



# CHAPITRE 1

## TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

1- Critères de Choix d'un langage de programmation

2- Blocs d'instructions

3-Conversion de l'algorithme en Python

4- Optimisation du code (Bonnes pratiques de codage, commentaires,...)

# Langages de programmation

Le langage de programmation est un outil à l'aide duquel le programmeur écrit des programmes exécutables sur un ordinateur.

Il y a toute une panoplie de langages disponibles.

**Exemples** : FORTRAN, COBOL, Pascal, Ada, C, Java, Python

Il est important de pouvoir évaluer ces langages afin de pouvoir les choisir de manière appropriée et de les améliorer

Trois critères d'évaluation sont généralement utilisés :

1. **la lisibilité,**
2. **la facilité d'écriture, et**
3. **la fiabilité**

# Besoin d'évaluation

**La lisibilité** correspond a la facilité avec laquelle un programme peut-être lu et compris

**La facilité** d'écriture correspond a la facilité avec laquelle un langage peut être utilisé pour créer un programme

**La fiabilité** correspond au degré de confiance avec lequel un programme peut être exécuté sous différentes conditions et aboutir aux mêmes résultats

Un autre critère important est le coût de ce langage

# Caractéristiques des langages affectant les critères d'évaluation

<u>Caractéristique</u>	<u>Critères</u>		
	<u>Lisibilité</u>	<u>Écriture</u>	<u>Fiabilité</u>
Simplicité/Orthogonalité	●	●	●
Structure de Contrôle	●	●	●
Type et Structure des Données	●	●	●
Conception Syntaxique	●	●	●
Soutien pour l'Abstraction		●	●
Expressivité		●	●
Vérification de Type			●
Traitement d'Exceptions			●

# Critères affectant la lisibilité

## La simplicité

- S'il y a beaucoup de composantes de bases (tels que les mots clés), il est difficile toutes les connaître
- S'il existe plusieurs façons d'exprimer une commande, il est aussi difficile de toutes les connaître
- Si les opérateurs peuvent être surchargés (c'est-à-dire écrits de différentes manières), cela constitue une autre source de difficulté pour la lecture

Néanmoins, trop de simplicité cause une difficulté dans la lecture. Il faut alors trouver un compromis entre la simplicité et la facilité d'écriture

# Critères affectant la lisibilité

## L'orthogonalité

L'orthogonalité est la propriété qui signifie **"Changer A ne change pas B"**.

Dans les langages de programmation, cela signifie que lorsque vous exécutez une instruction, rien que cette instruction ne se produit

De plus, la signification d'un élément du langage doit être indépendante du contexte dans lequel il apparaît

Si le degré d'orthogonalité est bas, il y aura beaucoup d'exceptions à ses règles, et cela causera beaucoup de difficultés pour la lecture, la compréhension et la production de code fiable

# Critères affectant la lisibilité

## Instructions de contrôle

Pour la lisibilité d'un langage de programmation, il est important d'avoir des structures de contrôle adéquates ( structures itératives, structures conditionnelles, etc)

Par exemple, l'un des plus grands problèmes du premier ,BASIC est que sa seule instruction de contrôle était le « **goto** »

Cela créait des situations où le lecteur était renvoyé à différents points du programme

# Critères affectant la lisibilité

## Types et structures de donnée

La présence de moyens appropriés pour définir des types et des structures de données dans un langage peut améliorer considérablement la lisibilité

Exemple : l'utilisation des type "Booleen" ou les enregistrements à types variés ("records") peut clarifier certaines instructions



# Critères affectant la lisibilité

## La syntaxe

Voici des exemples où la syntaxe peut influencer la lisibilité :

- La forme des identificateurs (noms de variables) : des identificateurs trop courts peuvent rendre la lecture difficile
- Les mots clés: ces mots peuvent aider à la compréhension (par exemple "begin", "end", "if", "and", "or", etc.)
- si ces les mots clés peuvent être utilisés comme identificateurs, cela rendra la lecture plus difficiles

# Critères affectant la facilité d'écriture

## La simplicité et l'orthogonalité

Si un langage a une grande variation de constructeurs syntaxiques, il est fort possible que certains programmeurs ne les connaissent pas

De même, il se peut que le programmeur ne connaisse certains constructeurs que superficiellement et les utilise de manière erronée

