

A. Les booléens :

Ce qu'il faut savoir :

- **George Boole**, un mathématicien et philosophe britannique du XIX^e siècle, créa de 1844 à 1854 une algèbre binaire, dite booléenne, n'acceptant que deux valeurs numériques : 0 et 1.

Cette algèbre aura de nombreuses applications en téléphonie et en informatique.



George Boole

- Un booléen est un type de variable (« **bool** ») qui n'a que deux états qui sont en général **True** (représenté par un bit 1) ou **False** (représenté par un bit 0).
- Sur Python, il ne faut pas confondre le symbole « **=** » qui sert à affecter une valeur à une variable avec le symbole « **==** » qui sert à tester une égalité.

Exemple : Si on tape sur la console de Python :

```
>>>a=2
>>>b=3
>>>a=b
```

Ici on affecte la valeur 2 à *a* , puis 3 à *b* et enfin on affecte la valeur de *b* à *a* .
(à la fin *a* vaut 3)

Si on tape sur la console de Python :

```
>>>a=2
>>>b=3
>>>a==b
```

Ici on affecte la valeur 2 à *a* , puis 3 à *b* et enfin on teste si la valeur de *a* est égale à celle de *b* .
(ici *a == b* est un booléen qui a la valeur False)

- Sur Python, « **==** » est un opérateur de comparaison.
Les autres opérateurs de comparaison sont : « **!=** » (différent de), « **<** » (strictement inférieur), « **>** » (strictement supérieur), « **<=** » (inférieur ou égal) et « **>=** » (supérieur ou égal).

Activité 1 : Pour chaque cas, détermine ce que va afficher Python (après avoir répondu, corrige toi en tapant les instruction sur la console de Python).

1. >>>a=3
>>>b=5
>>>a>b

2. >>>a=3
>>>b=5
>>>a!=b

3. >>>type(5)==str

4. >>>5=="5"

5. >>>5==5.0

6. >>>type(5)==type(5.0)

7. >>>var1="vive la nsi"

8. >>>var1="vive la nsi"

>>>var2="nsi"

>>>var2="nsi"

>>>var2 in var1

>>>var1 in var2

9. >>>a="j'aime"
>>>b="la nsi"
>>>c=a+b
>>>c=="j'aime la nsi"

10. >>>a=-5.0
>>>b=4.2
>>>a*b>4*a-1

remarque : « **in** » se traduit par « **dans** » en Anglais

Ce qu'il faut savoir : Les opérateurs booléens.

- L'opérateur « **not** » (qui signifie « non ») est le booléen **opposé**.

Voici **la table de vérité** qui définit cet opérateur pour un booléen a :

a	False	True
not a	True	False

a	0	1
not a	1	0

Remarque : quand le booléen est défini avec les chiffres 0 et 1, alors **not a = 1 - a**.

- L'opérateur « **and** » (qui signifie « et ») teste si deux booléens sont vrais **simultanément**.

Voici la table de vérité qui définit cet opérateur pour deux booléens a et b :

a and b	False	True
False	False	False
True	False	True

a and b	0	1
0	0	0
1	0	1

Remarque : quand les booléens sont définis avec les chiffres 0 et 1, alors **a and b = a x b**.

- L'opérateur « **or** » (qui signifie « ou ») teste si **au moins un des deux** booléens est vrai.

Voici la table de vérité qui définit cet opérateur pour deux booléens a et b :

a or b	False	True
False	False	True
True	True	True

a or b	0	1
0	0	1
1	1	1

Remarque : quand les booléens sont définis avec les chiffres 0 et 1, alors **a or b = a + b - a x b**.

Activité 2 : Pour chaque cas, détermine ce que va afficher Python (après avoir répondu, corrige toi en tapant les instruction sur la console de Python).

1. `>>>(8>4 or 1>2)`

2. `>>>a=True`
`>>>b=False`
`>>>not(a or b) or (a and not(b))`

3. `>>>a=3`
`>>>b=-7`
`>>>a**3>50 and b**2<50`

4. `>>>a=3`
`>>>b=-7`
`>>>(a**3>50 and b**2<50)`
`or (a**2<10 and b**2>10)`

remarque : « ****** » est le symbole pour l'exposant avec Python.

5. `>>>mot="science"`
`>>>mot[0]=="s"`

6. `>>>mot="science"`
`>>>mot[0]=="s" and mot[5]=="n"`

7. `>>>mot="science"`
`>>>not(mot[2:5]=="ien")`

8. `>>>var1="vive la nsi"`
`>>>var2="vive le prof"`
`>>>"vive" in (var1 and var2)`

remarque : Si x est une variable de type « string » sur Python, alors x[i] est la lettre de rang i, soit la (i+1)^è lettre de la chaîne de caractères.

B. Détermination des tables de certaines fonctions booléennes :

1. a. Recopie et complète le tableau ci-dessous.

a	b	a and b	b and a	a or b	b or a
False	False				
False	True				
True	False				
True	True				

b. Que peux-tu en déduire pour les booléens (a and b) , puis (b and a) ? Et pour (a or b), puis (b or a) ?

2. Sur le même modèle, construire un tableau pour chacune des questions et en tirer des conclusions :
- a. (a and True). b. (a or False) . c. (a and (b or c)) puis ((a and b) or (a and c)).
 - d. (a or (b and c)) puis ((a or b) and (a or c)). e. (a and not(a)). f. (a or not(a)).
 - g. not(not(a)). h. ((a and b) and c) puis (a and (b and c)).
 - i. ((a or b) or c) puis (a or (b or c)). j. (a and (a or b)) puis a.
 - k. (a or a and b) puis a. *remarque* : comme la multiplication est prioritaire par rapport à l'addition, and est prioritaire par rapport à or.
 - l. not(a and b) puis (not(a) or not(b)). m. not(a or b) puis (not(a) and not(b))

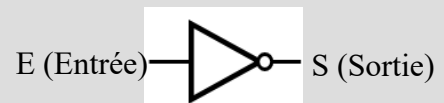
3. Résume les informations découvertes à travers cet exercice (ce sont les théorèmes sur les booléens).

C. Portes logiques :

Ce qu'il faut savoir :

- Les circuits d'un ordinateur manipulent uniquement des chiffres binaires (0 et 1) qui, techniquement en interne, sont simplement représentés par des tensions électriques. Ainsi, le chiffre 0 est représenté par une tension basse proche de 0 Volt et le chiffre 1 par une tension haute qui dépend des circuits.
- Les composantes qui vont permettre de faire des opérations sur ces chiffres binaires sont appelées **portes logiques**. Ces portes logiques sont des circuits électroniques composés de **transistors** qui se comportent comme **des interrupteurs qui laissent passer ou non le courant électrique**.
- Les portes logiques élémentaires sont :

- la porte « not »



- la porte « and »



- la porte « or »



Activité 1 :

Les entrées et sorties sont des tensions électriques représentant des booléens 0 (pour False) et 1 (pour True).

1. On veut créer une porte logique « not » sur le logiciel Logisim et simuler son fonctionnement.

Pour cela observe la vidéo suivante : <https://www.youtube.com/watch?v=gCOsSLtMogo>

Ouvre le logiciel Logisim et réalise ce travail.

2. Sur la même page du logiciel Logisim, crée une porte logique « and », puis une porte logique « or » et simule leur fonctionnement.

3. Complète alors les tables de vérité ci-dessous :

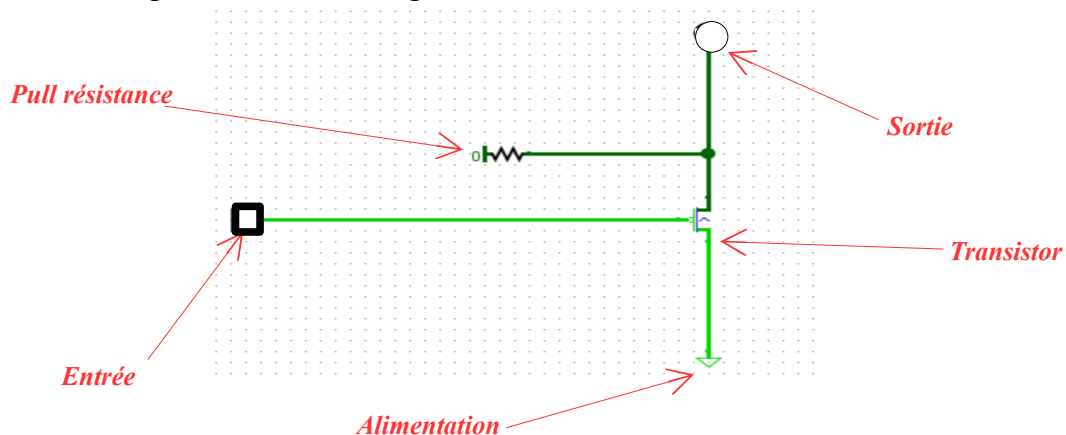
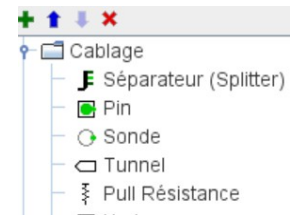
not	
E	S
0	
1	

and		
E1	E2	S
0	0	
0	1	
1	0	
1	1	

or		
E1	E2	S
0	0	
0	1	
1	0	
1	1	

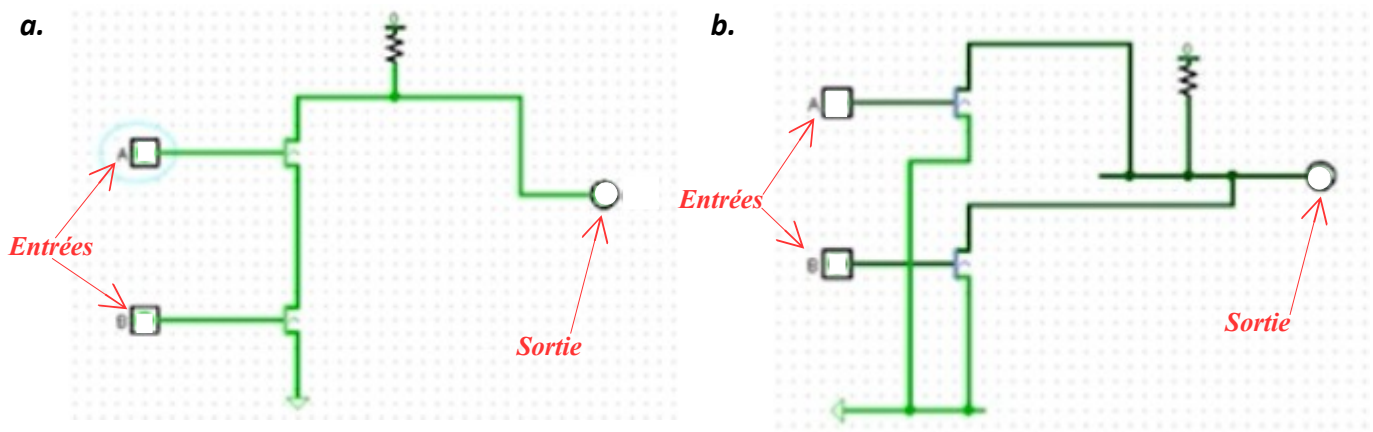
Activité 2 : On veut maintenant construire ces 3 portes logiques à l'aide de transistors avec le logiciel Logisim.

1. **a.** Ouvre un nouveau fichier Logisim et réalise le circuit ci-dessous en utilisant les composants accessibles dans le dossier « câblages » visible ci-contre et accessible en haut à gauche du fichier Logisim.



- b.** A l'aide de cet outil Simule le fonctionnement de ce circuit en mettant l'entrée à 0, puis 1. Quelle porte logique reconnaît-on ?
- c.** En cliquant sur « **Fichier** » puis « **Sauvegarder comme...** », enregistre ton travail dans le dossier **Partage (R) : NSI : Gauthier** en lui donnant le nom : « **NOM.fichier1** ».

2. Réalise de même chacun des deux circuits ci-dessous et précise à quelles porte logique correspond chacun d'entre eux.



- c. En cliquant sur « **Fichier** » puis « **Sauvegarder comme...** », enregistre tes travaux dans le dossier **Partage (R) : NSI : Gauthier** en leur donnant le nom : « **NOM.fichier2** » et « **NOM.fichier3** ».

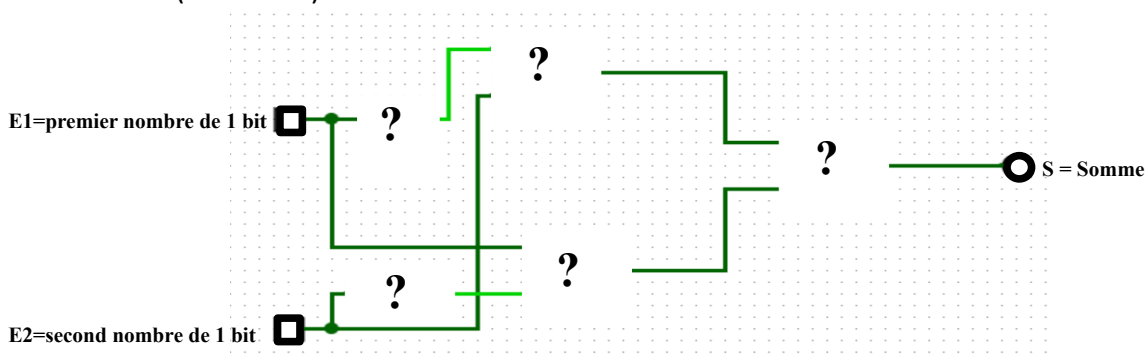
Activité 3 : On veut additionner 2 nombres binaires de 1 bit. Pour cela on va construire 2 fonctions logiques S (pour Somme) et R (pour Retenue) qui prennent en entrées E1 et E2 les 2 nombres binaires.

