

---

# Carnet de Travaux pratiques PYTHON et quelques éléments de didactique de l'algorithmique et de performances des algorithmes

Efforce de toi de ceci pour valider

---

**Dr DIALLO Mohamed Bobo**  
CAMES Côte d'Ivoire  
Campus de Cocody

# Didactique de l'algorithmique

Enseigner l'algorithmique n'est une tâche prioritaire qu'apprendre à programmer qu'a instruction set de sériabilité. En effet, tout processeur a un instruction set qui vous permet d'ajouter deux mémoires (*registres*), de les multiplier, and so on.

Considérons le jeu d'instruction d'un processeur INTEL 8086 :

|                                       |                    |                                        |                                                                                    |
|---------------------------------------|--------------------|----------------------------------------|------------------------------------------------------------------------------------|
| Instructions de transferts de données | Instruction        | Fonction                               | Utilité                                                                            |
|                                       | MOV dest, src      | Copie une donnée                       | Instruction la plus utilisée : transferts registre→mémoire, chargements, stockage. |
|                                       | PUSH src           | Place une valeur sur la pile           | Passage d'arguments, sauvegarde de contexte.                                       |
|                                       | POP dest           | Récupère la dernière valeur de la pile | Restauration d'état, fin d'appel de fonction.                                      |
| Instructions arithmétiques            | Instruction        | Fonction                               | Utilité                                                                            |
|                                       | ADD dest, src      | Addition                               | Opérations générales, incréments.                                                  |
|                                       | SUB dest, src      | Soustraction                           | Décréments, opérations diverses.                                                   |
|                                       | INC / DEC          | +1 / -1                                | Boucles, compteurs.                                                                |
|                                       | MUL src            | Multiplication non signée              | Calculs rapides en arithmétique.                                                   |
|                                       | IMUL src           | Multiplication signée                  | Utilisé notamment dans calculs algébriques.                                        |
|                                       | DIV / IDIV         | Division non signée / signée           | Opérations arithmétiques complexes.                                                |
| Instructions logiques                 | Instruction        | Fonction                               | Utilité                                                                            |
|                                       | AND dest, src      | ET logique                             | Masquage de bits, activation/désactivation de flags.                               |
|                                       | OR dest, src       | OU logique                             | Assemblage de bits, réglage de valeurs.                                            |
|                                       | XOR dest, src      | OU exclusif                            | Remise à zéro rapide : XOR EAX, EAX .                                              |
|                                       | NOT dest           | Inversion des bits                     | Opérations bit-à-bit.                                                              |
|                                       | SHL / SHR          | Décalage gauche / droite               | Multiplication/division par puissances de 2.                                       |
|                                       | ROL / ROR          | Rotation                               | Algorithmes cryptographiques et CRC.                                               |
| Instruction de contrôle de flux       | Instruction        | Fonction                               | Utilité                                                                            |
|                                       | JMP label          | Saut inconditionnel                    | Branchements directs.                                                              |
|                                       | JZ / JNZ           | Saut si ZF = 1 / 0                     | Conditions "if", comparaisons.                                                     |
|                                       | JG / JL / JE / JNE | Sauts selon résultat signé             | Conditions élaborées.                                                              |
|                                       | CALL func          | Appel de fonction                      | Gestion des sous-programmes.                                                       |
|                                       | RET                | Retour de fonction                     | Fin d'appel.                                                                       |
|                                       | LOOP label         | Décrément de RCX/ECX et boucle         | Boucles simples sans test explicite.                                               |

Par exemple pour additionner deux registres :

```
asm

mov al, 15          ; 0x0F
mov bl, 27          ; 0x1B
add al, bl          ; AL = 42
```

Figure 1: Vous avez compris ?

La programmation consiste alors fondamentalement à exploiter les possibilités d'une expressive algorithmation qui à chaque séquence d'instruction réalise une des fonctions bien précises nécessitant calcul, stockage et dictats de tout genre à un ordinateur pour parvenir à bon usage attendu.