# Critères affectant la facilité d'écriture

## L'abstraction

L'abstraction est la possibilité de définir des structures ou des opérations compliquées tout en cachant leurs détails (abstraction de processus et abstraction des données)

L'abstraction est très importante dans l'écriture d'un programme car elle peut rendre l'écriture beaucoup plus aisée

### 1. *Abstraction de processus* :

Quand un processus est abstrait dans un sous-programme il n'est pas nécessaire de répéter son code à chaque fois qu'il est utilisé . Un simple appel de la procédure/fonction est suffisant

### 2. *Abstraction des données*:

les données peuvent être abstraites par les langages de programmation de haut niveau dans des objets à interface simple. L'utilisateur n'a pas besoin de connaître les détails d'implémentation pour les utiliser (utilisation des arbres, tables de hachage,...)

# Critères affectant la facilité d'écriture

## L'expressivité

Un langage est expressif s'il offre des outils simples, commodes et intuitifs pour permettre au programmeur d'exprimer les différents concepts de programmation

Exemple : Utiliser des boucles "for" et "while" au lieu de "goto"

# Critères affectant la fiabilité

## Vérification de types

La vérification de type signifie qu'un langage est capable de détecter les erreurs relatives aux types de données

- Soit lors de la compilation
- Soit lors de l'exécution

Exemple :

Si le langage de programmation C ne détecte pas ces erreurs, le programme peut-être exécuté, mais les résultats ne seront pas significatifs

# Critères affectant la fiabilité

## Prise en Charge des Exceptions

La possibilité pour un programme d'intercepter les erreurs faites pendant l'exécution, de les corriger, et de continuer l'exécution augmente de beaucoup la fiabilité du langage de programmation

Exemple :

Des langages tels que Python, Ada, C++ et Java, Ruby, C# ont des capacités étendues de prise en charge des exceptions, mais de telles capacités sont absentes dans d'autres langages tels que le C ou le FORTRAN

# Critères affectant la fiabilité

## Lisibilité et facilité d'écriture

La lisibilité et la facilité d'écriture influencent la fiabilité des langages de programmation

En effet, si il n'y a pas de moyens naturels d'exprimer un algorithme, des solutions complexes seront utilisées (possible), et le risque d'erreurs (bugs) augmente

# Coût d'un langage de programmation

Ces coûts dépendent de plusieurs facteurs

1 - Si le langage n'est pas simple et orthogonal alors :

- les coûts de formation de programmeurs seront plus élevés
- l'écriture de programmes coûtera plus cher

2 - Autres facteurs

- Les coûts de la compilation et de l'exécution de programmes
- Les coûts de la maintenance de programmes
- le coût de la mise en marche du langage
- le coût lié au manque de fiabilité
- le coût de la maintenance du langage (correction, modification et ajout de nouvelles fonctionnalités)

3 - Néanmoins, il y a aussi d'autres critères tels que :

- La portabilité
- La généralité/spécificité
- La précision et la complétude de la description
- La vitesse d'exécution



# Langage python

Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une

approche modulaire et orientée objet de la programmation.

Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles

En février 1991, la première version publique, numérotée 0.9.0

Afin de réparer certains défauts du langage, la version Python 3.0 a été publié en décembre 2008.

Cette version a été suivie par une version 3.1 qui corrige les erreurs de la version 3.0



# Caractéristiques de Python

## Portable

Python est portable, non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires: MacOS, BeOS, NeXTStep, MS-DOS et les différentes variantes de Windows

## Gratuit

Python est gratuit, mais on peut l'utiliser sans restriction dans des projets commerciaux

## Simple

Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des projets complexes de plusieurs dizaines de milliers de lignes.

La syntaxe de Python est très simple et, combinée à des types de données évolués (listes, dictionnaires,...), conduit à des programmes à la fois très compacts et très lisibles.

Python gère ses ressources (mémoire, descripteurs de fichiers...) sans intervention du programmeur

## Orienté Objet

Python est orienté-objet. Il supporte l'héritage multiple et la surcharge des opérateurs.

Python intègre un système d'exceptions, qui permettent de simplifier considérablement la gestion des erreurs.