1. Recopie et complète la table de vérité de chacune de ces fonctions logiques :

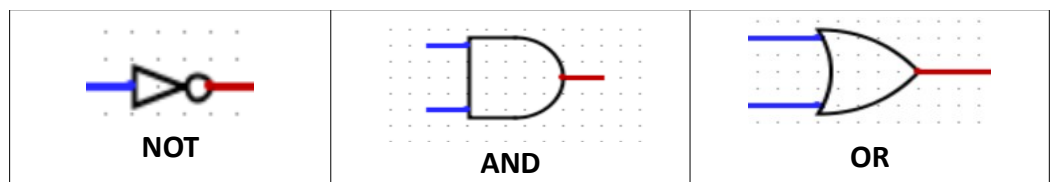
SOMME		
E1	E2	S
0	0	
0	1	
1	0	
1	1	

RETENUE		
E1	E2	R
0	0	
0	1	
1	0	
1	1	

2. a. A quel opérateur correspond la fonction logique qui donne la retenue ?
 b. Pour la fonction logique qui donne la somme, dans quels cas S est à 1 (True) ?
 c. Partant de E1=True (1) et E2=True (1), écris une expression logique avec E1 et E2 traduisant ta réponse du b.
 d. A l'aide de ce qui vient d'être fait, complète chaque « ? » par une des portes logiques utilisables (données ci-dessous) ce circuit dans lequel on a 2 entrées (les nombres binaires E1 et E2) et la sortie S (la somme).



Portes logiques utilisables
(une ou plusieurs fois) :



- e. Ouvre un nouveau fichier Logisim, puis à l'aide des questions précédentes, reproduis le circuit de la question c. et complète le afin qu'il permette d'obtenir les résultats des fonctions logiques S (Somme), mais aussi R (Retenue). Ce circuit se nomme **un demi-additionneur sur 1 bit.**

Par un clic sur le circuit nommé « main » en haut à gauche de la page, fait apparaître le nom « **main** » du circuit, que tu vas remplacer par « **demi_add** ».
 Veille à faire apparaître E1, E1 et S pour les entrées et la sortie en cliquant dessus et en inscrivant ces noms dans le label.

VHDL	Verilog
Représentation	Est
Emplacement du label	Nord
Police du Label	SansSerif Gras 16
Label Visible	Oui
Nom du circuit	main
Label partagé	

3. On veut maintenant additionner 2 nombres binaires de 1 bit en tenant compte de la retenue entrante. Pour cela on va construire 2 fonctions logiques S (pour Somme) et R_s (pour Retenue sortante) qui prennent en entrées E1 et E2 les 2 nombres binaires, et R_e (pour retenue entrante).

Recopie et complète la table de vérité de chacune de ces fonctions logiques :

SOMME			
R_e	E1	E2	S
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

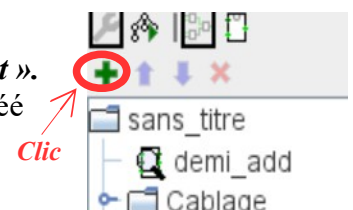
RETENUE sortante			
R_e	E1	E2	R_s
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

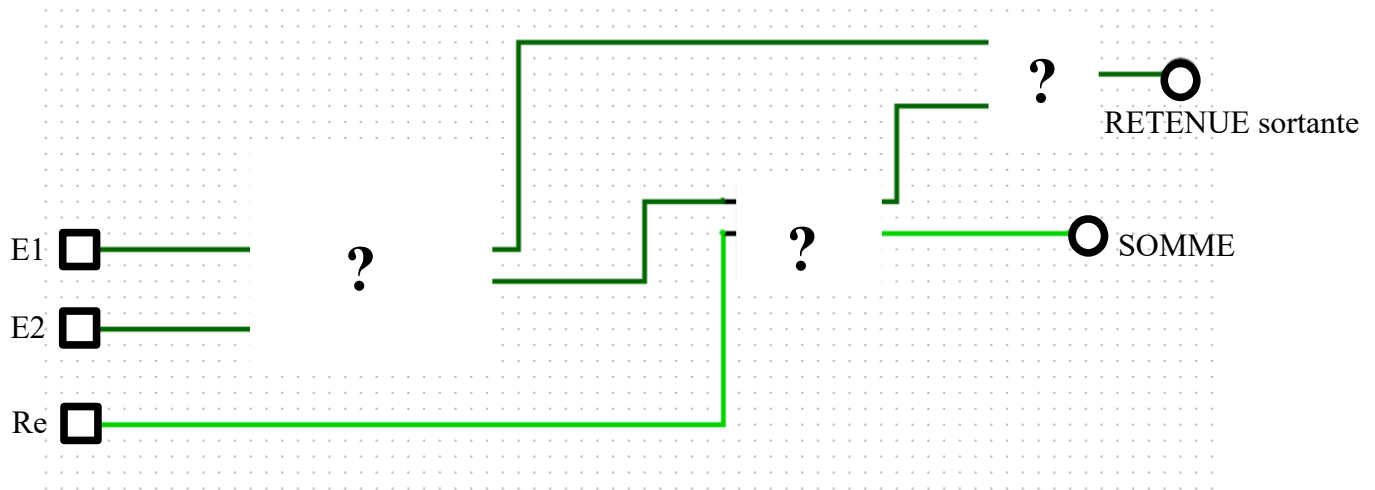
4. a. On va commencer par additionner (avec un demi-additionneur) E1 et E2 et on obtient une première somme intermédiaire S_i et une première retenue intermédiaire R_i . Il faut évidemment encore additionner (avec un autre demi-additionneur) S_i et R_e afin d'obtenir une seconde somme intermédiaire s_i et une seconde retenue intermédiaire r_i .

Recopie et complète le tableau ci-dessous :

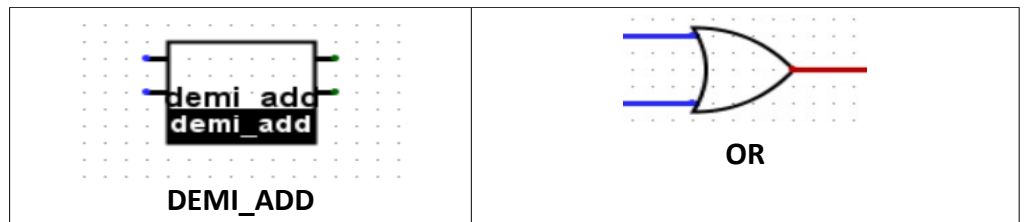
R_e	E1	E2	S_i	R_i	s_i	r_i
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

- b. En observant attentivement les deux derniers tableaux, compare la seconde somme intermédiaire s_i et la SOMME cherchée. Puis trouve un lien logique entre R_i , r_i et le RETENUE SORTANTE cherchée.
- c. Sur le fichier Logisim de la question 3, par un clic sur le « + » vert en haut à gauche de la page, tu vas ajouter une circuit que tu nommeras « **add_complet** ». Puis complète le circuit ci-dessous où l'on a utilisé le circuit « **demi_add** » créé à la question 2.



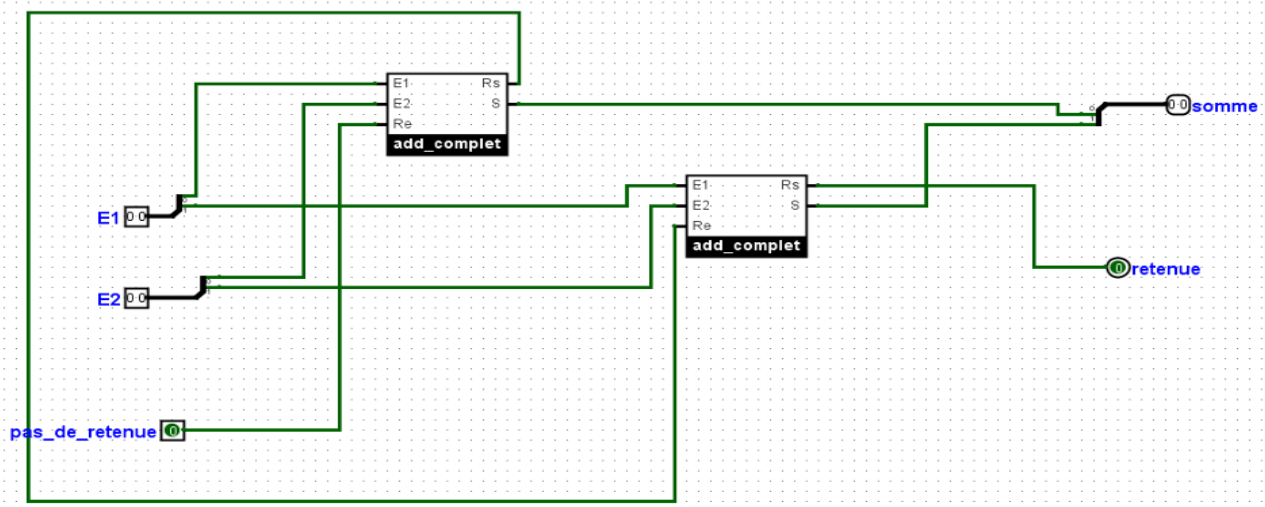


éléments utilisables
(une ou plusieurs fois) :



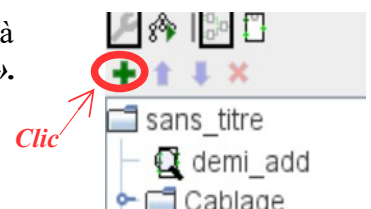
d. A l'aide des questions précédentes, reproduis sur Logisim ce circuit dans le circuit « **add_complet** ». Ce circuit se nomme un additionneur complet sur 1 bit.

5. On s'intéresse maintenant au circuit cidessous, où E1, E2 et S représentent des nombres binaires de 2 bits:



a. Explique en quelques phrases le fonctionnement de ce circuit.

b. Sur le fichier Logisim de la question 3, par un clic sur le « + » vert en haut à gauche de la page, tu vas ajouter une circuit que tu nommeras « **add_2bits** ». Puis reproduis le circuit précédent et enregistre ce fichier dans le dossier **Partage (R) : NSI : Gauthier** en le nommant: « **NOM.fichier3** ».



c. Teste ce circuit et reporte les résultats de tests dans le tableau ci-dessous que tu compléteras.

E1	E2	somme	retenue
(0,0)	(0,0)	(0,0)	0
(0,0)	(1,0)
etc...	etc...	etc...	etc...

A. Algorithmique - Premiers algorithmes:

Définition : Un **algorithme** est une liste d'instructions à exécuter pas à pas pour accomplir une tâche précise.

Remarques

- Un algorithme doit être lisible de tous ; Son intérêt est d'être codé dans un langage informatique pour qu'une machine (ordinateur, calculatrice, etc.) puisse l'exécuter rapidement et efficacement.
- Les trois phases d'un algorithme sont :
 - l'entrée des données ;
 - le traitement des données ;
 - la sortie des résultats.

Ex 1 - On considère l'algorithme suivant :

Choisir un nombre de départ	3					
Lui ajouter 1	4					
Multiplier le résultat par 2	8					
Soustraire 3 au résultat et afficher le résultat final	5					

- 1) Recopier le tableau ci-dessus, puis :
 - a) Appliquer successivement cet algorithme à -4 , 0 , $1/3$
 - b) Déterminer les nombres à choisir au départ pour afficher à la fin 0 puis 7
- 2) Écrire un nouvel algorithme qui permette, en partant du résultat final de l'algorithme ci-dessus, de retrouver le nombre de départ.
- 3) Traduire chacun de ces deux algorithmes par une formule en fonction du nombre de départ x .
Quelle est la nature des deux fonctions trouvées ?

Ex 2 - On considère l'algorithme suivant :

Choisir un nombre
Calculer le carré de ce nombre
Multiplier par 10
Ajouter 25
Afficher le résultat

- 1) En faisant un tableau comme celui de l'exercice I), appliquer cet algorithme à 2 , puis $\sqrt{2}$
- 2) Traduire cet algorithme par une formule en fonction de x .
- 3) Pénélope affirme que si le nombre choisi au départ est un entier alors le résultat est impair.
A-t-elle raison ? Justifier.
- 4) Ulysse affirme que le résultat est toujours positif quelque soit le nombre choisi au départ.
A-t-il raison ? Justifier.

B. Le langage Python:

Les algorithmes sont traduits en programmes informatiques à partir des différents langages de programmation qui existent. Dans ce chapitre, nous utiliserons **Python** comme langage de programmation.

1. Éléments de base

a. Les types

Le type d'une variable définit l'ensemble des valeurs qui peuvent lui être affectées. Les types de base sont **int**, **bool**, **float** qui sont des types numériques et **str**.