Figure 2: Commander un processeur comme un drone asservi solution de haute technologie pilotable d'expressive digitalité miraculeusement intriquat responsif

Ainsi tout ce qu'un langage vous a facilité, familiariser vous y de petits bouts de code à propulser qu'à tri fusion sophistication digeste.

Par exemple, Python vous a reversé une zone mémoire nommée `raw` :

```
var [::-1]
```

Alors que C vous l'a solutionné que de son efficacité :

```
for (i = 0; i < n/2; ++i)
    swap(a[i], a[n-1-i]);
```

Le novice en programmation doit vite tester et introspecter progressivement le vocabulaire syntaxique du langage pour pouvoir tout seul solutionner des problèmes à séquençage instructionnel résolu algorithmiquement. Ensuite lui apparaîtra la nécessité d'écrire des programmes optimisés dont le temps d'exécution pourra être réduit de plusieurs heures à quelques minutes, voire de quelques jours à plusieurs heures.

## Premiers pas en Python

---

- Choisir un éditeur Python pour environnement de développement en classe informatique
  - Cas Python votre premier langage : Spider qui ressemble au Turbo C et so else.
  - Cas vous avez déjà programmé avec Idle : vous aurez peut-être Spider like pour d'autres langages voire anaconda pour plein d'autres projets data science
- Les types en Python

Python infère le type automatiquement, on dit qu'il fait une *inférence de type*. Si votre variable a *rav* comme identifiant, vous pouvez vérifier son type avec l'instruction :

```
type(rav)
```

Les types de base indispensables sont :

| <code>type(rav)</code> |                                                                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int</code>       | <b>Entier</b><br><br>Les opérations usuelles sont immédiates<br>sauf la puissance (**)                                                                                                                                       |
| <code>float</code>     | <b>Réel</b><br><br>Python code les réels en double précision<br>IEEE 754 avec tous les risques afférents<br>de ne pas sécuriser les tests d'égalité<br>avec toutes les complications financières<br>que cela pourrait avoir. |

|     |                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| str | <p><b>Chaîne de caractère</b></p> <p>Les chaînes de caractères sont des listes de caractères indexables comme toute liste, voire tableau statique inexistant dans le jargon Python de base.</p> <p>rav[0] : Premier caractère</p> <p>rav[1] : Second caractère</p> <p>rav[-1]: Dernier caractère</p> <p>length(rav) :<br/>nombre de caractères</p> <p>rav[length(rav)] :<br/>Dernier caractère</p> |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Pour les collections de valeur consulter l'aide Python dans Idle comme au terminal Python quelconque :

help (list) pour savoir tout des listes Python d'une dizaine de minutes d'attention et de même help (dict) pour les tableaux associatifs nommés **dictionnaires** capacitant de clés de type chaîne de caractère ou auto-conçu des formalismes objets dus à tout langage.

- Les structures de contrôle en Python

|                                                                 |                                                                                                        |
|-----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Une boucle for explicite à la syntaxe le nombre de fois à itéré | <pre>for i in range(len(liste)):     if liste[i] == chercher:         position = i         break</pre> |
|-----------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                    |                                                                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <p>Quand à la boucle <code>while</code>, le nombre d'itérations dépend de la condition et de la valeur des variables initialement, et ne peut être inférée immédiatement à priori des cas pis.</p> | <pre>while True:     connexion = attendre_client()     traiter_requête(connexion)</pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|

- La saisie du clavier et l'affichage du terminal

|                      |                                                                                                                                                                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>input()</code> | La fonction <code>input</code> retourne toujours une chaîne de caractère attention dans le cas de manipulation d'entiers de convertir la chaîne de caractère saisie avec <code>int()</code> pour les entiers, et <code>float()</code> pour les valeurs réels. |
| <code>print()</code> | La syntaxe de la fonction <code>print</code> a évolué avec les versions Python notamment en ce qui concerne le formatage, et radicalement depuis Python3 où elle d'usage comme une fonction.                                                                  |

## Progression autour des nombres premiers

---

Un nombre premier n'a pas de diviseurs autre que lui-même et l'unité. Dans cette section, nous allons didacter la recherche d'un algorithme pour vérifier un nombre premier aux subentropies d'un néophyte en assignment.

