Algorithmique et Programmation

Semestres 5 et 6

ENSG

Introduction

Pourquoi étudier l’algorithmique et la programmation ?

Algorithmique vs **Programmation**

- Un algorithme est une procédure **étape par étape** pour résoudre un problème.
- Indépendant du langage de programmation
- Principe fondamental : diviser pour mieux régner

- La programmation consiste à traduire un algorithme dans un langage de programmation (e.g. Python)
- Nécessite une bonne connaissance du langage (grammaire & vocabulaire)

UE : Algorithmique et Programmation

AF1 : Algorithmique
AF2 : Programmation

Quel intérêt ?

Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l’information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

Quel intérêt ?

Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l'information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

Automatiser

- *Economiser de la ressource (humaine)*
- *Eviter les tâches répétitives*

Piezometers installed with dataloggers at different construction sites

Quel intérêt ?

Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l'information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

SIG (S6)

TRAITEMENT STATISTIQUE (S7)

TRAITEMENT DU SIGNAL (S5)
GEOPHYSIQUE (S7)

ANALYSE NUMÉRIQUE (S5)

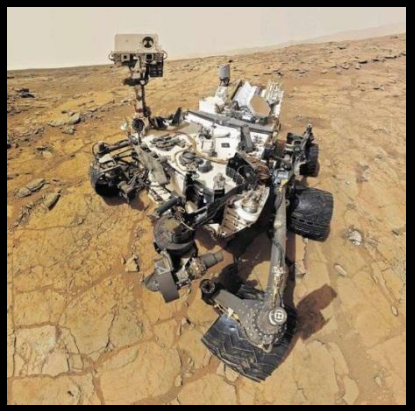
Quel intérêt ?

Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l'information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

Monitoring automatisé

Curiosity



DB™ Undergroung Drone for Mining Exploration and Mapping

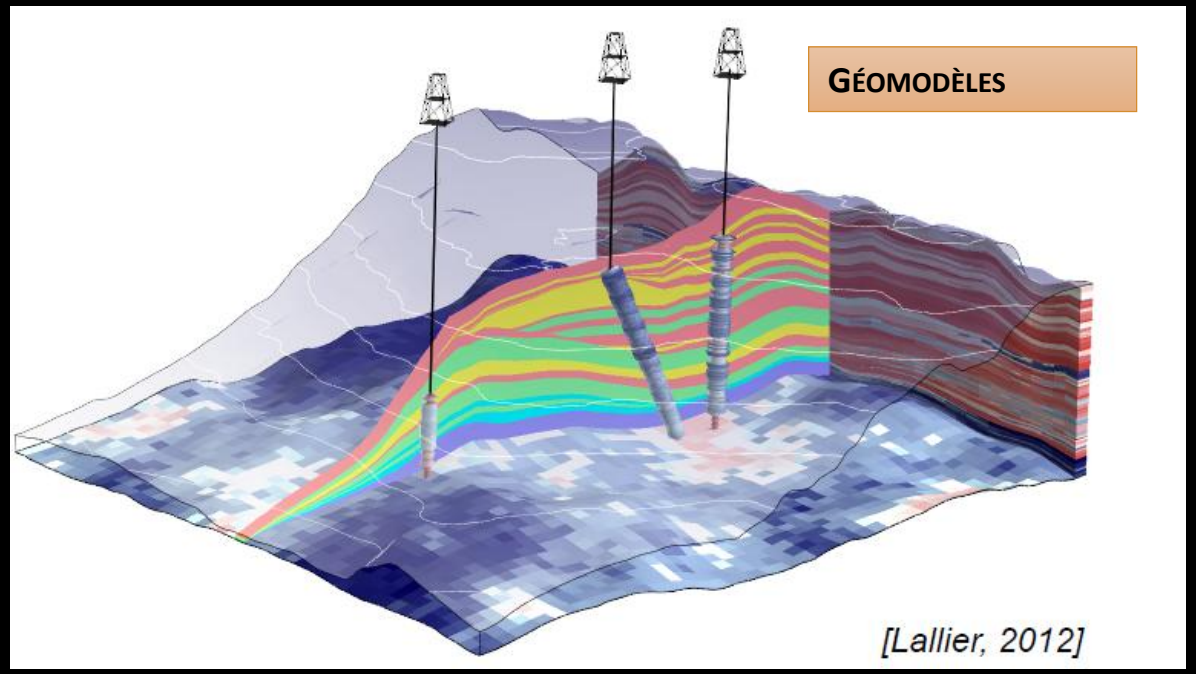
Atteindre des lieux difficiles / impossibles à explorer par l'homme

Quel intérêt ?

Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l'information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

Modéliser ce que l'on ne peut voir



Quel intérêt ?

Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l'information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

Comprendre le passé

Quel intérêt ?

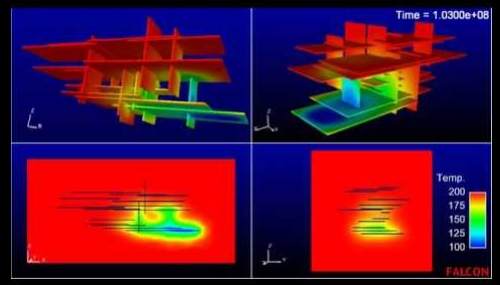
Le numérique a une part de plus en plus importante dans le monde professionnel : **Transition Numérique**

- Automatisation de l'information
- Traitement des données
- Commandes à distance
- Modélisation numérique
- Intelligence artificielle

Tester des scénarios futurs

Transfert de polluants (www.simmakers.com)

Avalanche sous-marine, V. Topin et al. *Phys. Rev. Lett*. (2012)



Géothermie, (Idaho National Lab. – youtube.com)

Quel intérêt ?

N'existe-t-il pas déjà des logiciels pour faire tout ça?

Oui, mais :

- Il faut **comprendre** le logiciel pour l'utiliser à bon escient (son fonctionnement, ses limites, etc.)
- Un logiciel ne répondra pas toujours à une demande particulière
- Certains logiciels nécessitent de la programmation pour être **complétés** / **modifiés** / **adaptés**
- Pour des **applications simples**, un logiciel peut être trop coûteux par rapport au développement en interne et selon les besoins de l'entreprise
- Il faut pouvoir **collaborer** avec des spécialistes

Quel intérêt ?

L'IA peut programmer pour moi, pourquoi apprendre?

- L'IA peut faire des **erreurs**. Il faut pouvoir les comprendre et les corriger. Il est aussi nécessaire de savoir **évaluer** un code.

- L'algorithmique c'est aussi une approche de résolution de problème, applicable dans d'autres contextes que la programmation (e.g. gestion, etc.). Cela apprend à structurer ses idées, subdiviser les problèmes, établir des liens cause – à – effet, etc. Il s'agit donc aussi d'une **compétence transversale** (soft-skill).

- L'IA est outil, pas une remplaçante. Il faut savoir poser la bonne question (**prompt engineering**). Il faut aussi savoir faire sans si nécessaire (non accessible, non autorisé, etc.). Nécessité d'avoir un socle de **compétences techniques**.

- Pour **créer**, il faut comprendre.

- Pour **collaborer**, il faut comprendre.

- Ethique et responsabilité : Vous avez une **responsabilité** (éthique, mais possiblement aussi légale) des travaux que vous fournissez. L'IA présente un coût environnemental important, il s'agit donc de l'utiliser à bon escient. Quid de la **propriété intellectuelle**.

Quel intérêt ?

L'IA peut programmer pour moi, pourquoi apprendre ?

🧩 5. Travailler avec l'IA, c'est mieux quand on connaît le code

Les développeurs qui savent utiliser l'IA **ne codent pas moins** :

ils codent **plus vite**, **plus intelligemment**, avec **moins d'erreurs**.

👉 Ceux qui combinent **compétence humaine** + **IA** seront les plus efficaces.