Ce qu'il faut savoir :

- ✓ Le type **int**, abréviation de integer, est utilisé pour représenter les **nombre entiers**.
- ✓ Le type **bool** permet de représenter les valeurs booléennes True (vrai) et False (faux).
- ✓ Le type **float** est utilisé pour représenter les **nombre réels**. Attention, en Python, le **point** remplace la **virgule** utilisée en mathématiques (donc 5,8 est représenté par 5.8 en Python).
- ✓ Le type **str**, abréviation de string, est utilisé pour représenter des chaînes de caractères. On utilise des guillemets, des apostrophes ou str comme `a='bonjour'`, `a="bonjour"` ou `a=str(bonjour)`.

b. Les opérations sur les types numériques

Les opérations de base

+	addition
-	soustraction
*	multiplication
/	division
**	puissance
//	division entière
%	reste de la division

```
15 // 6
```

Réponse : 2

```
15 % 6
```

Réponse : 3

L'écriture scientifique

Exemple :

```
2.75e3
```

Réponse : 2750.0

- **Les opérateurs mathématiques classiques de comparaisons** `=`, `≠`, `<`, `≤`, `>`, `≥` s'écrivent en Python respectivement `==`, `!=`, `<`, `<=`, `>`, `>=`.
- **Les opérations logiques :**
 - **a and b** prend la valeur **True** si **a** et **b** sont **True** et sinon prend la valeur **False** ;
 - **a or b** prend la valeur **False** si **a** et **b** sont **False** et sinon prend la valeur **True** ;
 - **not a** prend la valeur **True** si **a** est **False** et prend la valeur **False** si **a** est **True**.

Remarques :

- Sur Python, pour utiliser des fonctions mathématiques comme la racine carré, il faut l'importer à partir du module math avec l'instruction suivante `from math import sqrt` (sqrt étant la racine carré en python)
- Pour importer toutes les fonctions du module math, la syntaxe est `from math import *` (à mettre au début du programme : 1^{ère} ligne en général)

Ce qu'il faut savoir : En général, un **algorithme** est construit en trois étapes :

- **Entrée** : On saisit les données.
- **Initialisation** : Le programme attribue des valeurs initiales à des variables. Ces valeurs peuvent changer au cours du programme.
- **Traitement des données** : Les instructions du programme effectuent des opérations à partir des données saisies dans le but de résoudre un problème.
- **Sortie** : Les résultats sont affichés

2. Affectation de variables

Définition :

Une **variable** est composée d'un nom (ou identificateur), d'une adresse en mémoire où est enregistrée une valeur ou un ensemble de valeurs) et d'un type qui définit ses propriétés.

Une **variable** permet de stocker une valeur que l'on compte réutiliser plus tard.

Les instructions de base sur les variables sont les suivantes :

- la **saisie** : on demande à l'utilisateur de donner une valeur à une variable ;
- l'**affectation** : est une instruction qui commande à la machine de créer une variable en lui précisant son nom et sa valeur) ;
- l'**affichage** : on affiche la valeur de la variable.

○

Remarques

Les notations ci-dessous sont équivalentes :

- Affecter à b la valeur de a
- b prend la valeur de a

En Python

- L'**affectation** se fait avec le signe `=` (**exemple :** `u=5` : on affecte ici la valeur 5 à la variable u)
- La **saisie** se fait avec la commande `input ()` : elle permet d'inviter l'utilisateur à saisir une valeur à l'exécution du programme.

Exemple :

```
nom = input("Quel est  
votre nom ?")
```

Remarque :

input est une fonction qui renvoie toujours une chaîne de caractères. Pour changer le type de la variable, on utilise : int (pour les entiers) ou float (pour les réels). Voir l'exemple ci-dessous :

```
n = float(input("Entrer un nombre"))  
print("Le carré de", n,"est", n * n)
```

- L'affichage se fait avec la commande **print ()**

```
print("Bonjour !")
```

```
print(2)
```

```
a = -3  
print("Le carré de", a,"est", a * a)
```

Remarque :

La commande **print ()** permet d'afficher textuellement la partie entre les guillemets " " et d'afficher la valeur des variables en dehors des guillemets. En effet dans le dernier exemple ci-dessus, il sera affiché : Le carré de -3 est 9.

Ex 3 - On donne l'algorithme suivant :

```
Lire  $n$   
 $q$  prend la valeur de  $(n + 2) \times (n + 2)$   
 $q$  prend la valeur de  $q - (n + 4)$   
 $q$  prend la valeur de  $q / (n + 3)$   
Afficher  $q$ 
```

- 1) Tester cet algorithme pour $n = 4$, puis pour $n = 7$.
- 2) Ecrire le programme python correspondant à cet algorithme (en considérant n comme un réel) et tester sur Python pour $n = 6,2$ puis pour $n = -9,8$
- 3) Émettre une conjecture sur le résultat fourni par cet algorithme puis démontrer cette conjecture.
- 4) Un élève a saisi $n = -3$. Que se passe-t-il ? Pourquoi ?

Ex 4 - Un commerçant accorde une remise sur des articles. On souhaite connaître le montant de la remise en euros. Voici un algorithme écrit en langage naturel donnant la solution au problème :

Entrée

Saisir le prix de départ A
Saisir le pourcentage de remise P

Traitement des données

Affecter au montant de la remise R la valeur $A \times \frac{P}{100}$

.....

Sortie

Afficher R

.....

- 1) Calculer la valeur de la variable R lorsque A = 56 et P = 30.
Donner une interprétation concrète du résultat précédent.
- 2) Compléter l'algorithme pour qu'il affiche également le prix à payer B.
- 3) Traduire l'algorithme en langage Python puis calculer la valeur des variables R et B lorsque A = 159 et P = 24. Donner une interprétation concrète des résultats précédents.
- 4) Même question avec A = 13 et P = 45

Exercice 5 :

- 1) Rédiger en langage naturel puis en langage Python un algorithme permettant de calculer le pourcentage de réduction d'un article connaissant le prix de départ et le prix à payer.

Entrée

.....
.....
.....

Traitement des données

.....
.....
.....

Sortie

.....
.....

- 2) Calculer avec le pourcentage de réduction d'un article dont le prix de départ est de 75 euros et le prix final est de 49,5 euros.

Définition : La résolution de certains problèmes nécessite la mise en place d'un **test** pour savoir si l'on doit effectuer une tâche.

Si la condition est remplie alors la tâche sera effectuée, sinon on effectue (éventuellement) une autre tâche. Dans un algorithme, on code la structure du « Si... AlorsSinon... FinSi » sous la forme suivante :

```
Si      condition Alors
        Tâche 1
        Tâche 2
Sinon
        Tâche 1bis
        Tâche 2bis
FinSi
```

Remarques :

- A chaque « Si » doit correspondre un « Fin Si ».
- En revanche le « Sinon » n'est pas toujours obligatoire. S'il n'est pas présent, aucune tâche ne sera effectuée si la condition n'est pas remplie.
- Il est important de respecter les espaces laissés au début de chaque ligne car ils permettent une meilleure lisibilité de l'algorithme.

En Python

- La commande **if** permet de tester le contenu d'une variable et exécute une série d'instructions si les conditions sont remplies.
- L'indentation, décalage vers la droite du début de ligne, est un élément de syntaxe important en Python . Elle délimite des blocs de code et elle aide à la lisibilité en permettant d'identifier facilement ces blocs. La ligne précédant l'indentation se termine par le signe deux-points .

La structure la plus simple est :

```
if condition:
    instructions
```

Le mot condition désigne une expression et le mot instructions désigne une instruction ou un bloc d'instructions écrites sur plusieurs lignes. Voici un exemple :

```
if n % 2 == 0:
    n = n // 2
```

La structure if-else présente une alternative :

```
if condition:
    instructions1
else:
    instructions2
```

Par exemple :

```
if n % 2 == 0:
    n = n // 2
else:
    n = 3 * n + 1
```

La structure if-elif-else présente plusieurs alternatives :

```
if condition1:
    instructions1
elif condition2:
    instructions2
elif condition3:
    instructions3
...
else:
    instructions
```

Par exemple :

```
if n % 4 == 0:
    n = n // 4
elif n % 4 == 1:
    n = (3 * n + 1) // 4
elif n % 4 == 2:
    n = n // 2
else:
    n = (3 * n + 1) // 2
```

Remarques :

- Les lignes qui commencent par les mots if et elif se terminent par le signe deux-points ;
- Le mot else est suivi immédiatement par le signe deux-points ;
- C'est l'indentation qui permet de délimiter les blocs d'instruction à exécuter si la condition est vérifiée.

Exercices d'application : test if

Exercice 1

Afficher « Entrez les dimensions du triangle en commençant par la plus grande »
Lire a , b et c
Si
 Afficher « Le triangle est rectangle. »
Sinon
 Afficher « Le triangle n'est pas rectangle. »
FinSi

- 1) Quelle condition faut-il écrire après le « Si » ?
- 2) Traduire cet algorithme en langage python puis tester pour $a=5$, $b=4$ et $c=3$
- 3) Modifier cet algorithme de façon à afficher en plus « Données incorrectes » si a n'est pas le plus grand des trois nombres rentrés.
- 4) Traduire ce nouvel algorithme en langage python puis tester pour $a=4$, $b=5$ et $c=3$.
Puis pour $a=7$, $b=4$, $c=6$ et enfin tester pour $a=8,5$; $b=3,6$ et $c=7,7$.

Exercice 2 :

Ex 5 - Un particulier souhaite louer une voiture.

- L'agence de location A demande 100 € au départ et 0,20 € par kilomètre parcouru ;
- L'agence de location B demande 150 € au départ et 0,15 € par kilomètre parcouru.

- 1) Compléter cet algorithme pour aider le consommateur à trouver la formule la plus avantageuse
- 2) Traduire cet algo en langage python puis donner la meilleure formule quand on veut parcourir 1000km, puis pour 525,75 km et enfin pour 1500km.

Afficher « Entrer le nombre de kilomètres. »
Lire n
Affecter à a la valeur $100 + 0,20 \times n$
Affecter à b la valeur
Si
 Afficher « La formule A est plus avantageuse. »
Sinon
 Si
 Afficher « Les formules A et B sont équivalentes. »
 Sinon
 Afficher « La formule B est plus avantageuse. »
FinSi
FinSi

Exercice 3

Un magasin propose de tirer des photos sur papier au tarif de 0,16 € la photo pour les 75 premières photos, puis 0,12 € la photo pour les photos suivantes. Écrire un algorithme qui demande à l'utilisateur d'entrer le nombre n de tirages photos commandés et calcule le montant à payer. Traduire l'algorithme en langage python puis tester avec des valeurs inférieures à 75 puis avec des valeurs supérieures à 75

Exercice 4

Écrire un algorithme qui demande deux nombres, puis affiche la différence entre le plus grand et le plus petit. Traduire en Python puis tester...

Définitions : Les boucles permettent de répéter plusieurs fois une partie de l'algorithme.

Il existe deux sortes de boucles :

1. Boucle « Pour » :

Lorsque **le nombre de répétitions n est connu à l'avance**, on utilise un compteur initialisé à 1 et qui augmente automatiquement de 1 à chaque répétition jusqu'à n . On parle de boucle itérative.

Pour Variable allant de Valeur début à Valeur fin

Tâche 1

Tâche 2

Fin Pour

En Python :

La syntaxe est la suivante :

```
for i in range(n):
    instructions
```

La variable i est appelé un compteur. Elle prend successivement les valeurs 0, 1, ..., $n-1$.

Exemple 1: Ecrivez un programme qui copierait 20 fois « je ne dois pas mâcher du chewing-gum »

```
for i in range(1,21):
    print("je ne dois pas mâcher du chewing-gum")
print("fini")
```

ou

```
for i in range(0,20):
    print("je ne dois pas mâcher du chewing-gum")
print("fini")
```

2. Boucle « Tant que » :

Lorsque **le nombre de répétitions n n'est pas connu à l'avance**, il peut dépendre d'une condition. Le traitement est répété tant que la condition est vraie. Lorsqu'elle est fausse, on sort de la boucle. On parle de boucle conditionnelle.