Tout néo programmant commence par un algorithme naïf, puis l'améliore progressivement en fonction du chrono disponible.

L'algorithme le plus simple, se traduisant simplement consiste à vérifier qu'un nombre premier n'a pas de diviseurs entre 2 et le nombre entier le précédant inclus. Bien entendu 2 et 3 sont des nombres premiers.

Algorithme naïf :

Soit un entier naturel  $N$  dont la primalité n'est pas à priori connu

La solution consiste à chercher un diviseur entre 2 et  $N - 1$

Dès qu'il existe, le nombre  $N$  n'est pas premier et il faut alors arrêter la recherche

Sinon aucun diviseur n'est trouvé au terme de cette recherche, alors le nombre est PREMIER

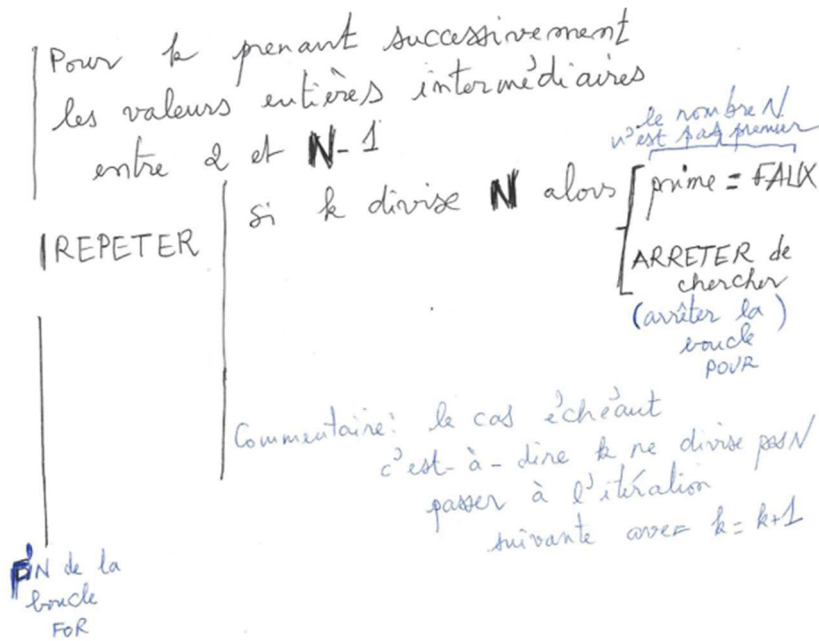
L'algorithme décrit ci-haut peut être documenté pour le programmeur novice souhaitant l'adopté aisément dans un langage de programmation quelconque comme suit

Soient

- $N$  un entier naturel (c'est-à-dire supérieur ou égal à 0) déjà initialisé à une valeur accessible au processeur
- Prime est une variable booléenne initialisée à VRAI (TRUE)

*Dans la recherche de diviseurs entre 2 et  $N-1$ , PRIME changera de valeurs à FAUX dès qu'un diviseur sera trouvé s'il est réductible sinon Prime restera à VRAI à l'issue de la recherche de diviseurs.*

- $K$  une variable entière initialisée à 0 utilisée pour itérer dans la boucle POUR nécessaire pour itérer le test de divisibilité dans la recherche de diviseurs.



La valeur de `PRIME` détermine à l'issue de la boucle le statut du nombre  $N$ .



Pour  $N=19$ , initialement les variables du programme auront les valeurs comme suit :

$$K = 2, \text{ PRIME} = \text{VRAI}$$

A la fin de la boucle `POUR` les variables ont la valeur ci-contre :

$$K = 18, \text{ PRIME} = \text{VRAI}$$

La boucle a 17 itérations et la valeur de `PRIME` ne change pas.

