NSI 1ère — Projet Python : Opérateur booléen universel

LFI Dubaï (AFLEC)
NSI — H. Salah

Travail en binôme — Durée totale : 5h en classe
Délai de remise : Dimanche 23 novembre, 20h00 (heure locale)

Objectif général. Concevoir et tester, en Python, un ensemble de fonctions réalisant des opérations binaires à partir d'opérateurs booléens : `additionneur_1_bit()`, `additionneur_complet()` et `multiplication()`.
Vous vous appuierez - si vous le souhaitez - sur le simulateur de circuits logiques suivant : https://logic.modulo-info.ch/

1. Organisation et calendrier (5h en présentiel)

Séance 1 **Jeudi 6 novembre (1h).** Lancement, répartition des rôles, recherche documentaire (additionneur 1 bit, opérateurs `and`, `or`, `xor`, `not`), prise en main du simulateur. Début du codage de `additionneur_1_bit()`.

Séance 2 **Mardi 11 novembre (2h).** Finalisation et tests unitaires de `additionneur_1_bit()`, implémentation de `additionneur_complet()` (addition de deux listes de bits), écriture des cas de tests.

Séance 3 **Jeudi 13 novembre (2h).** Implémentation de `multiplication()` par addition et décalages, validation par tests, préparation des livrables (Colab + document de présentation).

Règles de travail. Projet **en binôme**. On attend une **coopération effective** (répartition des tâches, relectures croisées). Le code doit être **commenté**, et testé. Toute source utilisée (schémas, pages web, vidéos) doit être **citées** dans le rapport.

2. Spécifications techniques (à respecter)

- Les bits sont représentés par des entiers `0` ou `1`. Les nombres binaires sont représentés par des **listes de bits**, bit de poids faible en **position 0** (ex. $6_{10} = 110_2$ est `[0,1,1]`).
- **Interdit** d'utiliser directement `+` sur des entiers pour calculer les sommes de bits. On attend une construction à partir d'opérateurs booléens (`and`, `or`, `xor`, `not`) et de logique de retenue.

2.1 Fonction `additionneur_1_bit(a,b,r_e)`

Entrées : $a, b, r_e \in \{0, 1\}$. **Sorties :** un **couple** (s, r_s) avec s la somme et r_s la retenue sortante.

2.2 Fonction `additionneur_complet(A,B)`

Entrées : deux listes de bits `A` et `B`. **Sortie :** la **liste de bits** `S` représentant $A + B$.
Contraintes : itérer sur les positions, propager la retenue en appelant `additionneur_1_bit()`.

2.3 Fonction `multiplication(A,B)`

Entrées : deux listes de bits `A` et `B`. **Sortie :** la **liste de bits** `P` représentant $A \times B$.
Principe attendu : algorithme "*shift-and-add*" : pour chaque bit de `B`, additionner `A` décalé si le bit vaut 1, en utilisant `additionneur_complet()`.

3. Squelettes de code (à compléter dans Google Colab)

Copiez ces squelettes dans vos notebooks Colab (un notebook commun peut contenir les trois fonctions et les tests).

fichier : `NSI_addition_binome_NOM1_NOM2.ipynb`

Signatures & docstrings :

```
from typing import List, Tuple

def additionneur_1_bit(a, b, r_e):
    ...
    ...
    ...

    Args:
        a, b, r_e: bits 0 ou 1
    Returns:
        (s, r_s): somme 1 bit et retenue sortante, tous deux 0 ou 1

    Exemples:
        >>> additionneur_1_bit(1, 0, 0)
        (1, 0)
        >>> additionneur_1_bit(1, 1, 0)
        (0, 1)
    """
    # TODO: coder avec and / or / xor / not

def additionneur_complet(A, B):
    ...
    ...
    ...

def multiplication(A, B):
    ...
    ...
    ...
```

4. Exemples attendus (oracles)

- `additionneur_1_bit(0,0,0)` → `(0,0)`
- `additionneur_1_bit(1,1,0)` → `(0,1)`
- `additionneur_1_bit(1,1,1)` → `(1,1)`
- `additionneur_complet([0,1,1],[1,0,1])` → `[1,1,0,1]` $(6 + 5 = 11)$
- `multiplication([0,1,1],[1,0])` → `[0,0,1,1]` $(6 \times 2 = 12)$

5. Tests à produire (obligatoires)

Vous devez fournir un bloc de tests exécutables. Couvrir au minimum :

a) Application de `additionneur_1_bit` sur 8 cas/test.

b) Au moins 6 cas pour `additionneur_complet` dont : tailles différentes, retenue finale, zéros, grandes tailles.

c) Au moins 6 cas pour `multiplication` dont : par 0, par 1, puissances de 2, nombres aléatoires.

6. Livrables & format d'envoi

À remettre avant Dimanche 23 novembre, 20h00 (heure locale) (aucun retard accepté).

1. **Code sur Google Colab** : un notebook `.ipynb` commun nommé `NSI_addition_binome_NOM1_NOM2.ipynb` contenant :
   — les trois fonctions finalisées ;
   — les cellules de tests (exécutables) avec captures de sorties si besoin.
2. **Document de présentation (PDF, 1–3 pages)** nommé `NSI_rapport_NOM1_NOM2.pdf` :
   — **méthodologie** (répartition des rôles, difficultés rencontrées, validation) ;
   — **schéma** ou capture du simulateur *logic.modulo-info.ch* de l'additionneur 1 bit (facultatif) ;
   — **justification** des formules de somme et de retenue ;
   — **lien partage Colab** (lecture).
3. **Remise** : envoyer à Monsieur SALAH par email : hamsa.salah@aflec-fr.org . L'email doit contenir *les deux fichiers* (`.ipynb` et `.pdf`). Les liens Colab doivent être accessibles sans demande de permission.

7. Barème détaillé (20 points)

Critères	Pts
Recherche & compréhension : table de vérité, schéma logique, choix de représentation des bits	2
Additionneur 1 bit : exactitude, respect des opérateurs booléens, clarté	5
Additionneur complet : propagation correcte de la retenue, gestion tailles, propreté	4
Multiplication : algorithme shift-and-add correct, réutilisation de l'additionneur	4
Tests : couverture exigée, lisibilité, pertinence des cas limites	2
Reporting (PDF) : structuration, explications, citations des sources, liens valides	2
Travail de groupe : organisation, répartition, journal de bord succinct (dans le PDF)	1
Total	**20**

Pénalités éventuelles (cumulables) : non-respect du format de fichiers / noms (-0,5), liens Colab inaccessibles (-1), absence de tests (-2), utilisation d'opérateurs arithmétiques interdits pour l'addition 1 bit (-2), plagiat/IA non sourcée et/ou non compris (-*jusqu'à* -20). **Tout dépôt après Dimanche 23 novembre, 20h00 (heure locale) = note 0.**

« Comprendre la logique sous-jacente permet d'écrire du code fiable. »