# Caractéristiques de Python

## Dynamique

Python est dynamique (l'interpréteur peut évaluer des chaînes de caractères représentant des expressions ou des instructions Python), orthogonal (un petit nombre de concepts suffit à engendrer des constructions très riches) et introspectif (un grand nombre d'outils de développement, comme le debugger sont implantés en Python lui-même).

Python est dynamiquement typé c'est à dire tout objet manipulable par le programmeur possède un type bien défini à l'exécution, qui n'a pas besoin d'être déclaré à l'avance.

Python possède actuellement deux implémentations

- Une interprétée, dans laquelle les programmes Python sont compilés en instructions portables, puis exécutés par une machine virtuelle
- L'autre génère directement du bytecode Java.

## Extensible

Python est extensible, on peut facilement l'interfacer avec des bibliothèques C existantes.

## Bibliothèque

La bibliothèque standard de Python, et les paquetages contribués, donnent accès à une grande variété de services: chaînes de caractères et expressions régulières, services UNIX standard (fichiers, pipes, signaux, sockets, threads...), protocoles Internet (Web, News, FTP, CGI, HTML...), persistance et bases de données, interfaces graphiques



# CHAPITRE 1

## TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

1- Critères de Choix d'un langage de programmation

### 2- Blocs d'instructions

3-Conversion de l'algorithme en Python

4- Optimisation du code (Bonnes pratiques de codage, commentaires,...)

# Structuration et notion de bloc

En Python, chaque instruction s'écrit sur une ligne sans mettre d'espace.

**Exemple :**

```
a = 10  
b = 3  
print(a, b)
```

Ces instructions simples peuvent cependant être mises sur la même ligne en les séparant par des points virgules ;, les lignes étant exécutées dans l'ordre de gauche à droite:

**Exemple :**

```
a = 10; b = 3; print(a, b)
```

# Structuration et notion de bloc

La séparation entre les en-têtes qui sont des lignes de définition de boucles, de fonction, de classe qui se terminent par les deux points : et le contenu ou 'bloc' d'instructions correspondant se fait par indentation des lignes

Une indentation s'obtient par le bouton tab (pour tabulation) ou bien par 4 espaces successifs. Ainsi, l'ensemble des lignes indentées constitue un bloc d'instructions.

Un programme python se structurera donc typiquement sous la forme ci-dessous :

```

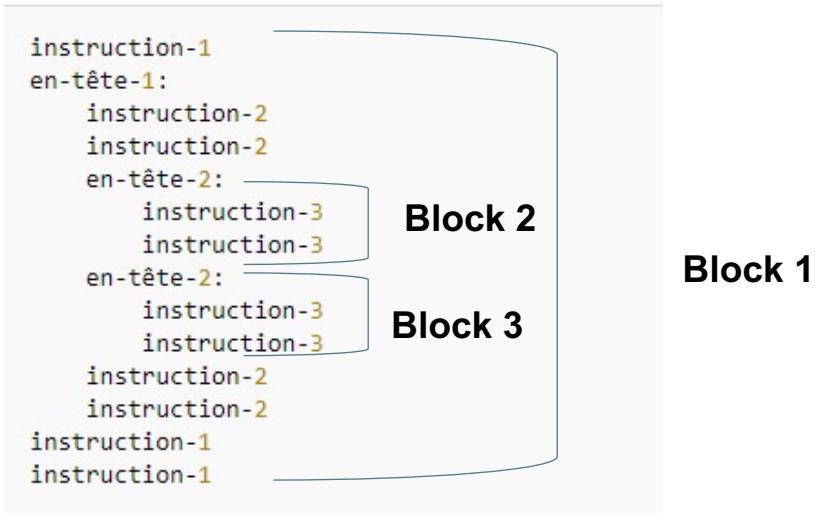
instruction-1
en-tête-1:
    instruction-2
    instruction-2
    en-tête-2:
        instruction-3
        instruction-3
    en-tête-2:
        instruction-3
        instruction-3
    instruction-2
    instruction-2
instruction-1
instruction-1

```

**Block 2**

**Block 3**

**Block 1**





# CHAPITRE 1

## TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

- 1- Critères de Choix d'un langage de programmation
- 2- Blocs d'instructions
- 3-Conversion de l'algorithme en Python**
- 4- Optimisation du code (Bonnes pratiques de codage, commentaires,...)