Réponse de ChatGPT

Contenu

Semestre 5

AF1 : Algorithmique

- Répondre de manière adaptée à un cahier des charges
- Développer un algorithme simple
- Structurer son raisonnement et son programme
- Définir un jeu de validation pertinent
- Rechercher l'optimisation de l'algorithme

AF2 : Programmation

- Appliquer les fondamentaux (boucles, conditionnelles, fonctions, types de données de base, i/o, etc.)
- Structurer ses données de manière adaptée (liste, dictionnaire, etc.)
- Appliquer les bonnes pratiques de programmation
- Rechercher l'efficacité de la programmation

Semestre 6

AF1 : Algorithmique

- Répondre de manière adaptée à un cahier des charges
- Développer un algorithme simple
- Structurer son raisonnement et son programme *par une approche orientée objet*
- Définir un jeu de validation pertinent
- Rechercher l'optimisation de l'algorithme

AF2 : Programmation

- Appliquer correctement la *programmation orientée objet*
- Structurer ses données de manière adaptée avec une *approche orientée objet*
- Appliquer les bonnes pratiques de programmation
- Rechercher l'efficacité de la programmation

Contenu

Semestre 5

1	Introduction	
2	Fondamentaux	Programmation
3		Algorithmique
4	Chaînes de caractères	Programmation
5		Algorithmique
6	Listes	Programmation
7		Algorithmique
8	Dictionnaires	Programmation
9		Algorithmique
10	Récapitulatif : Annales	

Evaluation :

1) ~~Contrôle continu : quizz et projet~~
2) **Examen final : 1h – sur feuille**

Ressources

UNIVERSITÉ DE LORRAINE | ARCHE

Distribution Python

ANACONDA.

IDE

Spyder

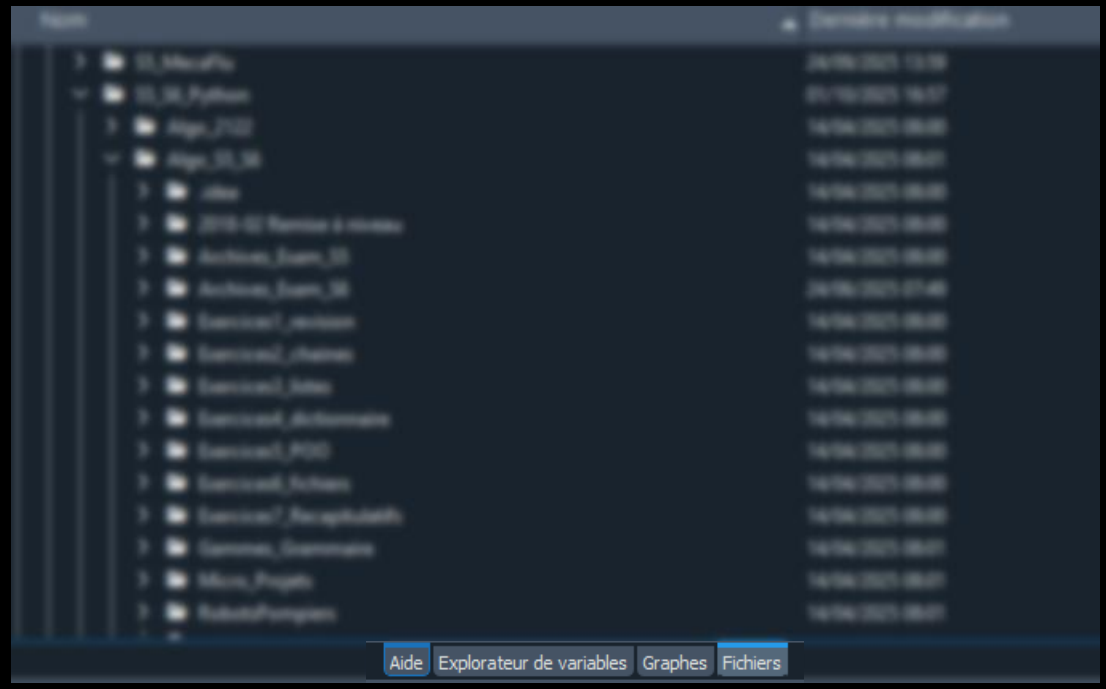
Documentation en ligne

https://www.python.org/doc/

Ressources

Disque réseau : **O:** Accessible sur tout les ordinateurs de l'école et depuis l'ENT

Créer un **répertoire** Algorithmique. Vous pourrez créer des sous-répertoires par thématique.