Tant que Condition est vraie

Tâche 1

Tache 2

Fin Tant que

En Python :

La syntaxe est la suivante :

```
while condition:
    instructions
```

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
print('Fini !')
```

Exemple 1 :

Que fait ce programme ?

Exemple 2 :

Les deux programmes suivant sont équivalents :

```
a=0
while a<10:
    print("boucle Tant que")
    a=a+1
print("Fin du programme")
```

```
for loop in range(10):
    print("boucle for")
print("Fin du programme")
```

Que font ces programmes ?

Les fonctions en Python

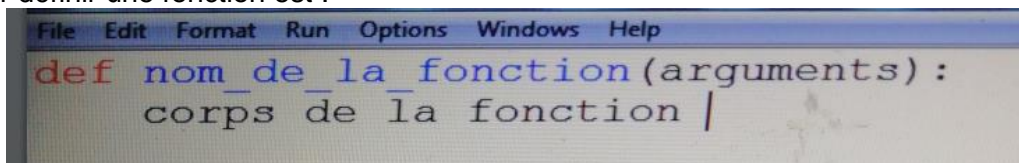
Ce qu'il faut savoir : Dans un programme, il est possible d'écrire des petits programmes ou des sous-programmes intermédiaires appelés *fonctions*.

Définition : Une fonction est un programme qui porte *un nom* et utilise zéro, une ou plusieurs variables appelés *paramètres ou arguments*.

Remarques :

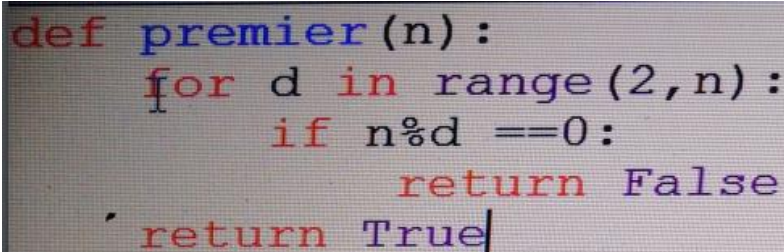
- Le nom d'une fonction ne peut pas avoir d'espace. On met donc des tirets « _ ».
- Si la fonction n'utilise aucun paramètre, on la note **def nom_de_la_fonction()**
- Les arguments (paramètres) sont séparés par des virgules. La première ligne, ligne de définition, se termine par le signe deux-points. Le corps de la fonction est un bloc indenté par rapport à la ligne de définition.
- L'instruction *return* permet de renvoyer une valeur de type entier, décimal (dit flottant) ou chaîne de caractères qui peut être réutilisée dans un autre programme ou une autre fonction. Elle interrompt le programme dès qu'elle s'est exécutée.

La syntaxe pour définir une fonction est :



```
File Edit Format Run Options Windows Help
def nom_de_la_fonction(arguments):
    corps de la fonction |
```

Exemple de fonction :



```
def premier(n):
    for d in range(2,n):
        if n%d ==0:
            return False
    return True
```

Que fait cette fonction ?

Exercices : Boucles « Pour »

Ex 1 - La somme des n premiers nombres entiers :

1	Lire n	1) Tester cet algorithme pour $n = 3$ à l'aide d'un tableau où on donnera les valeurs de i et s après chaque exécution de la ligne 4.
2	Affecter à s la valeur 0	2) Traduire cet algorithme en langage Python puis tester cet algorithme pour $n = 5$ puis pour $n=100$.
3	Pour i allant de 1 à n	
4	Affecter à s la valeur $s + i$	
5	Fin Pour	
6	Afficher s	

Ex 2 - On considère l'algorithme ci dessous :

1	Lire a et n	1) Tester cet algorithme pour $a = 2$ et $n = 3$ à l'aide d'un tableau où on donnera les valeurs de i et p après chaque exécution de la ligne 4.
2	Affecter à p la valeur 1	2) Traduire cet algorithme en langage Python puis tester cet algorithme pour $a = 3$ et $n = 2$.
3	Pour i allant de 1 à n	3) Que calcule cet algorithme ?
4	Affecter à p la valeur $p \times a$	
5	Fin Pour	
6	Afficher p	

Ex 3 - Placement sur un compte épargne :

Fatou place 50 000 F CFA sur un compte épargne à 2% par an. Chaque année, les intérêts s'ajoutent au capital. Elle souhaite savoir quel sera son capital au bout de 7 ans.

1	Lire a et c	1) Compléter cet algorithme pour répondre au problème.
2	Pour i allant de 1 à	2) Tester cet algorithme pour $a = 7$ et $c = 50\,000$ à l'aide d'un tableau où on donnera les valeurs de a et c après chaque exécution de la ligne 3.
3	Affecter à c la valeur $c \times \dots$	3) Programmer cet algorithme sur Python et contrôler la réponse au problème posé.
4	Fin Pour	
5	Afficher c	

Ex 4 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 3 :

- 1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse choisir le taux de rémunération du compte épargne.
- 2) Tester l'algorithme pour un taux de 2,5% puis de 3%.

Ex 5 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 4 : Fatou compte aussi placer 2 000 F CFA de plus par an.

- 1) Modifier l'algorithme (langage naturel et python) pour savoir quel sera alors son capital au bout de 7 ans.
- 2) Tester l'algorithme pour un taux de 2% ; 2,5% et 3%.

Ex 6 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 5 :

- 1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse entrer la somme qu'il veut placer en plus par an.
- 2) Tester l'algorithme pour un taux de 2,5% et un versement supplémentaire de 5 000 F CFA par an.

Exercices : Boucles « Tant que »

Ex 1 - Afficher les nombres entiers pairs inférieurs à un nombre choisi :

1	Lire n
2	Affecter à p la valeur 0
3	Tant que $n \geq p$
4	Afficher p
5	Affecter à p la valeur $p + 2$
6	Fin Tant que

- 1) Tester cet algorithme pour $n = 7$ à l'aide d'un tableau dans lequel on précisera les valeurs des différentes variables au niveau du « Fin Tant que ».
- 2) Programmer cet algorithme sur python et tester pour $n=18$, $n=47$...

Ex 2 - Placement sur un compte épargne :

Fatou place 50 000 F CFA sur un compte épargne à 5% par an. Chaque année, les intérêts s'ajoutent au capital. Elle souhaite savoir au bout de combien d'année son capital dépassera 100 000 F CFA et quel sera alors son capital.

1	Affecter à a la valeur 0
2	Lire c
3	Tant que $c < 100\ 000$
4	Affecter à c la valeur $c \times \dots$
5	Affecter à a la valeur $a + 1$
	Fin Tant que
	Afficher a
	Afficher c

- 1) Compléter cet algorithme pour répondre au problème.
- 2) Tester cet algorithme pour $c = 50\ 000$ à l'aide d'un tableau où on donnera les valeurs de a et c après chaque exécution de la ligne 3.
- 3) Programmer cet algorithme sur python et contrôler la réponse au problème posé.

Ex 3 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 2 :

- 1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse choisir le taux de rémunération du compte épargne.
- 2) Tester l'algorithme pour un taux de 2% puis de 3%.

Ex 4 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 3 : Fatou compte aussi placer 2 000 F CFA de plus par an.

- 1) Modifier l'algorithme (langage naturel et python) pour savoir au bout de combien d'année son capital dépassera 100 000 F CFA et quel sera alors son capital.
- 2) Tester l'algorithme pour un taux de 2% ; 3% ; 5%.

Ex 5 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 4 :

- 1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse entrer la somme qu'il veut placer en plus par an.
- 2) Tester l'algorithme pour un taux de 2,5% et un versement supplémentaire de 5 000 F CFA par an.

Exercices : Fonctions

Exercice 1 :

Soit le programme python suivant :

```
def carre(n):  
    return n*n
```

- Quel est le nom de la fonction définie dans le programme ci-dessus
- Cette fonction a-t-elle des paramètres ? Si oui combien ? et lesquels ?
- Que fait cette fonction ?
- Taper le programme sur Python et donner le résultat obtenu pour $n=8$ puis pour $n=6,2$

Exercice 2 :

Soient les deux programmes python suivants :

```
def premiers_carres(k):  
    for i in range(k):  
        print carre(i)
```

```
def premiers_carres(k):  
    i = 0  
    while i < k:  
        print carre(i)  
        i = i + 1
```

- Dans chacun de ces deux programmes est définie une fonction . Que fait la fonction du 1^{er} programme ? la fonction du 2^e programme ? Que remarquez-vous ?
- Taper chacun de ces deux programmes sur Python et donner le résultat obtenu pour chacun d'eux pour $n=5$ puis pour $n=12$
- Que faut-il rajouter au programme si on veut qu'il demande d'abord un entier k à l'utilisateur puis qu'il affiche par la suite les k premiers carrés

Exercice 3 :

Ecrire une fonction qui prend en paramètres deux nombres et renvoie le plus grand des deux nombres.

Exercice 4 :

Ecrire une fonction qui prend en paramètre un entier strictement positif k et qui renvoie la somme des k premiers carrés non nuls.

A. Les booléens :

Ce qu'il faut savoir :

- **George Boole**, un mathématicien et philosophe britannique du XIX^e siècle, créa de 1844 à 1854 une algèbre binaire, dite booléenne, n'acceptant que deux valeurs numériques : 0 et 1.

Cette algèbre aura de nombreuses applications en téléphonie et en informatique.



George Boole

- Un booléen est un type de variable (« **bool** ») qui n'a que deux états qui sont en général **True** (représenté par un bit 1) ou **False** (représenté par un bit 0).
- Sur Python, il ne faut pas confondre le symbole « **=** » qui sert à affecter une valeur à une variable avec le symbole « **==** » qui sert à tester une égalité.

Exemple : Si on tape sur la console de Python :

```
>>>a=2
>>>b=3
>>>a=b
```

Ici on affecte la valeur 2 à *a* , puis 3 à *b* et enfin on affecte la valeur de *b* à *a* .
(à la fin *a* vaut 3)

Si on tape sur la console de Python :