# Script et langage python

- Peu de ponctuation
- Pas de point virgule ";"
- Tabulations/4 espaces significatifs
- Scripts avec exécution d'un fichier (extension .py) ex: script.py

Affichage d'une invite (*prompt*)  
`>>>` 5 + 3 `read` : l'utilisateur tape une expression  
8 `eval` et `print` : calcul et affichage du résultat  
`>>>` Réaffichage d'une invite

Python utilise un identifiant pour nommer chaque objet.

Python n'offre pas la notion de variable, mais plutôt celle de référence (adresse) d'objet.

```
>>> b = 2
>>> c = b
>>> c
2
>>> b = 3
>>> c
2
```



# Types de données

Les types de données les plus utilisés sont :

- Type entier(integer)
- `>>> a= 1, 5555, 1102`
- Type réel(float)
- `>>> b= .003, 2., 1`
- Type boolean
- `>>> a = True`

## Types de caractère

```
>>> phrase1 = 'les oeufs durs.'  
>>> phrase2 = '"Oui", répondit-il,'  
>>> phrase3 = "j'aime bien"  
>>> print(phrase2, phrase3, phrase1)  
  
"Oui", répondit-il, j'aime bien les oeufs durs.
```

# Variables

Une variable est créée au moment où vous lui attribuez une valeur pour la première fois.

```
>>>x = 5  
>>>y = "John"  
>>>print(x)  
>>>print(y)
```

Les variables de chaîne peuvent être déclarées à l'aide de guillemets simples ou doubles:

```
>>>x = "John" # ceci est identique à faire x = 'John'
```

# Variables

## Règles pour les variables Python

Un nom de variable doit commencer par une lettre ou le caractère de soulignement

Un nom de variable ne peut pas commencer par un nombre

Un nom de variable ne peut contenir que des caractères alphanumériques et des traits de soulignement (A-z, 0-9 et \_)

Les noms de variable sont sensibles à la casse (age, Age et AGE sont trois variables différentes)

Python permet d'affecter des valeurs à plusieurs variables sur une seule ligne

```
>>>x, y, z = "Orange", "Banana", "Cherry"  
>>>print(x)  
>>>print(y)  
>>>print(z)
```

# Variables de sortie

La fonction print Python est souvent utilisée pour afficher des variables et des chaînes de caractères.

Pour combiner à la fois du texte et une variable, Python utilise le caractère +:

```
>>>x = "cool"  
>>>print("Python est " + x)  
  
>>>x = "Python est "  
>>>y = "cool"  
>>>z = x + y  
>>>print(z)
```

# Variables de sortie

La fonction print Python est souvent utilisée pour afficher des variables et des chaînes de caractères.

```
>>> print("Hello") ; print("Joe")
```

```
Hello
```

```
Joe
```

```
>>> print("Hello", end="") ; print("Joe")
```

```
HelloJoe
```

```
>>> print("Hello", end=" ") ; print("Joe")
```

```
Hello Joe
```

Le mot clé **end** évite le retour à la ligne

# Variables de sortie

La fonction print Python est souvent utilisée pour afficher des variables et des chaînes de caractères.

```
>>> x = 32
>>> nom = "John"
>>> print(nom, "a", x, "ans")
John a 32 ans

>>> x = 32
>>> nom = "John"
>>> print(nom, "a", x, "ans", sep="")
Johna32ans

>>> print(nom, "a", x, "ans", sep="-")
John-a-32-ans
```

Le mot clé **sep** précise une séparation entre les variables chaînes

# Affichage formaté

La méthode .format() permet une meilleure organisation de l'affichage des variables

```
>>> x = 32
>>> nom = "John"
>>> print("{} a {} ans".format(nom, x))
John a 32 ans

>>> x = 32
>>> nom = "John"
>>> print("{0} a {1} ans".format(nom, x))
John a 32 ans
```

# Manipulation des Types numériques

Les quatre opérations arithmétiques de base se font de manière simple sur les types numériques (nombres entiers et floats)

>>> x = 45	>>> (x * 10) + y	>>> i = 0
>>> x + 2	452.5	>>> i = i + 1
47	>>> 3 / 4	>>> i
>>> x - 2	0.75	1
43	>>> 2**3	>>> i += 1
>>> x * 3	8	>>> i
135	>>> 5 // 4	2
>>> y = 2.5	1	>>> i += 2
>>> x - y	>>> 5 % 4	>>> i
42.5	1	4