En généralisant pour un nombre premier  $N$  quelconque sauf cas triviaux premier (2) et second (3) à exceptionnat malgré validité de la généralité de la définition des nombres premiers,  $N-2$  itérations sont requises avant d'être sûr que le nombre est premier et la valeur de la variable `PRIME` ne change jamais.

Si le nombre  $N$  est pair, il n'y a qu'une seule itération, le programme se termine avec les variables dans l'état suivant :

$$K = 2, \text{ PRIME} = \text{FALSE}$$

Y compris quand  $N$  est un grand nombre tel  $2^{30}$ .

Un algorithme moins efficace aurait consisté à compter les diviseurs de  $N$ . Puis si le nombre a exactement 2 diviseurs alors le nombre est premier.

Dans ce cas pour un nombre premier, le nombre d'itérations de la boucle de recherche diviseurs et de détermination du nombre de diviseurs serait le même que pour un nombre quelconque non premier.

Dans ce dernier cas, cet algorithme n'est pas efficace et implémenté pour ce problème de nombre premier sur machine dans un langage de programmation résulterait dans un grand gaspillage pour les nombres réductibles vérifiables plus rapidement par la solution naïve.

Un petit exercice pour appliquer l'algorithme trouvé

---

### Exercice 101 : Ainsi Allahou et Mouhammadant dans le qourane

**Trouve le premier nombre premier dont l'ensemble de nombres premiers ainsi qu'il majore d'inclusive adhérence contienne tous les chiffres de la base 10**

# Comment rendre l'algorithme plus performant ?

Les mathématiques aidantes on sait :

1. Inutile de chercher des diviseurs parmi les nombres pairs en dehors de 2
2. Inutile de chercher au-delà de la partie entière de la racine carrée.
3. En dehors de 2 et 3, un nombre premier est de la forme  $6K - 1$  ou  $6K + 1$ . Par exemple  $19 = 6 * 3 + 1$ , et  $29 = 6 * 5 - 1$ .

## Exercice : Dénombrement du cardinal de nombres premiers dans un intervalle de grand nombre

**Ecris un programme pour afficher et déterminer le cardinal de nombres premiers dans un intervalle de grand nombre  $[10^9; 10^9 + 10000]$ . Quel est la valeur du cardinal et combien de temps est nécessaire pour l'exécution de l'algorithme de recherche ?**

```
1  import time
2  begin=time.time()
3  ctr = 0 # a la fin du programme contient le nombre de nombre premiers trouvé dans l'intervalle
4
5  for n in range(10**9+1, 10**9+10001, 2):
6
7      yetprime = True # initialisation de la variable yetprime
8      for i in range(2, int(n**0.5) + 1): #Recherche de diviseurs
9          if n % i==0:
10             yetprime = False # Diviseur trouvé
11             break # sortie de la boucle la plus imbriquée - passe au n suivant
12
13         # A ce stade la recherche de diviseurs est terminée
14         if yetprime == True : # Test de comparaison
15             ctr += 1 #incrémentatation : ctr = ctr + 1
16             #print(n) # affiche le nombre premier trouvé sur la console
17
18 #fin des deux boucles
19 elapsed = time.time() - begin
20 print(f"Temps : {elapsed}")
21 print(f"Cardinal des nombres premiers trouvés", ctr)
```

Figure 1: Un programme pour afficher et déterminer le cardinal de nombres premiers dans un intervalle de grand nombre  $[10^9; 10^9 + 10000]$ .