```
>>>a=2
>>>b=3
>>>a==b
```

Ici on affecte la valeur 2 à *a* , puis 3 à *b* et enfin on teste si la valeur de *a* est égale à celle de *b* .
(ici *a == b* est un booléen qui a la valeur False)

- Sur Python, « **==** » est un opérateur de comparaison.
Les autres opérateurs de comparaison sont : « **!=** » (différent de), « **<** » (strictement inférieur), « **>** » (strictement supérieur), « **<=** » (inférieur ou égal) et « **>=** » (supérieur ou égal).

Activité 1 : Pour chaque cas, détermine ce que va afficher Python (après avoir répondu, corrige toi en tapant les instruction sur la console de Python).

1. >>>a=3
>>>b=5
>>>a>b

False

2. >>>a=3
>>>b=5
>>>a!=b

True

3. >>>type(5)==str

False

4. >>>5=="5"
False

5. >>>5==5.0
True

6. >>>type(5)==type(5.0)
False

7. >>>var1="vive la nsi"
>>>var2="nsi"
>>>var2 in var1
True

8. >>>var1="vive la nsi"
>>>var2="nsi"
>>>var1 in var2
False

9. `>>>a="j'aime"
>>>b=" la nsi"
>>>c=a+b
>>>c=="j'aime la nsi"`
True

10. `>>>a=-5.0
>>>b=4.2
>>>a*b>4*a-1`
False

Ce qu'il faut savoir : Les opérateurs booléens.

- L'opérateur « **not** » (qui signifie « non ») est le booléen **opposé**.

Voici **la table de vérité** qui définit cet opérateur pour un booléen a :

a	False	True
not a	True	False

a	0	1
not a	1	0

Remarque : quand le booléen est défini avec les chiffres 0 et 1, alors **not a = 1 – a**.

- L'opérateur « **and** » (qui signifie « et ») teste si deux booléens sont vrais **simultanément**.

Voici la table de vérité qui définit cet opérateur pour deux booléens a et b :

a and b	False	True
False	False	False
True	False	True

a and b	0	1
0	0	0
1	0	1

Remarque : quand les booléens sont définis avec les chiffres 0 et 1, alors **a and b = a x b**.

- L'opérateur « **or** » (qui signifie « ou ») teste si **au moins un des deux** booléens est vrai.

Voici la table de vérité qui définit cet opérateur pour deux booléens a et b :

a or b	False	True
False	False	True
True	True	True

a or b	0	1
0	0	1
1	1	1

Remarque : quand les booléens sont définis avec les chiffres 0 et 1, alors **a or b = a + b - a x b**.

Activité 2 : Pour chaque cas, détermine ce que va afficher Python (après avoir répondu, corrige toi en tapant les instruction sur la console de Python).

1. `>>>(8>4 or 1>2)`
True

2. `>>>a=True
>>>b=False
>>>not(a or b) or (a and not(b))`
True

3. `>>>a=3
>>>b=-7
>>>a**3>50 and b**2<50`
False

4. `>>>a=3
>>>b=-7
>>>(a**3>50 and b**2<50)
or (a**2<10 and b**2>10)`
True

5. `>>>mot="science"
>>>mot[0]=="s"`
True

6. `>>>mot="science"
>>>mot[0]=="s" and mot[5]=="n"`
False

7. `>>>mot="science"
>>>not(mot[2:5]=="ien")`
False

8. `>>>var1=" vive la nsi"
>>>var2="vive le prof"
>>>"vive" in (var1 and var2)`
True

B. Détermination des tables de certaines fonctions booléennes :

1. **a.** Recopie et complète le tableau ci-dessous.

a	b	a and b	b and a	a or b	b or a
False	False	False	False	False	False
False	True	False	False	True	True
True	False	False	False	True	True
True	True	True	True	True	True

b. Que peux-tu en déduire pour les booléens (a and b) , puis (b and a) ? Et pour (a or b), puis (b or a) ?
(a and b)=(b and a) puis (a or b)=(b or a)

2. Sur le même modèle, construire un tableau pour chacune des questions et en tirer des conclusions :

a. (a and True).

b. (a or False) .

a	a and True	a or False
False	False	False
True	True	True

c. (a and (b or c)) puis ((a and b) or (a and c)).

a	b	c	a and (b or c)	(a and b) or (a and c)
False	False	False	False	False
False	False	True	False	False
False	True	False	False	False
False	True	True	False	False
True	False	False	False	False
True	False	True	True	True
True	True	False	True	True
True	True	True	True	True

d. (a or (b and c)) puis ((a or b) and (a or c)).

a	b	c	a or (b and c)	(a or b) and (a or c)
False	False	False	False	False
False	False	True	False	False
False	True	False	False	False
False	True	True	True	True
True	False	False	True	True
True	False	True	True	True
True	True	False	True	True
True	True	True	True	True

e. (a and not(a)).

f. (a or not(a)).

g. not(not(a)).

a	a and not(a)	a or not(a)	not(not(a))
False	False	True	False
True	False	True	True

h. ((a and b) and c) puis (a and (b and c)).

a	b	c	(a and b) and c	a and (b and c)
False	False	False	False	False
False	False	True	False	False
False	True	False	False	False
False	True	True	False	False
True	False	False	False	False
True	False	True	False	False
True	True	False	False	False
True	True	True	True	True

i. ((a or b) or c) puis (a or (b or c)).

a	b	c	(a or b) or c	a or (b or c)
False	False	False	False	False
False	False	True	True	True
False	True	False	True	True
False	True	True	True	True
True	False	False	True	True
True	False	True	True	True
True	True	False	True	True
True	True	True	True	True

j. (a and (a or b)) puis a.

k. (a or a and b) puis a.

a	b	a and (a or b)	a or a and b
False	False	False	False
False	True	False	False
True	False	True	True
True	True	True	True

l. not(a and b) puis (not(a) or not(b)).

a	b	not(a and b)	not(a) or not(b)
False	False	True	True
False	True	True	True
True	False	True	True
True	True	False	False

m. not(a or b) puis (not(a) and not(b))

a	b	not(a or b)	not(a) and not(b)
False	False	True	True
False	True	False	False
True	False	False	False
True	True	False	False

3. Résume les informations découvertes à travers cet exercice (ce sont les théorèmes sur les booléens).

- (a and True) = (a or False) = a
- (a and not(a)) = False (a or not(a)) = True not(not(a)) = a
- (a and (b or c)) = ((a and b) or (a and c))
- (a or (b and c)) = ((a or b) and (a or c))
- ((a and b) and c) = (a and (b and c))
- ((a or b) or c) = (a or (b or c))
- (a and (a or b)) = (a or a and b) = a
- not(a and b) = (not(a) or not(b))
- not(a or b) puis (not(a) and not(b))

C. Portes logiques :

Ce qu'il faut savoir :

● Les circuits d'un ordinateur manipulent uniquement des chiffres binaires (0 et 1) qui, techniquement en interne, sont simplement représentés par des tensions électriques. Ainsi, le chiffre 0 est représenté par une tension basse proche de 0 Volt et le chiffre 1 par une tension haute qui dépend des circuits.

● Les composantes qui vont permettre de faire des opérations sur ces chiffres binaires sont appelées **portes logiques**. Ces portes logiques sont des circuits électroniques composés de **transistors** qui se comportent comme **des interrupteurs qui laissent passer ou non le courant électrique**.

● Les portes logiques élémentaires sont :

- la porte « not »



- la porte « and »



- la porte « or »



Activité 1 :

Les entrées et sorties sont des tensions électriques représentant des booléens 0 (pour False) et 1 (pour True).

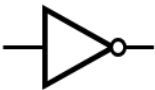
1. On veut créer une porte logique « not » sur le logiciel Logisim et simuler son fonctionnement.


Pour cela observe la vidéo suivante : <https://www.youtube.com/watch?v=gCOsSLtMogo>


Ouvre le logiciel Logisim et réalise ce travail.

2. Sur la même page du logiciel Logisim, crée une porte logique « and », puis une porte logique « or » et simule leur fonctionnement.

3. Complète alors les tables de vérité ci-dessous :

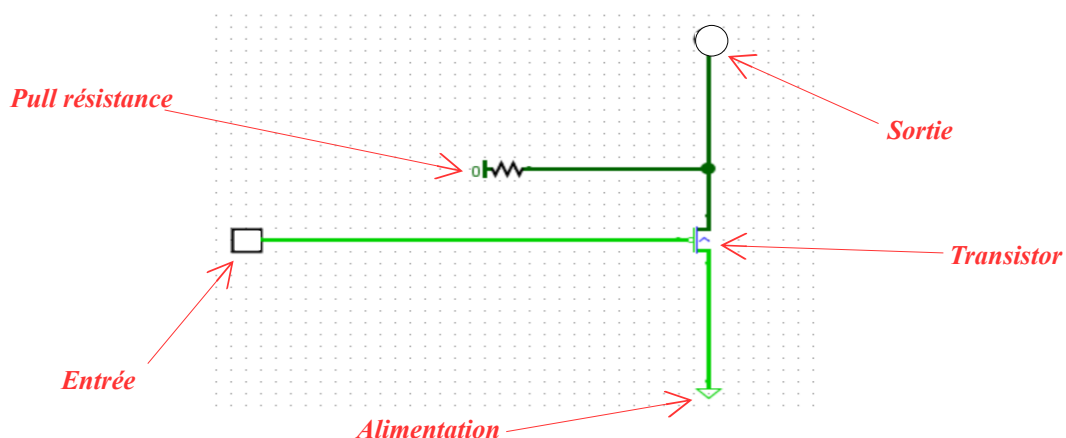
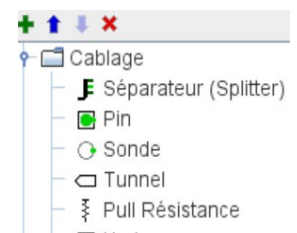
	
not	
E	S
0	1
1	0


		
and		
E1	E2	S
0	0	0
0	1	0
1	0	0
1	1	1

		
or		
E1	E2	S
0	0	0
0	1	1
1	0	1
1	1	1

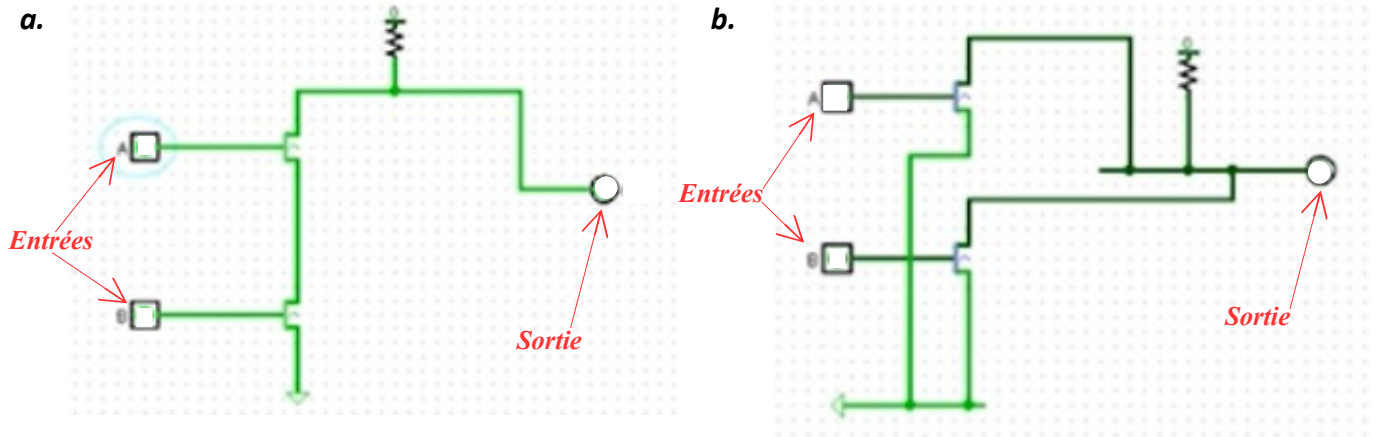
Activité 2 : On veut maintenant construire ces 3 portes logiques à l'aide de transistors avec le logiciel Logisim.

1. a. Ouvre un nouveau fichier Logisim et réalise le circuit ci-dessous en utilisant les composants accessibles dans le dossier « câblages » visible ci-contre et accessible en haut à gauche du fichier Logisim.



- b. A l'aide de cet outil  Simule le fonctionnement de ce circuit en mettant l'entrée à 0, puis 1.
Quelle porte logique reconnaît-on ?
- c. En cliquant sur « **Fichier** » puis « **Sauvegarder comme...** », enregistre ton travail dans le dossier **Partage (R) : NSI : Gauthier** en lui donnant le nom : « **NOM.fichier1** ».

2. Réalise de même chacun des deux circuits ci-dessous et précise à quelles porte logique correspond chacun d'entre eux.



c. En cliquant sur « **Fichier** » puis « **Sauvegarder comme...** », enregistre tes travaux dans le dossier **Partage (R) : NSI : Gauthier** en leur donnant le nom : « **NOM.fichier2** » et « **NOM.fichier3** ».