# Manipulation des chaînes de caractères

En Python une chaîne de caractères est un objet de la classe str

Les opérateurs de concaténation (+) et de répétition (\*)

```
1. >>> s1 = "Welcome"
2. >>> s2 = "Python"
3. >>> s3 = s1 + " to " + s2
4. >>> s3
5. 'Welcome to Python'
6. >>> s4 = 3 * s1
7. >>> s4
8. 'WelcomeWelcomeWelcome'
9. >>> s5 = s1 * 3
10. >>> s5
11. 'WelcomeWelcomeWelcome'
12. >>>
```

## Les opérateurs in et not in

```
1. >>> s1 = "Welcome"
2. >>> "come" in s1
3. True
4. >>> "come" not in s1
5. False
6. >>>
```

```
1. s = input("Entrer une chaîne de caractères: ")
2.
3. if "Python" in s:
4.     print("Python est dans ", s)
5. else:
6.     print("Python n'est pas dans ", s)
```

# Manipulation des chaînes de caractères

En Python une chaîne de caractères est un objet de la classe str.

Création de chaînes des caractères

Créer des chaînes de caractères en utilisant le mot clé str comme suit :

```
1. s1 = str() # Créer un objet chaîne de caractères vide
2. s2 = str("Welcome") # Créer l' objet chaîne de caractères "Welcome"
```

```
1. s1 = "" # identique à s1 = str()
2. s2 = "Welcome" # identique à s2 = str("Welcome")
```

# Manipulation des chaînes de caractères

En Python une chaîne de caractères est un objet de la classe str

Fonctions des chaînes de caractères

Plusieurs fonctions intégrées en Python sont utilisées avec les chaînes de caractères.

```
1. >>> s = "Welcome"
2. >>> len(s)
3. 7
4. >>> max(s)
5. 'o'
6. >>> min(s)
7. 'W'
8. >>>
```

Puisque s a 7 caractères, len(s) renvoie 7 (ligne 3).

Notez que les lettres minuscules ont une valeur ASCII supérieure à celle des lettres majuscules, donc max(s) retourne 'o' (ligne 5) et min(s) retourne 'W' (ligne 7).

# Manipulation des chaînes de caractères

## Fonctions des chaînes de caractères

```
1. s = input("Entrer une chaîne de caractères: ")
2.
3. if len(s) % 2 == 0:
4.     print(s, " contient un nombre pair de caractères")
5. else:
6.     print(s, " contient un nombre impair de caractères")
```

## L'opérateur indice [ ]

Une chaîne de caractères est une séquence de caractères.

Un caractère de la chaîne est accessible par l'opérateur indice [ ]

```
1. >>> s = "Welcome"
2. >>> for i in range(0, len(s), 2):
3.     ... print(s[i], end = ' ')
4. Wloe
5. >>>
```

# Manipulation des chaînes de caractères

## Découpage en tranche ([début :fin])

```
1. >>> s = "Welcome"
2. >>> s[1 : 4]
3. 'elc'
```

```
1. >>> s = "Welcome"
2. >>> s[ : 6]
3. 'Welcom'
4. >>> s[4 : ]
5. 'ome'
6. >>> s[1 : -1]
7. 'elcom'
8. >>>
```

## Comparaison de chaînes de caractères

```
1. s1 = input("Entrer la 1ère chaîne de caractères: ")
2. s2 = input("Entrer la 2ème chaîne de caractères: ")
3. if s2 < s1:
4.     s1, s2 = s2, s1
5.
6. print("Les deux chaînes sont dans cet ordre:", s1, s2)
```

```
10. False
11. >>> "green" <= "glow"
12. False
13. >>> "ab" <= "abc"
14. True
```

# Manipulation des chaînes de caractères

## Recherche de sous-chaînes

```
1. >>> s = "welcome to python"
2. >>> s.endswith("thon")
3. True
4. >>> s.startswith("good")
5. False
6. >>> s.find("come")
7. 3
8. >>> s.find("become")
9. -1

>>> s.count("o")
3
>>> s.rfind("o")
15
```

# Structure conditionnelle

## Exemple 1

```
>>>a = 33
>>>b = 200
>>>if b > a:
>>>    print("b is greater than a")
```

## Exemple 2

```
>>>a = 33
>>>b = 33
>>>if b > a:
>>>    print("b is greater than a")
>>>elif a == b:
>>>    print("a and b are equal")
```