Bonnes pratiques

Donnez des adjectifs décrivant un « bon » code ?

Bonnes pratiques

Fonctionnel

Fiable

- Il n’y a pas d’erreurs de compilation / d’exécution
- Tous les cas sont gérés
- Répond au cahier des charges (paramètres d’entrée, information en sortie)

Les tests de validation

Le débuggeur

Compilation pas à pas : *e.g.* **PythonTutor**

Bonnes pratiques

Fonctionnel

Fiable

Contre-exemple

```
def saluer(nom)
    print("Bonjour, " + nom)


saluer("Anne"
```

```
def est_premier(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
        else:
            return True
```

Bonnes pratiques

Fonctionnel

Fiable

Bonne version à compléter

Bonnes pratiques

Fonctionnel

Fiable

Les tests de validation

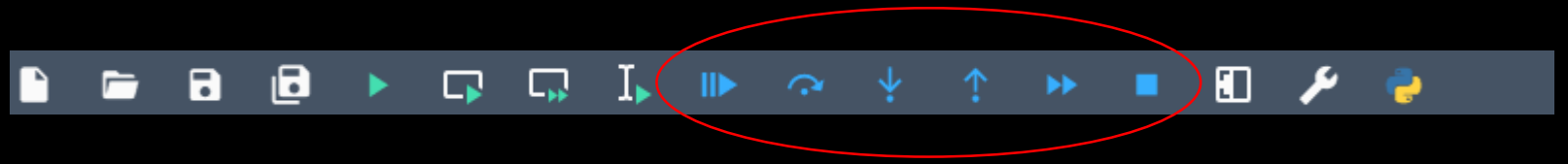
- ✓ Faire des « petits » tests tôt et régulièrement :
  - ✓ Il vaut mieux tester des petits éléments de code que de tout tester à la fin
  - ✓ Essayer d'écrire vos tests avant votre code
- ✓ Chaque test doit valider un unique comportement
- ✓ Penser à tester les cas limites, en particulier les « modes d'échec » (e.g. division par 0)
- ✓ Conserver les informations liées à vos tests (=> Programme principal)