Activité 3 : On veut additionner 2 nombres binaires de 1 bit. Pour cela on va construire 2 fonctions logiques S (pour Somme) et R (pour Retenue) qui prennent en entrées E1 et E2 les 2 nombres binaires.

1. Recopie et complète la table de vérité de chacune de ces fonctions logiques :

SOMME		
E1	E2	S
0	0	0
0	1	1
1	0	1
1	1	0

RETENUE		
E1	E2	R
0	0	0
0	1	0
1	0	0
1	1	1

2. a. A quel opérateur correspond la fonction logique qui donne la retenue ? **And**

b. Pour la fonction logique qui donne la somme, dans quels cas S est à 1 (True) ?

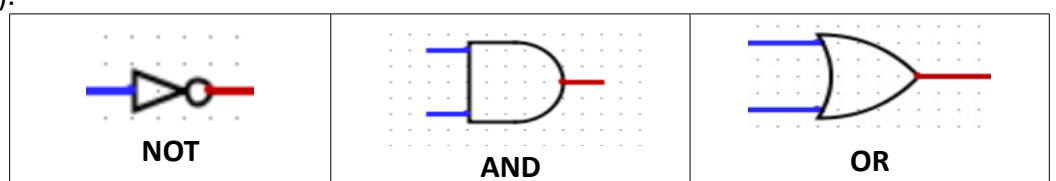
Quand on a : (faux pour E1 et vrai pour E2) ou (vrai pour E1 et faux pour E2)

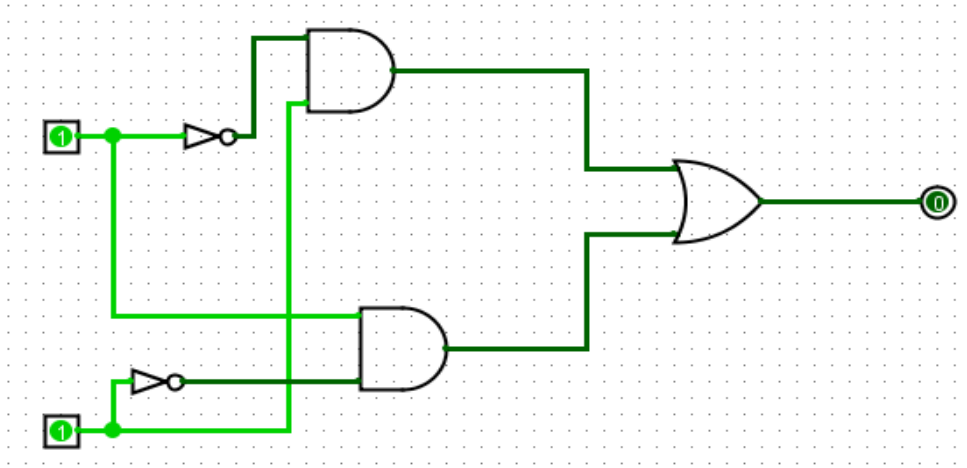
c. Partant de E1=True (1) et E2=True (1), écris une expression logique avec E1 et E2 traduisant ta réponse du b.

(not(E1) and E2) ou (E1 and not(E2))

d. A l'aide de ce qui vient d'être fait, complète chaque « ? » par une des portes logiques utilisables (données ci-dessous) ce circuit dans lequel on a 2 entrées (les nombres binaires E1 et E2) et la sortie S (la somme).

Portes logiques utilisables
(une ou plusieurs fois) :



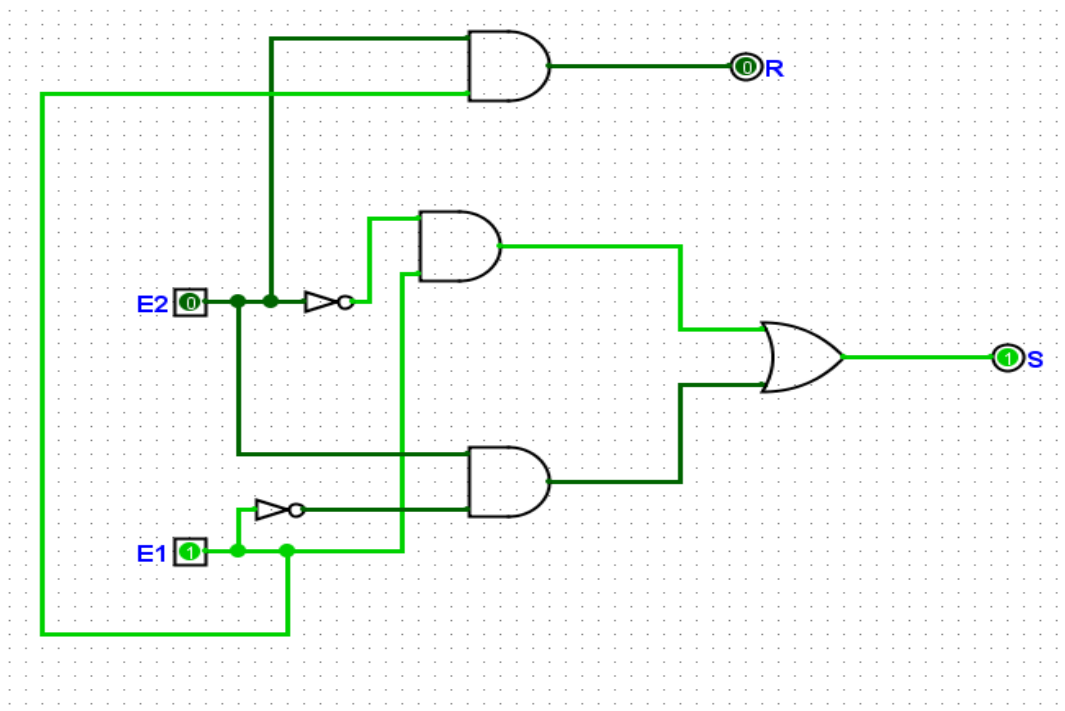


- e. Ouvre un nouveau fichier Logisim, puis à l'aide des questions précédentes, reproduis le circuit de la question c. et complète le afin qu'il permette d'obtenir les résultats des fonctions logiques S (Somme), mais aussi R (Retenue). Ce circuit se nomme **un demi-additionneur sur 1 bit.**

Par un clic sur le circuit nommé « main » en haut à gauche de la page, fait apparaître le nom « **main** » du circuit, que tu vas remplacer par « **demi_add** ».

Veille à faire apparaître E1, E1 et S pour les entrées et la sortie en cliquant dessus et en inscrivant ces noms dans le label.

VHDL	Verilog
Représentation	Est
Emplacement du label	Nord
Police du Label	SansSerif Gras 16
Label Visible	Oui
Nom du circuit	main
Label partagé	



3. On veut maintenant additionner 2 nombres binaires de 1 bit en tenant compte de la retenue entrante. Pour cela on va construire 2 fonctions logiques S (pour Somme) et R_s (pour Retenue sortante) qui prennent en entrées $E1$ et $E2$ les 2 nombres binaires, et R_e (pour retenue entrante).

Recopie et complète la table de vérité de chacune de ces fonctions logiques :

SOMME			
R_e	$E1$	$E2$	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

RETENUE sortante			
R_e	$E1$	$E2$	R_s
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

4. a. On va commencer par additionner (avec un demi-additionneur) $E1$ et $E2$ et on obtient une première somme intermédiaire S_i et une première retenue intermédiaire R_i . Il faut évidemment encore additionner (avec un autre demi-additionneur) S_i et R_e afin d'obtenir une seconde somme intermédiaire s_i et une seconde retenue intermédiaire r_i .

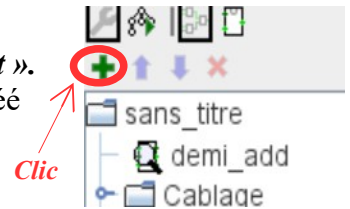
Recopie et complète le tableau ci-dessous :

R_e	$E1$	$E2$	S_i	R_i	s_i	r_i
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	0	0	1	0
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	0

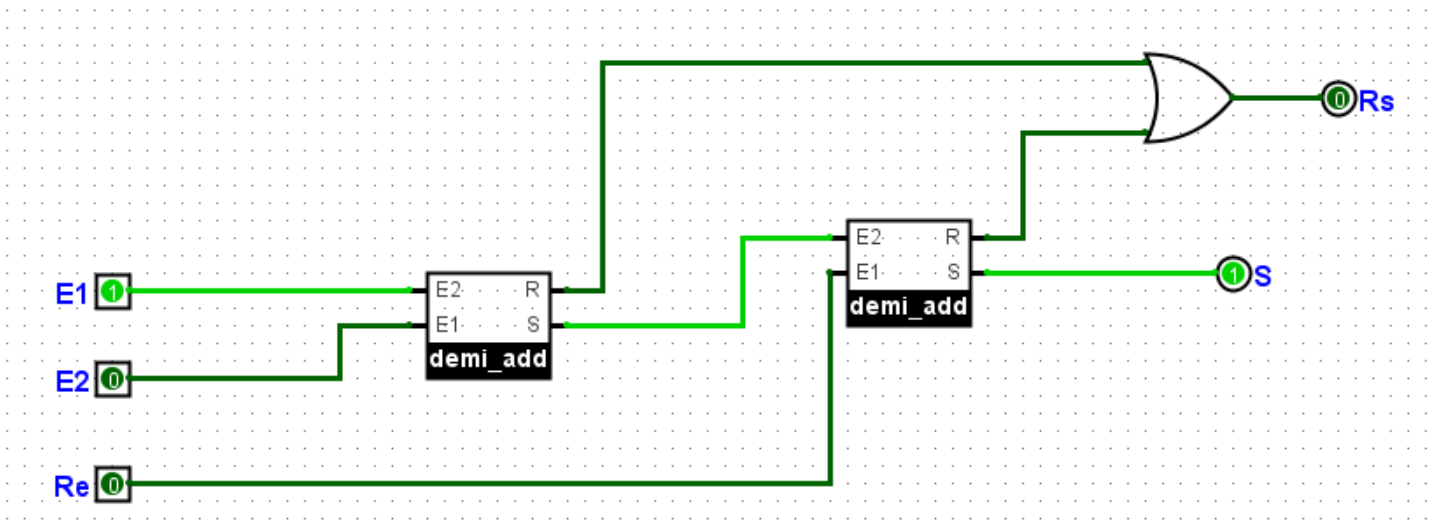
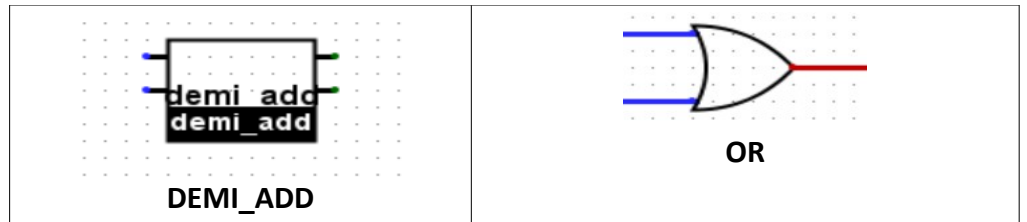
b. En observant attentivement les deux derniers tableaux, compare la seconde somme intermédiaire s_i et la SOMME cherchée. Puis trouve un lien logique entre R_i , r_i et la RETENUE SORTANTE cherchée.

$s_i = \text{SOMME}$ et $R_i \text{ or } r_i = \text{RETENUE SORTANTE}$

c. Sur le fichier Logisim de la question 3, par un clic sur le « + » vert en haut à gauche de la page, tu vas ajouter une circuit que tu nommeras « *add_complet* ». Puis complète le circuit ci-dessous où l'on a utilisé le circuit « *demi_add* » créé à la question 2.

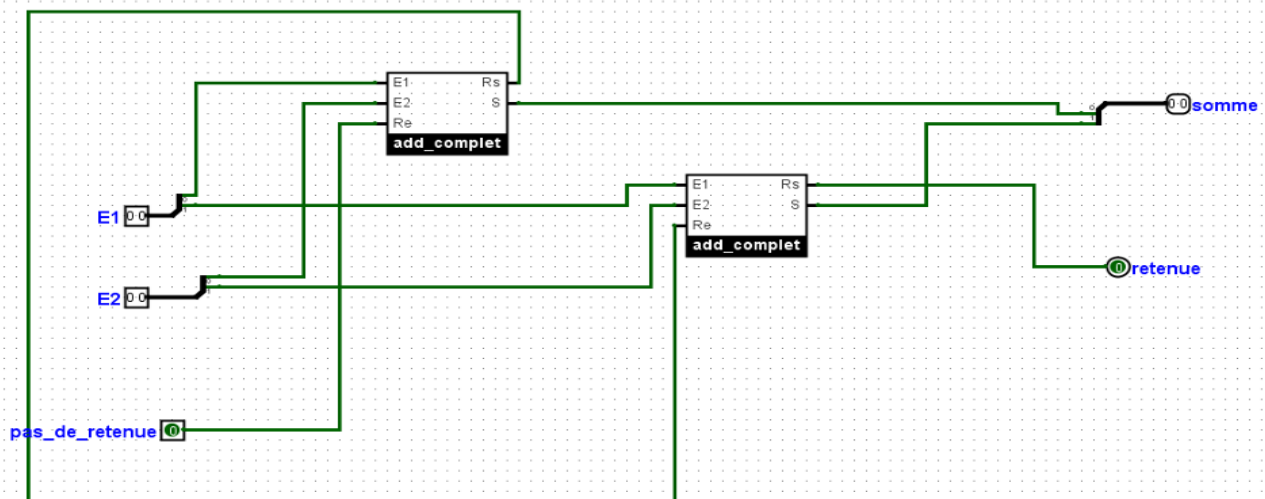


éléments utilisables
(une ou plusieurs fois) :



d. A l'aide des questions précédentes, reproduis sur Logisim ce circuit dans le circuit « *add_complet* ». Ce circuit se nomme un additionneur complet sur 1 bit.

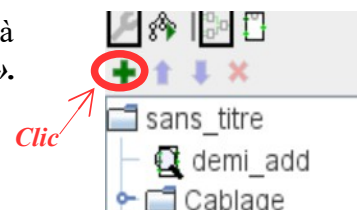
5. On s'intéresse maintenant au circuit cidessous, où E1, E2 et S représentent des nombres binaires de 2 bits:



a. Explique en quelques phrases le fonctionnement de ce circuit.

Avec un premier « *add_complet* » on additionne les premiers bits de E1 et E2 en prenant 0 comme retenue entrante (car pas de retenue au départ du calcul). La somme obtenue est le premier bit de la somme en sortie et la retenue sortante devient la retenue entrante du second « *add_complet* ». Dans le second « *add_complet* » on additionne les seconds bits de E1 et E2 en prenant comme retenue entrante, la retenue sortante précédente. La retenue obtenue est la retenue en sortie et la somme obtenue est le second bit de la somme en sortie.

b. Sur le fichier Logisim de la question 3, par un clic sur le « + » vert en haut à gauche de la page, tu vas ajouter une circuit que tu nommeras « *add_2bits* ». Puis reproduis le circuit précédent.



c. Teste ce circuit et reporte les résultats de tests dans le tableau ci-dessous que tu complèteras.

E1	E2	somme	retenue
(0,0)	(0,0)	(0,0)	0
(0,0)	(1,0)	(1,0)	0
(0,0)	(0,1)	(0,1)	0
(0,0)	(1,1)	(1,1)	0
(0,1)	(0,0)	(0,1)	0
(0,1)	(1,0)	(1,1)	0
(0,1)	(0,1)	(1,0)	0
(0,1)	(1,1)	(0,0)	1
(1,0)	(0,0)	(1,0)	0
(1,0)	(1,0)	(0,0)	1
(1,0)	(0,1)	(1,1)	0
(1,0)	(1,1)	(0,1)	1
(1,1)	(0,0)	(1,1)	0
(1,1)	(1,0)	(0,1)	1
(1,1)	(0,1)	(0,0)	1
(1,1)	(1,1)	(1,0)	1

CORRECTION :

A. Algorithmique - Premiers algorithmes:

Définition : Un **algorithme** est une liste d'instructions à exécuter pas à pas pour accomplir une tâche précise.

Remarques

- Un algorithme doit être lisible de tous ; Son intérêt est d'être codé dans un langage informatique pour qu'une machine (ordinateur, calculatrice, etc.) puisse l'exécuter rapidement et efficacement.
- Les trois phases d'un algorithme sont :
 - l'entrée des données ;
 - le traitement des données ;
 - la sortie des résultats.

Ex 1 - On considère l'algorithme suivant :

Choisir un nombre de départ	3					
Lui ajouter 1	4					
Multiplier le résultat par 2	8					
Soustraire 3 au résultat et afficher le résultat final	5					

- 1) Recopier le tableau ci-dessus, puis :
 - a) Appliquer successivement cet algorithme à -4 , 0 , $1/3$
 - b) Déterminer les nombres à choisir au départ pour afficher à la fin 0 puis 7
- 2) Écrire un nouvel algorithme qui permette, en partant du résultat final de l'algorithme ci-dessus, de retrouver le nombre de départ.
- 3) Traduire chacun de ces deux algorithmes par une formule en fonction du nombre de départ x .
Quelle est la nature des deux fonctions trouvées ?

Ex 2 - On considère l'algorithme suivant :

Choisir un nombre
Calculer le carré de ce nombre
Multiplier par 10
Ajouter 25
Afficher le résultat

- 1) En faisant un tableau comme celui de l'exercice 1), appliquer cet algorithme à 2 , puis $\sqrt{2}$
- 2) Traduire cet algorithme par une formule en fonction de x .
- 3) Pénélope affirme que si le nombre choisi au départ est un entier alors le résultat est impair.
A-t-elle raison ? Justifier.
- 4) Ulysse affirme que le résultat est toujours positif quelque soit le nombre choisi au départ.
A-t-il raison ? Justifier.

A. Le langage Python:

Les algorithmes sont traduits en programmes informatiques à partir des différents langages de programmation qui existent. Dans ce chapitre, nous utiliserons **Python** comme langage de programmation.

1. Éléments de base

a. Les types

Le type d'une variable définit l'ensemble des valeurs qui peuvent lui être affectées. Les types de base sont **int**, **bool**, **float** qui sont des types numériques et **str**.

Ce qu'il faut savoir :

- ✓ Le type **int**, abréviation de integer, est utilisé pour représenter les **nombres entiers**.
- ✓ Le type **bool** permet de représenter les valeurs booléennes True (vrai) et False (faux).
- ✓ Le type **float** est utilisé pour représenter les **nombres réels**. Attention, en Python, **le point** remplace la **virgule** utilisée en mathématiques (donc 5,8 est représenté par 5.8 en Python).
- ✓ Le type **str**, abréviation de string, est utilisé pour représenter des chaînes de caractères. On utilise des guillemets, des apostrophes ou str comme `a='bonjour'`, `a="bonjour"` ou `a=str(bonjour)`.

a. Les opérations sur les types numériques

Les opérations de base

+	addition
-	soustraction
*	multiplication
/	division
**	puissance
//	division entière
%	reste de la division