## Exemple3

```
>>>a = 200
>>>b = 33
>>>if b > a:
>>>    print("b is greater than a")
>>>elif a == b:
>>>    print("a and b are equal")
>>>else:
>>>    print("a is greater than b")
```

# Structure conditionnelle

Si vous n'avez qu'une seule instruction à exécuter, vous pouvez la mettre sur la même ligne que l'instruction if

Le mot clé or est un opérateur logique et est utilisé pour combiner des instructions conditionnelles

```
>>> if a > b: print("a is greater than b")
```

Le mot clé and est un opérateur logique et est utilisé pour combiner des instructions conditionnelles:

```
>>> a = 200
>>> b = 33
>>> c = 500
>>> if a > b or a > c:
>>>     print("At least one of the conditions is True")
```

```
>>> a = 200
>>> b = 33
>>> c = 500
>>> if a > b and c > a:
>>>     print("Both conditions are True")
```



# Les boucles d'itérations

Python a deux commandes de boucle primitives :

- Boucle while
- Boucle for

Avec la boucle while, nous pouvons exécuter un ensemble d'instructions tant qu'une condition est vraie

```
>>>i = 1  
>>>while i < 6:  
>>>    print(i)  
>>>    i += 1
```

# Les boucles d'itérations

Avec l'instruction `break`, nous pouvons arrêter la boucle même si la condition `while` est vraie:

```
>>>i = 1
>>>while i < 6:
>>>    print(i)
>>>    if i == 3:
>>>        break
>>>    i += 1
```

Avec l'instruction `continue`, nous pouvons arrêter l'itération en cours et continuer avec la suivante:

```
>>>i = 0
>>>while i < 6:
>>>    i += 1
>>>    if i == 3:
>>>        continue
>>>    print(i)
```

# Les boucles d'itérations

Une boucle for est utilisée pour itérer sur une séquence (c'est-à-dire une liste, un tuple, un dictionnaire, un ensemble ou une chaîne):

```
>>>fruits = ["apple", "banana", "cherry"]
>>>for x in fruits:
>>>     print(x)
```

Même les chaînes sont des objets itérables, elles contiennent une séquence de caractères:

Avec l'instruction break, nous pouvons arrêter la boucle avant d'avoir bouclé tous les éléments:

```
>>>for x in "banana":
>>>     print(x)
```

```
>>>fruits = ["apple", "banana", "cherry"]
>>>for x in fruits:
>>>     print(x)
>>>     if x == "banana":
>>>         break
```

# Les boucles d'itérations

Pour parcourir un ensemble de codes un nombre spécifié de fois, nous pouvons utiliser la fonction range ().

La fonction range () renvoie une séquence de nombres, commençant à 0 par défaut, et incrémentant de 1 (par défaut), et se termine à un nombre spécifié.

```
>>>for x in range(6):  
>>>    print(x)
```

La fonction range () par défaut est 0 comme valeur de départ, mais il est possible de spécifier la valeur de départ en ajoutant un paramètre: range (2, 6), ce qui signifie des valeurs de 2 à 6 (mais pas 6).

```
>>>for x in range(2, 6):  
>>>    print(x)
```

La fonction range () par défaut incrémente la séquence de 1, mais il est possible de spécifier la valeur d'incrément en ajoutant un troisième paramètre: range (2, 30, 3)

```
>>>for x in range(2, 30, 3):  
>>>    print(x)
```



# CHAPITRE 1

## TRANSFORMER UNE SUITE D'ÉTAPES ALGORITHMIQUE EN UNE SUITE D'INSTRUCTIONS PYTHON

1- Critères de Choix d'un langage de programmation

2- Blocs d'instructions

3-Conversion de l'algorithme en Python

**4- Optimisation du code (Bonnes pratiques de codage,  
commentaires,...)**

# Python Enhancement Proposal (PEP)

## Introduction

Afin d'améliorer le langage Python, la communauté qui développe Python publie régulièrement des Python Enhancement Proposal (PEP), suivi d'un numéro.

Il s'agit de propositions concrètes pour améliorer le code, ajouter de nouvelles fonctionnalités, mais aussi des recommandations sur la manière d'utiliser Python, bien écrire du code, etc.