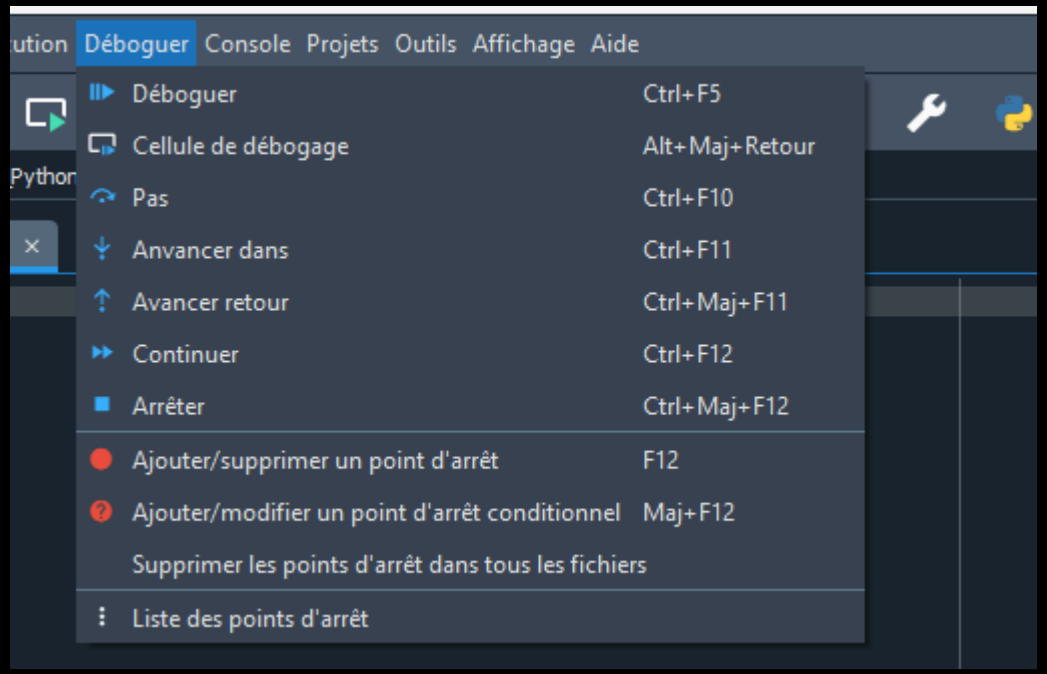
Bonnes pratiques

Fonctionnel

Fiable

Le débugger





Bonnes pratiques

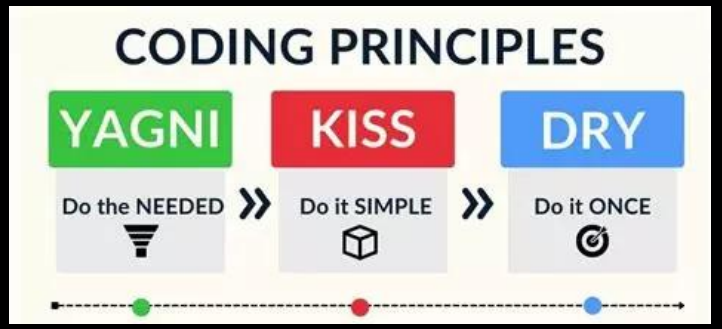
Compréhensible

Simple

- Respecter le principe KISS
- Eviter les structure inutiles = Respecter le principe YAGNI
- Préférer la clarté à la concision extrême
- L'obfuscation

Code obfusqué

```
def O(l):
 return sum([int(chr(ord(c)^42)) for c in list(map(chr,[ord(x)^42 for x in l])) if c.isdigit()])

print(O("5;4"))
```



Version simple

```
def somme_chiffres(donnees):
    return sum(int(c) for c in donnees if c.isdigit())

print(somme_chiffres("5;4"))  # 9
```

Bonnes pratiques

Compréhensible

Simple

Contre-exemples

```python
def somme_liste(liste):

    total = 0
    index = 0
    longueur = len(liste)

    while index < longueur:
        element = liste[index]
        total = total + element
        index += 1

    return total

# Utilisation
nombres = [1, 2, 3, 4, 5]
print(somme_liste(nombres))   # 15
```

```python
def additionner(a, b):

    nombres = []
    nombres.append(a)
    nombres.append(b)
    total = sum(nombres)
    return total

# Utilisation
print(additionner(3, 4))  # 7
```

Bonnes pratiques

Compréhensible

Simple

Bonne version à compléter

Bonnes pratiques

Lisible

Le PEP8
https://www.python.org/dev/peps/pep-0008/

L'autoformateur

- Longueur de ligne : on évite les lignes trop longues pour permettre de toute lire d'un seul tenant (utilisation de \)
- Lignes blanches : on aère le code, permet de repérer rapidement les blocs d'information
- Convention de nommage : donne de la consistance, permet de comprendre rapidement le rôle des fonctions, paramètres, etc.
- Eviter les « nombres magiques » : Remplacer par des paramètres

Bonnes pratiques

Lisible

Contre-exemples

```
# Bad Practice
area = length * 3.14

# Good Practice
PI = 3.14
area = length * PI
```

```
def calc(a, b, c):
    return a + b - c
```