**1. De combien se réduit le temps de recherche si on ne vérifie la réductibilité que par des diviseurs impairs ?**

```

1  ## Recherche de diviseurs que parmi les nombres impairs
2  begin=time.time()
3
4  ctr = 0 # a la fin du programme contient le nombre de nombre premiers trouvé dans l'intervalle
5
6  for n in range(10**9+1, 10**9+10001, 2): #Séquence de nombres impairs
7
8      yetprime = True # initialisation de la variable yetprime
9
10     for i in range(3, int(n**0.5) + 1, 2): #Recherche de diviseurs parmi les impairs
11         if n % i == 0:
12             yetprime = False # Diviseur trouvé
13             break # sortie de la boucle la plus imbriquée - passe au n suivant
14
15     # A ce stade la recherche de diviseurs est terminée
16     if yetprime == True : # Test de comparaison
17         ctr += 1 #incrémentatation : ctr = ctr + 1
18         # print(n) # affiche le nombre premier trouvé sur la console
19
20 #fin des deux boucles

```

**Figure 2: On réduit le temps de recherche si on ne vérifie la réductibilité que par des diviseurs impairs.**

## 2. Quel est le gain en ne considérant pas les nombres pas de la forme nécessaire $6k+1$ au delà de 5

```

1  begin=time.time()
2  ctr = 0 # a la fin du programme contient le nombre de nombre premiers trouvé dans l'intervalle
3
4  for n in range(10**9+1, 10**9+10001, 2):
5
6      yetprime = True # initialisation de la variable yetprime
7
8      if n % 6 == 1 or n % 6 == 5: #on peut espérer que le nombre soit premier
9          for i in range(3, int(n**0.5) + 1, 2): #Recherche de diviseurs
10             if n % i == 0:
11                 yetprime = False # Diviseur trouvé
12                 break # sortie de la boucle la plus imbriquée - passe au n suivant
13             else:
14                 yetprime = False
15
16     # A ce stade la recherche de diviseurs est terminée
17     if yetprime == True : # Test de comparaison
18         ctr += 1 #incrémentatation : ctr = ctr + 1
19         # print(n) # affiche le nombre premier trouvé sur la console
20

```

**Figure 3: En ne considérant pas les nombres pas de la forme nécessaire  $6k+1$  au delà de 5. Le nombre est encore réduit.**

## 3. Combien de temps met la bibliothèque Python sur votre ordinateur ?

```

1  import sympy
2
3  begin = time.time()
4
5  L = list(sympy.primerange(10**9+1, 10**9+10001))
6
7  elapsed = time.time() - begin
8
9  print(f"Temps : {elapsed}")
10
11 print(f"Cardinal des nombres premiers trouvés", len(L))
12

```

Figure 4: Quelle est la performance de la solution bibliothèque Python sympy ?

**4. Visualiser le gain de performance sous forme de graphique judicieusement choisi en utilisant pyplot.**

**5. Met à jour le graphique de gain des performances en précalculant la liste des diviseurs premiers pour l'intervalle de recherche ?**

```

1  import math as MT
2  begin=time.time()
3  precomputedprimes = list(sympy.primerange(3, int(MT.sqrt(10**9+10001))))
4  N=0
5  for n in range(10**9+1,10**9+10001,2):
6      yetprime = True
7      if n%6==1 or n%6==5:#il se peut que le nombre soit premier
8          #for i in range(3,int(n**0.5)+1,2):
9              for i in precomputedprimes:
10                 if n % i == 0:
11                     yetprime=False
12
13                 break
14             else :
15                 yetprime=False
16
17             if yetprime ==True :
18                 N=N+1
19                 # print(n)
20  temps=time.time()-begin

```

Figure 5: On peut précalculer la liste des diviseurs premiers pour l'intervalle de recherche. Et ne chercher des diviseurs pour cet intervalle que parmi eux

---

## Problème pour approfondir

**Etudiez la densité des nombres premiers par intervalles de 1000 entre 1 et 1 million. Puis par intervalle de 10. Enfin par intervalles de 100. L'objectif étant d'identifier des intervalles de haute densité de nombres premiers pour certaines échelles tel ci-dessus.**

## Quelques exercices complémentaires pour réviser

---

### La Saisie Sécurisée en Python

Sois indompté quelque soit l'échelle de ta progressée aux tests futiles. Utilises tu la fonction input sans te faire surprendre par un arrêt brutal de tes solutions au test par un tiers des cas non prévus de tes plages facilitées.