On parle de code pythonique lorsque ce dernier respecte les règles d'écriture définies par la communauté Python mais aussi les règles d'usage du langage.

La PEP 8 Style Guide for Python Code 2 est une des plus anciennes PEP (les numéros sont croissants avec le temps). Elle consiste en un nombre important de recommandations sur la syntaxe de Python.

# Python Enhancement Proposal

## Indentation

L'indentation est obligatoire en Python pour séparer les blocs d'instructions.

Cela vient d'un constat simple, l'indentation améliore la lisibilité d'un code

Dans la PEP 8, la recommandation pour la syntaxe de chaque niveau d'indentation est très simple : 4 espaces

## Importation des modules

Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi bibliothèques ou libraries). Ce sont des « boîtes à outils » qui vont vous être très utiles.

L'utilisation de la syntaxe `import module` permet d'importer tout une série de fonctions organisées par «thèmes ». Par exemple, les fonctions gérant les nombres aléatoires avec `random` et les fonctions mathématiques avec `math`. Python possède de nombreux autres modules internes (c'est-à-dire présent de base lorsqu'on installe Python)

Exemple

```
>>> import math
>>> math.cos(math.pi / 2)
6.123233995736766e-17
>>> math.sin(math.pi / 2)
1.0
```

# Python Enhancement Proposal

## Importation des modules

Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi bibliothèques ou libraries). Ce sont des « boîtes à outils » qui vont vous être très utiles.

L'utilisation de la syntaxe `import module` permet d'importer tout une série de fonctions organisées par «thèmes ». Par exemple, les fonctions gérant les nombres aléatoires avec `random` et les fonctions mathématiques avec `math`. Python possède de nombreux autres modules internes (c'est-à-dire présent de base lorsqu'on installe Python)

### Exemple

```
>>> import math
>>> math.cos(math.pi / 2)
6.123233995736766e-17
>>> math.sin(math.pi / 2)
1.0
```

Le chargement d'un module se fait avec l'instruction `import module` plutôt qu'avec `from module import *`

Si on souhaite ensuite utiliser une fonction d'un module, la première syntaxe conduit à `module.fonction()` ce qui rend explicite la provenance de la fonction. Avec la seconde syntaxe, il faudrait écrire `fonction()` ce qui peut :

- mener à un conflit si une de vos fonctions a le même nom;
- rendre difficile la recherche de documentation si on ne sait pas d'où vient la fonction, notamment si plusieurs modules sont chargés avec l'instruction `from module import *`
- Dans un script Python, on importe en général un module par ligne. D'abord les modules internes (classés par ordre alphabétique), c'est-à-dire les modules de base de Python, puis les modules externes (ceux que vous avez installés en plus)



# Python Enhancement Proposal

## Règles de nommage

Les noms de variables, de fonctions et de modules doivent être en minuscules avec un caractère « souligné » (« tiret du bas » ou underscore en anglais) pour séparer les différents « mots » dans le nom.

```
ma_variable  
fonction_test_27()  
mon_module
```

Les constantes sont écrites en majuscules

```
MA_CONSTANTE  
VITESSE_LUMIERE
```

# Python Enhancement Proposal

## Gestion des espaces

La PEP 8 recommande d'entourer les opérateurs (+, -, /, \*, ==, !=, >=, not, in, and, or. . . ) d'un espace avant et d'un espace après. Par exemple :

```
# code recommandé :  
ma_variable = 3 + 7  
mon_texte = "souris"  
mon_texte == ma_variable  
# code non recommandé :  
ma_variable=3+7  
mon_texte="souris"  
mon_texte== ma_variable
```

Il n'y a, par contre, pas d'espace à l'intérieur de crochets, d'accolades et de parenthèses :

```
# code recommandé :  
ma_liste[1]  
mon_dico{"clé"}  
ma_fonction(argument)  
# code non recommandé :  
ma_liste[ 1 ]  
mon_dico{"clé" }  
ma_fonction( argument )
```

# Python Enhancement Proposal

Ni juste avant la parenthèse ouvrante d'une fonction ou le crochet ouvrant d'une liste ou d'un dictionnaire

```
# code recommandé :  
ma_liste[1]  
mon_dico{"clé"}  
ma_fonction(argument)  
# code non recommandé :  
ma_liste [1]  
mon_dico {"clé"}  
ma_fonction (argument)
```