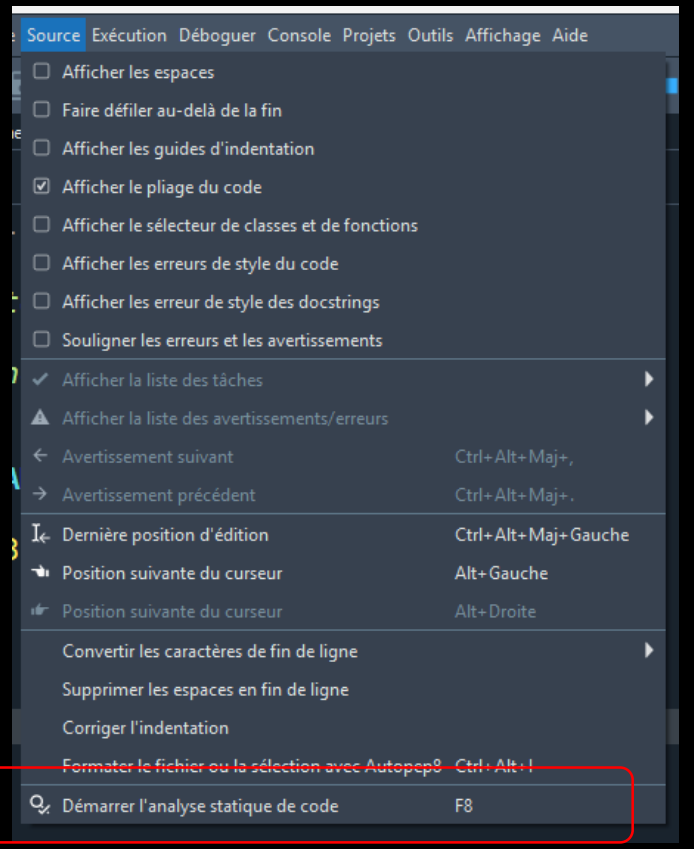
```
def ADD(a,b):print(a+b)

x=  3
y =4
ADD( x ,y )
```

```
import math, sys


def CalculeVolumeSphere(R):

    volume=4/3*mzth.pi*R**3
    print( "Le volume est:",volume )



def main( ):

  radius =10
  CalculeVolumeSphere( radius )

print("Fin du programme")
```

