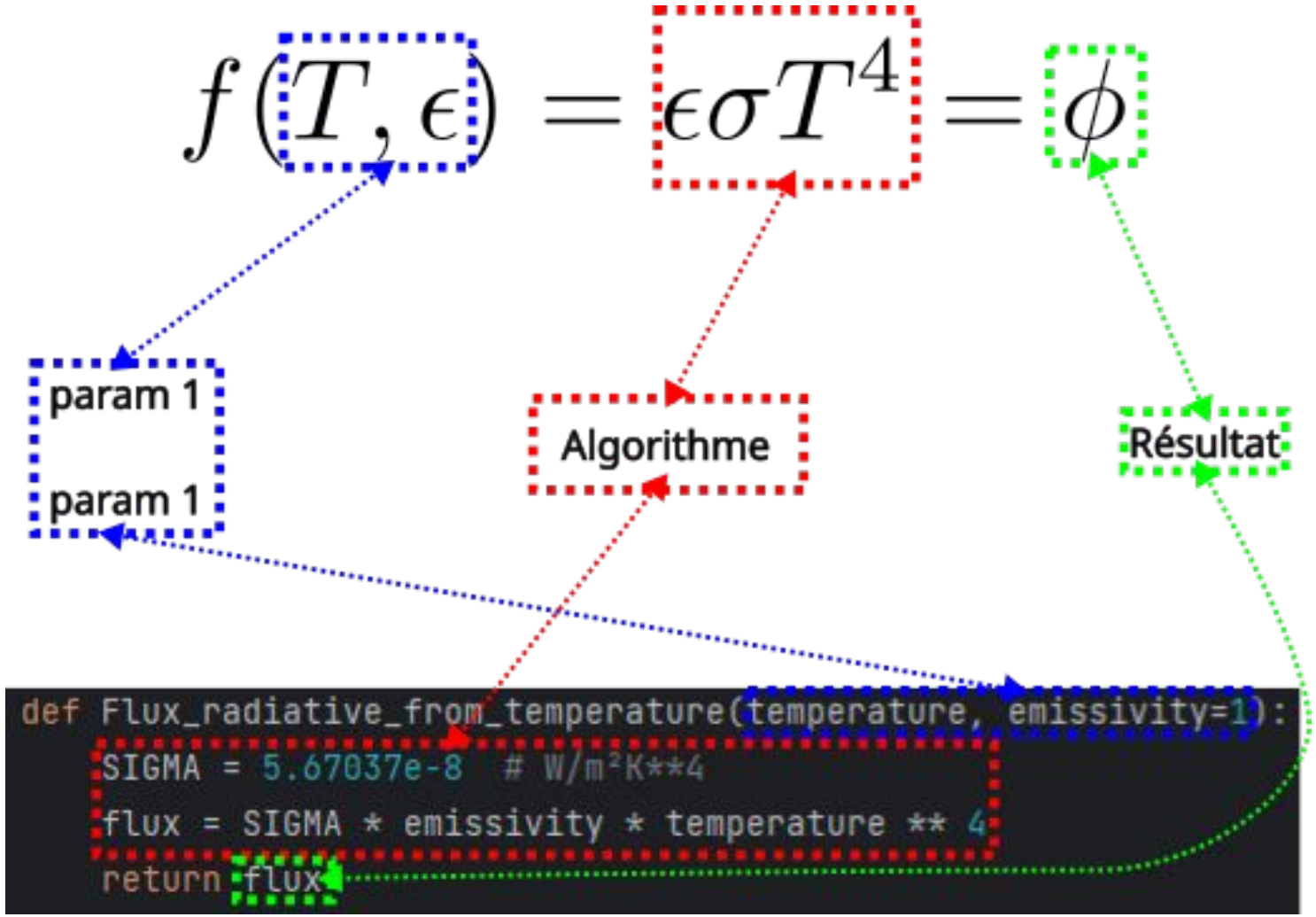
Les opérateurs mathématiques :

- **+ et - :** *additionnent et soustraient* pour les **int**, **float** et **array**. Mais, *concatène* ou fait la *différence* pour les **list**
- *** et ** :** correspond à la *multiplication* et à la *puissance.*
- **/ :** Effectue la *division rationnelle.* Résultats différents sur **int** et **float**
- **// et % :** Outils de *division euclidienne*, donne *la partie entière* et *le reste*
- **= :** Attribution de variables

Les Comparateurs :
<,> et <=, >= : vérifie inf/sup et inf/sup ou égal, renvoi un bolléen.
== : Vérifie l'égalité, renvoie un bolléen
!= : Vérifie la différencre, renvoie un bolléen

Les opérateurs booléens :
OR : renvoie vrai si un des éléments est vrai
AND : renvoie vrai si tous les éléments son vrai
XOR : comme **OR**, mais renvoie Faux si tous vrais
NOR : Opposé de **OR**

Les fonctions et méthodes



Penser par bloc

Input :

Éléments de départ pour résoudre notre problématique.

Algorithme :

Ensemble des opérations et processus à effectuer pour résoudre la problématique.

Output :

Résultat voulu.