```
15 // 6
```

Réponse : 2

```
15 % 6
```

Réponse : 3

L'écriture scientifique

Exemple :

```
2.75e3
```

Réponse : 2750.0

- **Les opérateurs mathématiques classiques de comparaisons** =, ≠, <, ≤, >, ≥ s'écrivent en Python respectivement ==, !=, <, <=, >, >=.
- **Les opérations logiques :**
 - **a and b** prend la valeur **True** si **a** et **b** sont **True** et sinon prend la valeur **False** ;
 - **a or b** prend la valeur **False** si **a** et **b** sont **False** et sinon prend la valeur **True** ;
 - **not a** prend la valeur **True** si **a** est **False** et prend la valeur **False** si **a** est **True**.

Remarques :

- Sur Python, pour utiliser des fonctions mathématiques comme la racine carrée, il faut l'importer à partir du module math avec l'instruction suivante `from math import sqrt` (sqrt étant la racine carrée en python)
- Pour importer toutes les fonctions du module math, la syntaxe est `from math import *` (à mettre au début du programme : 1^{ère} ligne en général)

Ce qu'il faut savoir : En général, un **algorithme** est construit en trois étapes :

- **Entrée** : On saisit les données.
- **Initialisation** : Le programme attribue des valeurs initiales à des variables. Ces valeurs peuvent changer au cours du programme.
- **Traitement des données** : Les instructions du programme effectuent des opérations à partir des données saisies dans le but de résoudre un problème.
- **Sortie** : Les résultats sont affichés

2. Affectation de variables

Définition :

Une **variable** est composée d'un nom (ou identificateur), d'une adresse en mémoire où est enregistrée une valeur ou un ensemble de valeurs) et d'un type qui définit ses propriétés.

Une **variable** permet de stocker une valeur que l'on compte réutiliser plus tard.

Les instructions de base sur les variables sont les suivantes :

- la **saisie** : on demande à l'utilisateur de donner une valeur à une variable ;
- l'**affectation** : est une instruction qui commande à la machine de créer une variable en lui précisant son nom et sa valeur) ;
- l'**affichage** : on affiche la valeur de la variable.

○

Remarques

Les notations ci-dessous sont équivalentes :

- Affecter à b la valeur de a
- b prend la valeur de a

En Python

- L'**affectation** se fait avec le signe `=` (**exemple :** `u=5` : on affecte ici la valeur 5 à la variable u)
- La **saisie** se fait avec la commande `input ()` : elle permet d'inviter l'utilisateur à saisir une valeur à l'exécution du programme.

Exemple :

```
nom = input("Quel est  
votre nom ?")
```

Remarque :

input est une fonction qui renvoie toujours une chaîne de caractères. Pour changer le type de la variable, on utilise : int (pour les entiers) ou float (pour les réels). Voir l'exemple ci-dessous :

```
n = float(input("Entrer un nombre"))  
print("Le carré de", n, "est", n * n)
```

- L'affichage se fait avec la commande **print ()**

```
print("Bonjour !")
```

```
print(2)
```

```
a = -3  
print("Le carré de", a, "est", a * a)
```

Remarque :

La commande **print ()** permet d'afficher textuellement la partie entre les guillemets " " et d'afficher la valeur des variables en dehors des guillemets. En effet dans le dernier exemple ci-dessus, il sera affiché : Le carré de -3 est 9.

Ex 1 - On donne l'algorithme suivant :

```
Lire n  
q prend la valeur de  $(n + 2) \times (n + 2)$   
q prend la valeur de  $q - (n + 4)$   
q prend la valeur de  $q / (n + 3)$   
Afficher q
```

1) Tester cet algorithme pour $n = 4$, puis pour $n = 7$.

Si $n=4$, $q=(n+2) \times (n+2)=6 \times 6=36$, et $q=q-(n+4)=36-(4+4)=28$, puis $q=\frac{q}{n+3}=\frac{28}{7}=4$. Affiche 4.

Si $n=7$, $q=(n+2) \times (n+2)=9 \times 9=81$, et $q=q-(n+4)=81-(7+4)=70$, puis $q=\frac{q}{n+3}=\frac{70}{10}=7$. Affiche 7.

- 2) Écrire le programme python correspondant à cet algorithme (en considérant n comme un réel) et tester sur Python pour $n = 6,2$ puis pour $n = -9,8$.

```
n=float(input("choisir un nombre"))
q=(n+2)*(n+2)
q=q-(n+4)
q=q/(n+3)
print("le résultat est : ",q)
```

- 3) Émettre une conjecture sur le résultat fourni par cet algorithme puis démontrer cette conjecture.

Il semble que le résultat obtenu est toujours égal au nombre de départ choisi.

Démonstration de cette conjecture :

Si on choisit un nombre n quelconque, alors on calcule à la première étape $(n+2)(n+2)$.

Puis à la seconde étape $(n+2)(n+2)-(n+4)=n^2+4n+4-n-4=n^2+3n$, soit en factorisant $n(n+3)$.

A la dernière étape on obtient finalement $\frac{n(n+3)}{n+3}=n$. Cela prouve qu'on retrouve toujours n, le nombre du départ.

- 4) Un élève a saisi $n = -3$. Que se passe-t-il ? Pourquoi ?

Il s'affiche le message d'erreur ci-contre : `ZeroDivisionError: float division by zero`

En effet, à la dernière étape on divise par $n+3$. Si $n=-3$, on divise par $-3+3=0$, ce qui est impossible.

Ex 2 - Un commerçant accorde une remise sur des articles. On souhaite connaître le montant de la remise en euros. Voici un algorithme écrit en langage naturel donnant la solution au problème :

Entrée

Saisir le prix de départ A

Saisir le pourcentage de remise P

Traitement des données

Affecter au montant de la remise R la valeur $\frac{A \times P}{100}$

.....

Sortie

Afficher R

.....

- 1) Calculer la valeur de la variable R lorsque $A = 56$ et $P = 30$.

Donner une interprétation concrète du résultat précédent.

$R = 56 \times \frac{30}{100} = 16,80$. Cela signifie que si le prix de départ est de 56 euros et que le pourcentage de remise est 30%, alors le montant de la remise est de 16,80 euros.

2) Compléter l'algorithme pour qu'il affiche également le prix à payer B.

Entrée

Saisir le prix de départ A

Saisir le pourcentage de remise P

Traitement des données

Affecter au montant de la remise R la valeur $\frac{A \times P}{100}$

Affecter au prix à payer B la valeur A-R

Sortie

Afficher R

Afficher B

3) Traduire l'algorithme en langage Python puis calculer la valeur des variables R et B lorsque A = 159 et P = 24. Donner une interprétation concrète des résultats précédents.

```
A=float(input("Saisir le prix de départ :"))
P=float(input("Saisir le pourcentage de remise :"))
R=A*P/100
B=A-R
print("Le montant de la réduction est : ",R," euros")
print("Le prix à payer est : ",B," euros")
```

Lorsque A=159 et P=24, on obtient R=38,16 et B=120,84. Cela veut dire que si le prix de départ est 159 euros et le pourcentage de remise est 24%, alors le montant de la remise est de 38,16 euros et le prix à payer est de 120,84 euros.

4) Même question avec A = 13 et P = 45.

Lorsque A=13 et P=45, on obtient R=5,85 et B=7,15. Cela veut dire que si le prix de départ est 13 euros et le pourcentage de remise est 45%, alors le montant de la remise est de 5,85 euros et le prix à payer est de 7,15 euros.

Exercice 5 :

1) Rédiger en langage naturel puis en langage Python un algorithme permettant de calculer le pourcentage de réduction d'un article connaissant le prix de départ et le prix à payer.

Entrée

Saisir le prix de départ A

Saisir le prix à payer B

Traitement des données

Affecter au montant de la remise R = A-B

Affecter au pourcentage de remise $P = \frac{R \times 100}{A}$

Sortie

Afficher P

```

A=float(input("Saisir le prix de départ :"))
B=float(input("Saisir le prix à payer :"))
R=A-B
P=(R*100)/A
print("Le pourcentage de réduction est : ",P," %")

```

- 2) Calculer avec le pourcentage de réduction d'un article dont le prix de départ est de 75 euros et le prix final est de 49,5 euros.

On obtient un pourcentage de réduction de $P=34\%$.

Instruction conditionnelle

Définition : La résolution de certains problèmes nécessite la mise en place d'un **test** pour savoir si l'on doit effectuer une tâche.

Si la condition est remplie alors la tâche sera effectuée, sinon on effectue (éventuellement) une autre tâche. Dans un algorithme, on code la structure du « Si... AlorsSinon... FinSi » sous la forme suivante :

```

Si      condition Alors
        Tâche 1
        Tâche 2
Sinon
        Tâche 1bis
        Tâche 2bis
FinSi

```

Remarques :

- A chaque « Si » doit correspondre un « Fin Si ».
- En revanche le « Sinon » n'est pas toujours obligatoire. S'il n'est pas présent, aucune tâche ne sera effectuée si la condition n'est pas remplie.
- Il est important de respecter les espaces laissés au début de chaque ligne car ils permettent une meilleure lisibilité de l'algorithme.

En Python

- La commande **if** permet de tester le contenu d'une variable et exécute une série d'instructions si les conditions sont remplies.
- L'indentation, décalage vers la droite du début de ligne, est un élément de syntaxe important en Python . Elle délimite des blocs de code et elle aide à la lisibilité en permettant d'identifier facilement ces blocs. La ligne précédant l'indentation se termine par le signe deux-points .

La structure la plus simple est :

```
if condition:
    instructions
```

Le mot condition désigne une expression et le mot instructions désigne une instruction ou un bloc d'instructions écrites sur plusieurs lignes. Voici un exemple :

```
if n % 2 == 0:
    n = n // 2
```

La structure if-else présente une alternative :

```
if condition:
    instructions1
else:
    instructions2
```

Par exemple :

```
if n % 2 == 0:
    n = n // 2
else:
    n = 3 * n + 1
```

La structure if-elif-else présente plusieurs alternatives :

```
if condition1:
    instructions1
elif condition2:
    instructions2
elif condition3:
    instructions3
...
else:
    instructions
```

Par exemple :

```
if n % 4 == 0:
    n = n // 4
elif n % 4 == 1:
    n = (3 * n + 1) // 4
elif n % 4 == 2:
    n = n // 2
else:
    n = (3 * n + 1) // 2
```

Remarques :

- Les lignes qui commencent par les mots if et elif se terminent par le signe deux-points ;
- Le mot else est suivi immédiatement par le signe deux-points ;
- C'est l'indentation qui permet de délimiter les blocs d'instruction à exécuter si la condition est vérifiée.

Exercices d'application : test if

Exercice 1

Afficher « Entrez les dimensions du triangle en commençant par la plus grande »
Lire a , b et c
Si $a^2 = b^2 + c^2$
 Afficher « Le triangle est rectangle. »
Sinon
 Afficher « Le triangle n'est pas rectangle. »
FinSi

- 1) Quelle condition faut-il écrire après le « Si » ? **Voir sur l'algorithme.**
- 2) Traduire cet algorithme en langage python puis tester pour $a=5$, $b=4$ et $c=3$
- 3) Modifier cet algorithme de façon à afficher en plus « Données incorrectes » si a n'est pas le plus grand des trois nombres rentrés.
- 4) Traduire ce nouvel algorithme en langage python puis tester pour $a=4$, $b=5$ et $c=3$. Puis pour $a=7$, $b=4$, $c=6$ et enfin tester pour $a=8,5$; $b=3,6$ et $c=7,7$.

- 2) *$a=\text{float}(\text{input}(\text{"choisir la plus grande dimension du triangle :"}))$
 $b=\text{float}(\text{input}(\text{"choisir la seconde dimension du triangle :"}))$
 $c=\text{float}(\text{input}(\text{"choisir la dernière dimension du triangle :"}))$
 $\text{if } a**2==b**2+c**2:$
 $\text{print}(\text{"Le triangle est rectangle."})$
 else:
 $\text{print}(\text{"Le triangle n'est pas rectangle."})$*

Pour $a=5$, $b=4$ et $c=3$, on obtient : « Le triangle est rectangle ».

- 3) **Afficher « Entrez les dimensions du triangle en commençant par la plus grande »**
Lire a , b et c
Si $a < b$ ou $a < c$
 Afficher « Données incorrectes. »
Sinon si $a^2 = b^2 + c^2$
 Afficher « Le triangle est rectangle. »
Sinon
 Afficher « Le triangle n'est pas rectangle. »
FinSi


```

4)  a=float(input("choisir la plus grande dimension du triangle :"))
    b=float(input("choisir la seconde dimension du triangle :"))
    c=float(input("choisir la dernière dimension du triangle :"))
    if a<b or a<c:
        print("données incorrectes")
    elif a**2==b**2+c**2:
        print("Le triangle est rectangle.")
    else:
        print("Le triangle n'est pas rectangle.")

```

pour a= 4, b=5 et c=3, on obtient : « Données incorrectes ».

Pour a=7, b=4, c=6, on obtient : « Le triangle n'est pas rectangle ».

Pour a= 8,5 ; b=3,6 et c=7,7, on obtient : « Le triangle est rectangle ».

Exercice 2 :

Un particulier souhaite louer une voiture.

- L'agence de location A demande 100 € au départ et 0,20 € par kilomètre parcouru ;

- L'agence de location B demande 150 € au départ et 0,15 € par kilomètre parcouru.

1) Compléter cet algorithme pour aider le consommateur à trouver la formule la plus avantageuse

2) Traduire cet algo en langage python puis donner la meilleure formule quand on veut parcourir 1000km, puis pour 525,75 km et enfin pour 1500km.