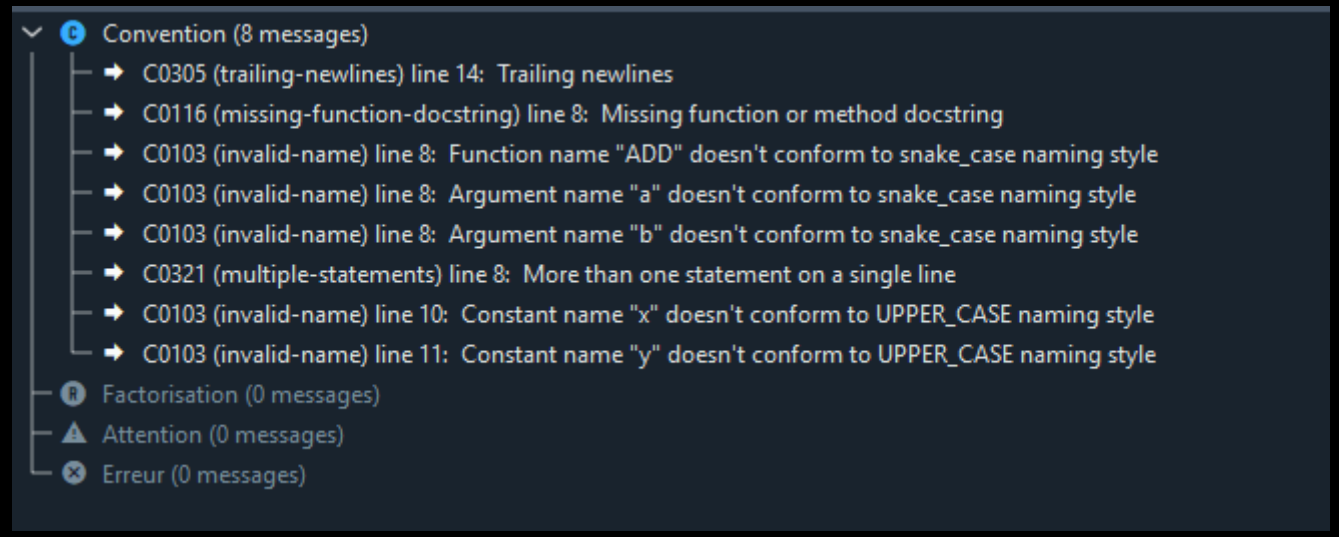
Bonnes pratiques

Lisible

Contre-exemples



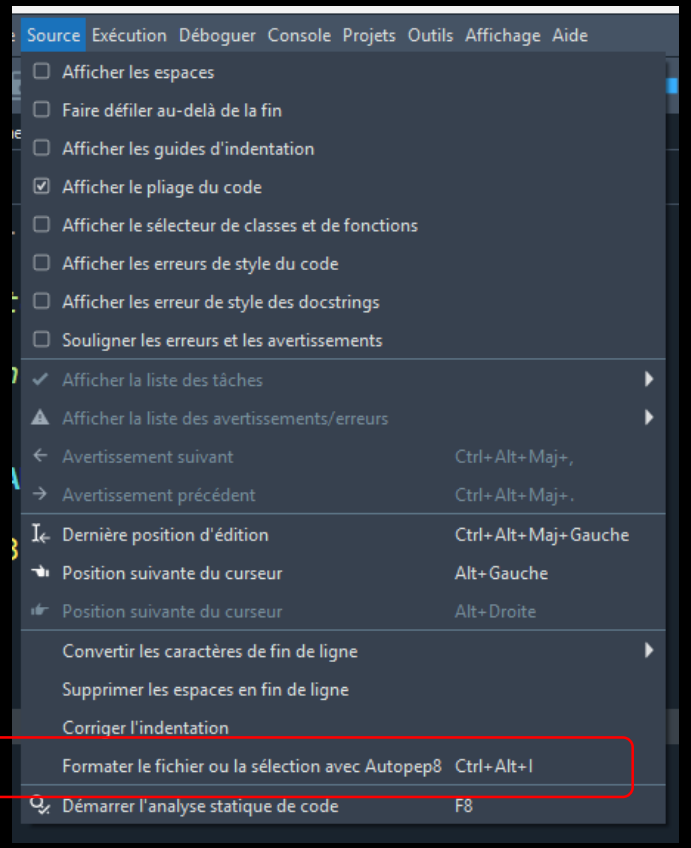

```python
def ADD(a,b):print(a+b)

x=  3
y =4
ADD( x ,y )
```



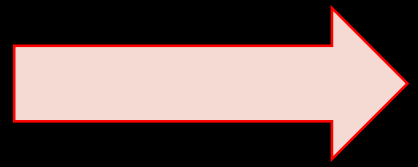
Bonnes pratiques

Lisible

Contre-exemples




```
def ADD(a,b):print(a+b)

x=  3
y =4
ADD( x ,y )
```



```
def ADD(a, b): print(a+b)


x = 3
y = 4
ADD(x, y)
```

Bonnes pratiques

Compréhensible

Simple

Bonne version à compléter

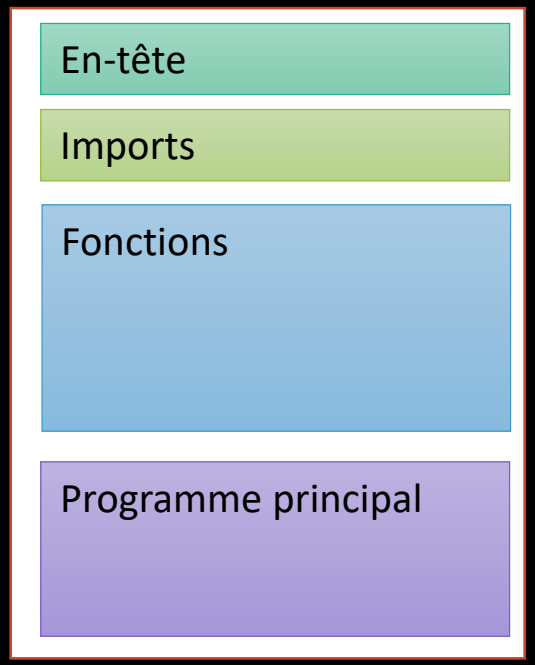
Bonnes pratiques

Maintenable

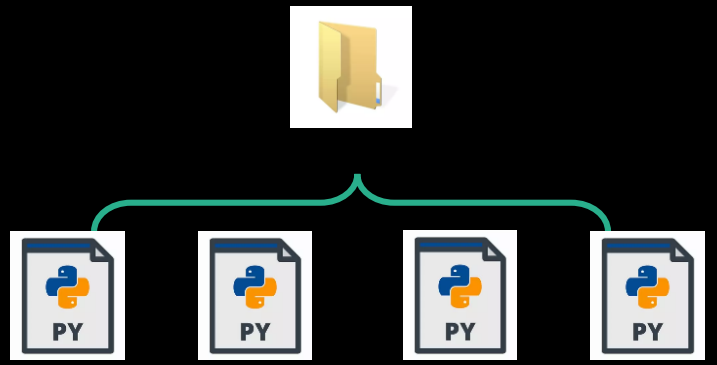
Evolutif

- Structure
- Modularité : Chaque fonction fait **une** action bien !