### J'espère que factorielle ne te fait vaciller quelque soit l'heure

Contraste les performances et la fiabilité de la solution factorielle itérative de celle récursive en Python.

### La suite récursive itérative de mémoire antépénultième

Soit la suite de fibonacci connu de tout bachelier aux sciences mathématiques et expérimentales. Programme le calcul d'un terme quelconque de la suite puis étudie la convergence de la suite. Tu commenceras par la solution récursive facile à écrire, puis solutionne itérativement.

---

## Sois insurmonté aux suites et séries programmables récursivement et itérativement

|                                                                                                     |
|-----------------------------------------------------------------------------------------------------|
| <b>Actualiser la solution quand la suite devient : <math>F(n+3) = F(n+2) + F(n+1) + F(n)</math></b> |
|-----------------------------------------------------------------------------------------------------|

### Vérifier une phrase équilibrée

Ecris un programme permettant de saisir une phrase dans laquelle les parenthèses, les crochets, les guillemets sont toujours équilibrés dans un langage expressif pour extraire les différentes propositions encadrées de parenthèses séparées de ET et de OU.

### La tour de Hanoï et sa difficulté itérative

Démontre graphiquement la solution du problème de la tour de Hanoï abordé en utilisant la récursivité. Une solution itérative a-t-elle un intérêt quelconque ?

### Problème de la monnaie au distributeur de billet

Va au GAB et ait le choix de minimiser le nombre de billet, ou de diversifier les couleurs des billets pour en avoir le choix.

# Problèmes d'approfondissement pour les pas si novices

---

## L'arbre recouvrant (Spanning-tree)

Solutionne efficacement l'élagage d'un graphe arbitraire en arbre i.e. sans boucle. L'application la plus emblématique est le spanning tree protocol, qui en ait une version distribuée et décentralisée utilisait dans les topologies de commutateur aussi appelé *bridges*. Le STP est important pour éviter les boucles de trame indéfiniment dans les réseaux Ethernet. Plusieurs versions de STP existent avec différentes performances de convergence.

## Le chemin le plus court dans un graphe

Router dans un réseau reposant sur un graphe de nœud nécessite de relayer suivant une métrique sur le chemin le plus court. Un des algorithmes emblématiques est celui de Dijkstra, l'alternative courante étant celle de Bellman-Ford. Implémente ta propre solution prenant en compte à certain nœuds la présence de solutions de chemins alternatifs vers la destination cible et compare ses performances sur des graphes judicieusement choisis.

## L'arbre et l'optimisation de la recherche dans une table de routage

Une table de routage dans la mémoire d'un routeur peut contenir 600 mille, à un million d'entrées d'adresse réseau, disons IPv4 agrégés suivant les préfixes commun.

Lorsqu'un paquet arrive au routeur, il consulte la table de routage avant de décider où relayer le paquet si le préfixe de sa destination se trouve en mémoire dans la table de routage.

Implémenter la table de routage sous forme d'une structure de données judicieusement choisi et une fonction `lookup` dont vous mesurerez le temps de solutionnement comparé à une simple solution basé sur dictionnaire Python ayant comme clé un préfixe de destination et comme valeur l'interface cible .

## La chance d'attention d'un contenu sur tes réseaux sociaux milliardimes et comment l'algorithmique t'y a salvaté ?

Quelle est la probabilité que votre dernière publication sur Instagram en réels avec 1 milliards d'utilisateurs si l'algorithme reste neutre sans la favoriser en dehors de vos propres abonnés ?

Simule le comportement d'un algorithme de recommandation sponsorisé pour avoir un certain nombre de vues maximisant le score d'attention.

# Problèmes des annales et solutions

---

## Projet IEEE 754

Réalise une interface permettant de coder et décoder des nombres flottants suivant la norme IEEE 754