On met un espace après les caractères **: et**, (mais pas avant)

```
# code recommandé :  
ma_liste = [1, 2, 3]  
mon_dico = {"clé1": "valeur1", "clé2": "valeur2"}  
ma_fonction(argument1, argument2)  
# code non recommandé :  
ma_liste = [1 , 2 ,3]  
mon_dico = {"clé1": "valeur1", "clé2": "valeur2"}  
ma_fonction(argument1 ,argument2)
```

# Python Enhancement Proposal

Par contre, pour les tranches de listes, on ne met pas d'espace autour du `:`

```
ma_liste = [1, 3, 5, 7, 9, 1]
# code recommandé :
ma_liste[1:3]
ma_liste[1:4:2]
ma_liste[::2]
# code non recommandé :
ma_liste[1 : 3]
ma_liste[1: 4:2 ]
ma_liste[ : :2]
```

On n'ajoute pas plusieurs espaces autour du `=` ou des autres opérateurs

```
# code recommandé :
x1 = 1
x2 = 3
x_old = 5
```

```
# code non recommandé :
x1      = 1
x2      = 3
x_old   = 5
```

# Python Enhancement Proposal

Longueur de ligne

Une ligne de code ne doit pas dépasser 79 caractères

Le caractère \ permet de couper des lignes trop longues.

```
>>> ma_variable = 3
>>> if ma_variable > 1 and ma_variable < 10 \
... and ma_variable % 2 == 1 and ma_variable % 3 == 0:
...     print(f"ma variable vaut {ma_variable}")
...
ma variable vaut 3
```

À l'intérieur d'une parenthèse, on peut revenir à la ligne sans utiliser le caractère \. C'est particulièrement utile pour préciser les arguments d'une fonction ou d'une méthode, lors de sa création ou lors de son utilisation :

```
>>> def ma_fonction(argument_1, argument_2,
...                 argument_3, argument_4):
...     return argument_1 + argument_2
...
>>> ma_fonction("texte très long", "tigre",
...             "singe", "souris")
'texte très longtigresouris'
```

# Python Enhancement Proposal

Les parenthèses sont également très pratiques pour répartir sur plusieurs lignes une chaîne de caractères qui sera affichée sur une seule ligne :

```
>>> print(" ATGCGTACAGTATCGATAAC"  
...       "ATGACTGCTACGATCGGATA"  
...       "CGGGTAACGCCATGTACATT")  
ATGCGTACAGTATCGATAACATGACTGCTACGATCGGATACGGGTAACGCCATGTACATT
```

L'opérateur + est utilisée pour concaténer les trois chaînes de caractères et que celles-ci ne sont pas séparées par des virgules. À partir du moment où elles sont entre parenthèses, Python les concatène automatiquement. On peut aussi utiliser les parenthèses pour évaluer un expression trop longue :

```
>>> ma_variable = 3  
>>> if (ma_variable > 1 and ma_variable < 10  
... and ma_variable % 2 == 1 and ma_variable % 3 == 0):  
...     print(f"ma variable vaut {ma_variable}")  
...  
ma variable vaut 3
```

# Python Enhancement Proposal



## Lignes vides

Dans un script, les lignes vides sont utiles pour séparer visuellement les différentes parties du code. Il est recommandé de laisser deux lignes vides avant la définition d'une fonction.

On peut aussi laisser une ligne vide dans le corps d'une fonction pour séparer les sections logiques de la fonction, mais cela est à utiliser avec parcimonie.

## Commentaires

Les commentaires débutent toujours par le symbole # suivi d'un espace. Les commentaires donnent des explications claires sur l'utilité du code et doivent être synchronisés avec le code, c'est-à-dire que si le code est modifié, les commentaires doivent l'être aussi (le cas échéant).

Les commentaires sont sur le même niveau d'indentation que le code qu'ils commentent. Ils sont constitués de phrases complètes, avec une majuscule au début (sauf si le premier mot est une variable qui s'écrit sans majuscule) et un point à la fin.

PEP 8 recommande la cohérence entre la langue utilisée pour les commentaires et la langue utilisée pour nommer les variables. Pour un programme scientifique, les commentaires et les noms de variables sont en anglais.