Fichier .py



Dossier contenant plusieurs scripts :



Dont un script principal (e.g. main.py)

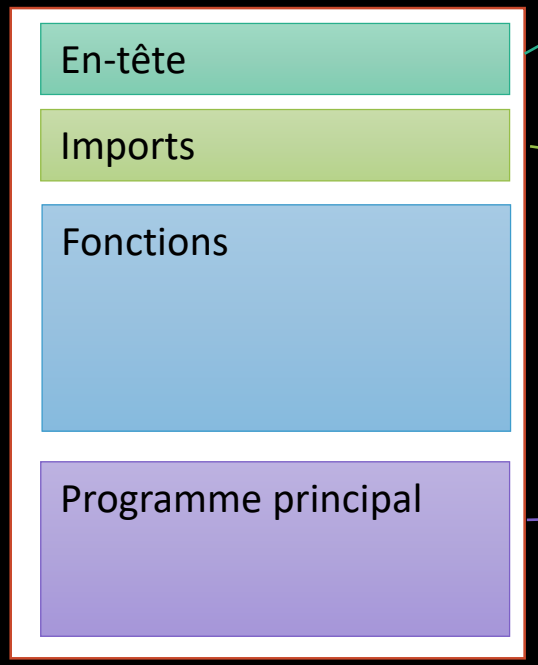
Programme plus complexe : plusieurs niveaux

Bonnes pratiques

Maintenable

Evolutif

Fichier .py



Contient :

Le nom du projet, le nom de l'auteur, le nom du fichier, la date et toute information générale pertinente

```
# -*- coding:utf-8 -*-

"""
__nom_fichier__ = "BouclesFor_11a16"
__author__ = "tinetl"
__date__ = "mars 2018"


Exercices - Boucles For
```

Imports :

Bibliothèques (e.g. random, math, numpy, etc.) ou autres scripts/fichiers (sans l'extension .py)

Indiqué par : if __name__ == '__main__':

Contient :

Les tests de validation des fonctions du fichier
L'application (si 1 seul fichier, sinon dans main.py)

Bonnes pratiques

Maintenable

Evolutif

Contre-exemples

```python
def process_data(data):

    # Filter data
    data = [x for x in data if x > 0]

    # Calculate average
    average = sum(data) / len(data)

    # Print results
    print("Average : ", average)

    return data
```

```python
def show_details(product):
    print("Name : ", product[0])
    print("Price : ", product[1])
    print("Category : ", product[2])


def show_summary(product):
    print("Name : ", product[0])
    print("Price : ", product[1])
```

Bonnes pratiques

Maintenable

Evolutif

Bonne version à compléter

Bonnes pratiques

Collaboratif

Commentaires:

- ✓ Pas de commentaires qui contredisent le code
- ✓ Un commentaire est une phrase complète
- ✓ Les commentaires sont clairs et facilement compréhensibles
- ✓ Commentaires en bloc : Expliquent le code qui suit
- ✓ Commentaires en ligne : Expliquent le code sur la ligne

Documentation: """…"""

- ✓ Docstring = Chaîne de caractères qui arrive en début de module, fonction, etc.
- ✓ Permet de générer automatiquement la documentation
- ✓ Accessible avec help
- ✓ Convention des docstring : PEP 257 (https://peps.python.org/pep-0257/)
- ✓ Pour une fonction : **Rôle de la fonction, paramètres d'entrée, sortie**

Bonnes pratiques

Collaboratif

Commentaires: #

```
# Filter data
```

Documentation: """..."""

```
def process_data(data):
    """


    Parameters
    ----------
    data : TYPE
        DESCRIPTION.

    Returns
    -------
    data : TYPE
        DESCRIPTION.

    """
```

Bonnes pratiques

Performant

Efficace

Zen of Python

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```