```

1)  1) Afficher « Entrer le nombre de kilomètres. »
    Lire n
    Affecter à a la valeur 100 + 0,20 x n
    Affecter à b la valeur 150 + 0,15 x n
    Si a<b
        Afficher « La formule A est plus avantageuse. »
    Sinon
        Si a=b
            Afficher « Les formules A et B sont équivalentes. »
        Sinon
            Afficher « La formule B est plus avantageuse. »
    FinSi
FinSi

```

```

2)  n=float(input("Saisir le nombre de kilomètres :"))
    a=100+0.20*n
    b=150+0.15*n
    if a<b:
        print("La formule A est plus avantageuse.")
    elif a==b:
        print("Les deux formules sont équivalentes")
    else:
        print("La formule B est la plus avantageuse")

```

Pour 1000km, on obtient : « Les deux formules sont équivalentes ».

Pour 525,75km, on obtient : « La formule A est plus avantageuse ».

Pour 1500km, on obtient : « La formule B est plus avantageuse ».

Exercice 3

Un magasin propose de tirer des photos sur papier au tarif de 0,16 € la photo pour les 75 premières photos, puis 0,12 € la photo pour les photos suivantes. Écrire un algorithme qui demande à l'utilisateur d'entrer le nombre n de tirages photos commandés et calcule le montant à payer. Traduire l'algorithme en langage python puis tester avec des valeurs inférieures à 75 puis avec des valeurs supérieures à 75

Algorithme en langage naturel :

Afficher « Entrer le nombre de photos. »
Lire n le nombre de photos
Si $n \leq 75$:
 Affecter au prix à payer P la valeur $n \times 0,16$
Sinon :
 Affecter au prix à payer P la valeur $75 \times 0,16 + (n - 75) \times 0,12$
Fin Si
Afficher P .

Algorithme en langage Python :

```
n=int(input("Saisir le nombre de photos :"))
a=100+0.20*n
b=150+0.15*n
if n<=75:
    p=n*0.16
else:
    p=75*0.16+(n-75)*0.12
print("Le prix à payer est : ",p," euros")
```

Si $n=40$, alors le prix est 6,40 euros. Si $n=80$, le prix est 12,6 euros.

Exercice 4

Écrire un algorithme qui demande deux nombres, puis affiche la différence entre le plus grand et le plus petit. Traduire en Python puis tester...

Algorithme en langage naturel :

Afficher « Entrer un nombre. »
Lire x_1 ce nombre
Afficher « Entrer un autre nombre. »
Lire x_2 ce nombre
Si $x_1 > x_2$:
 Affecter à la différence D la valeur $x_1 - x_2$
Sinon :
 Affecter à la différence D la valeur $x_2 - x_1$
Fin Si
Afficher D .

Algorithme en langage Python :

```
x1=float(input("Saisir un nombre."))
x2=float(input("Saisir un autre nombre."))
if x1>x2:
    D=x1-x2
else:
    D=x2-x1
print("La différence entre les 2 nombres est : ",D)
```

Boucles « Pour » et « Tant que »

Définitions : Les boucles permettent de répéter plusieurs fois une partie de l'algorithme.
Il existe deux sortes de boucles :

1. Boucle « Pour » :

Lorsque **le nombre de répétitions n est connu à l'avance**, on utilise un compteur initialisé à 1 et qui augmente automatiquement de 1 à chaque répétition jusqu'à n . On parle de boucle itérative.

Pour Variable allant de Valeur début à Valeur fin
 Tâche 1
 Tâche 2
Fin Pour

En Python :

La syntaxe est la suivante :

```
for i in range(n):
    instructions
```

La variable i est appelé un compteur. Elle prend successivement les valeurs 0, 1, ..., $n-1$.

Exemple 1: Ecrivez un programme qui copierait 20 fois « je ne dois pas mâcher du chewing-gum »

```
for i in range(1,21):
    print("je ne dois pas mâcher du chewing-gum")
print("fini")
```

ou

```
for i in range(0,20):
    print("je ne dois pas mâcher du chewing-gum")
print("fini")
```

2. Boucle « Tant que » :

Lorsque **le nombre de répétitions n n'est pas connu à l'avance**, il peut dépendre d'une condition. Le traitement est répété tant que la condition est vraie. Lorsqu'elle est fausse, on sort de la boucle. On parle de boucle conditionnelle.

Tant que *Condition est vraie*

Tâche 1

Tâche 2

Fin Tant que

En Python :

La syntaxe est la suivante :

```
while condition:
    instructions
```

Exemple 1 :

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
print('Fini !')
```

Que fait ce programme ?

Ce programme écrit les valeurs de i tant que $i \leq 5$.

Dès que $i = 5$ le programme écrit « c'est fini ! ».

Donc il sera écrit :

1 2 3 4 5 c'est fini !

Exemple 2 :

Les deux programmes suivant sont équivalents :

```
a=0
while a<10:
    print("boucle Tant que")
    a=a+1
print("Fin du programme")
```

```
for loop in range(10):
    print("boucle for")
print("Fin du programme")
```

Que font ces programmes ?

Ces 2 programmes écrivent 0 1 2 3 4 5 6 7 8 9 Fin du programme.

Les fonctions en Python

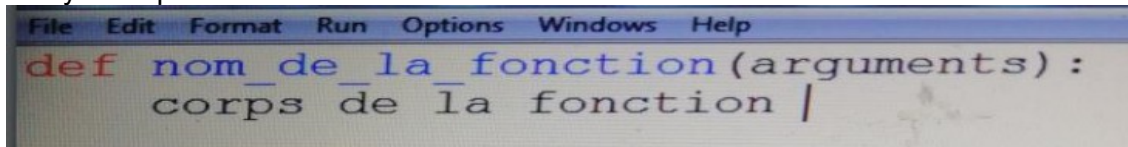
Ce qu'il faut savoir : Dans un programme, il est possible d'écrire des petits programmes ou des sous-programmes intermédiaires appelés *fonctions*.

Définition : Une fonction est un programme qui porte *un nom* et utilise zéro, une ou plusieurs variables appelés *paramètres ou arguments*.

Remarques :

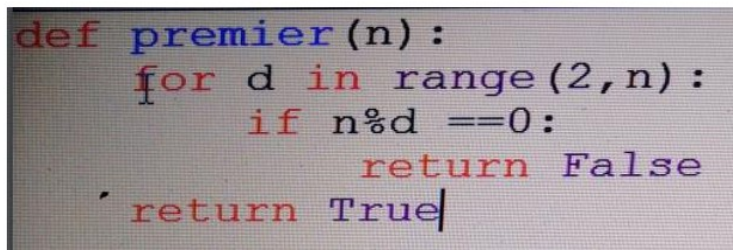
- Le nom d'une fonction ne peut pas avoir d'espace. On met donc des tirets « _ ».
- Si la fonction n'utilise aucun paramètre, on la note **def nom_de_la_fonction()**
- Les arguments (paramètres) sont séparés par des virgules. La première ligne, ligne de définition, se termine par le signe deux-points. Le corps de la fonction est un bloc indenté par rapport à la ligne de définition.
- L'instruction *return* permet de renvoyer une valeur de type entier, décimal (dit flottant) ou chaîne de caractères qui peut être réutilisée dans un autre programme ou une autre fonction. Elle interrompt le programme dès qu'elle s'est exécutée.

La syntaxe pour définir une fonction est :



```
File Edit Format Run Options Windows Help
def nom_de_la_fonction(arguments):
    corps de la fonction |
```

Exemple de fonction :



```
def premier(n):
    for d in range(2, n):
        if n%d == 0:
            return False
    return True|
```

Que fait cette fonction ?

Cette fonction renvoie « False » dès qu'un entier d compris entre 2 et $n-1$ est un diviseur de n .

Si aucun entier d compris entre 2 et $n-1$ n'est diviseur de n , la fonction renvoie « True ».

Autrement dit, si n est un nombre premier, alors la fonction renvoie « True », sinon elle renvoie « False ».

Exercices : Boucles « Pour »

Ex 1 - La somme des n premiers nombres entiers :

1	Lire n
2	Affecter à s la valeur 0
3	Pour i allant de 1 à n
4	Affecter à s la valeur $s + i$
5	Fin Pour
6	Afficher s

1) Tester cet algorithme pour $n = 3$ à l'aide d'un tableau où on donnera les valeurs de i et s après chaque exécution de la ligne 4.
2) Traduire cet algorithme en langage Python puis tester cet algorithme pour $n = 5$ puis pour $n=100$.

1) si $n=3$:

i	1	2	3
s	1	3	6

2)

```
n=int(input("choisir un nombre entier."))
s=0
for i in range(1,n+1):
    s=s+i
print("s=",s)
```

Si $n=5$, on obtient $s=15$. Si $n=100$, on obtient $s=5050$.

Ex 2 - On considère l'algorithme ci dessous :

1	Lire a et n
2	Affecter à p la valeur 1
3	Pour i allant de 1 à n
4	Affecter à p la valeur $p \times a$
5	Fin Pour
6	Afficher p

1) Tester cet algorithme pour $a = 2$ et $n = 3$ à l'aide d'un tableau où on donnera les valeurs de i et p après chaque exécution de la ligne 4.
2) Traduire cet algorithme en langage Python puis tester cet algorithme pour $a = 3$ et $n = 2$.
3) Que calcule cet algorithme ?

1) Si $a=2$ et $n=3$:

i	1	2	3
p	2	4	8

2)

```
n=int(input("choisir un nombre entier."))
a=int(input("choisir un autre entier."))
p=1
for i in range(1,n+1):
    p=p*a
print("p=",p)
```

Si $n=2$ et $a=3$, on obtient $p=9$.

3) **Cet algorithme calcule a^n .**

Ex 3 - Placement sur un compte épargne :

Fatou place 50 000 F CFA sur un compte épargne à 2% par an. Chaque année, les intérêts s'ajoutent au capital. Elle souhaite savoir quel sera son capital au bout de 7 ans.

1	Lire <i>a</i> et <i>c</i>
2	Pour <i>i</i> allant de 1 à <i>a</i>
3	Affecter à <i>c</i> la valeur $c \times 1,02$
4	Fin Pour
5	Afficher <i>c</i>

- 1) Compléter cet algorithme pour répondre au problème.
- 2) Tester cet algorithme pour $a = 7$ et $c = 50\,000$ à l'aide d'un tableau où on donnera les valeurs de *i* et *c* après chaque exécution de la ligne 3.
- 3) Programmer cet algorithme sur Python et contrôler la réponse au problème posé.

1) **Fait sur l'algorithme.**

2) **Si $a=7$ et $c=50\,000$:**

<i>i</i>	1	2	3	4	5	6	7
<i>c</i>	51000	52020	53060.4	54121.608	55204.04	56308.12	57434.28

3) ***a=int(input("choisir un nombre d'années."))***
c=float(input("Choisir le capital initial."))
for i in range(1,a+1):
c=c*1.02
print("la somme au bout de ",a,"années est :",c,"cfa.")

Ex 4 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 3 :

- 1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse choisir le taux de rémunération du compte épargne.

Langage naturel :

Lire *a* , *c* et *t*
Pour *i* allant de 1 à *a*
Affecter à *c* la valeur $c \times (1+t/100)$
Fin Pour
Afficher *c*

Python :

a=int(input("choisir un nombre d'années."))
c=float(input("Choisir le capital initial."))
t=float(input("Choisir le taux d'intérêts"))
for i in range(1,a+1):
c=c*(1+t/100)
print("la somme au bout de ",a,"années est :",c,"cfa.")

- 2) Tester l'algorithme pour un taux de 2,5% puis de 3%.
Pour un taux de 2,5% on obtient environ 59434,29cfa.
Pour un taux de 3% on obtient environ 61493,69cfa.

Ex 5 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 4 : Fatou compte aussi placer 2 000 F CFA de plus par an.

1) Modifier l'algorithme (langage naturel et python) pour savoir quel sera alors son capital au bout de 7 ans.

Langage naturel :

Lire a , c et t
Pour i allant de 1 à a
 Affecter à c la valeur $c \times (1+t/100)+2000$
Fin Pour
Afficher c

Python :

```
a=int(input("choisir un nombre d'années."))  
c=float(input("Choisir le capital initial."))  
t=float(input("Choisir le taux d'intérêts"))  
for i in range(1,a+1):  
    c=c*(1+t/100)+2000  
print("la somme au bout de ",a,"années est :",c,"cfa.")
```

2) Tester l'algorithme pour un taux de 2% ; 2,5% et 3%.

Pour un taux de 2,5% on obtient environ 74529,15cfa.

Pour un taux de 3% on obtient environ 76818,62cfa.

Ex 6 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 5 :

1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse entrer la somme qu'il veut placer en plus par an.

Langage naturel :

Lire a , c , t et s
Pour i allant de 1 à a
 Affecter à c la valeur $c \times (1+t/100)+s$
Fin Pour
Afficher c

Python :

```
a=int(input("choisir un nombre d'années."))  
c=float(input("Choisir le capital initial."))  
t=float(input("Choisir le taux d'intérêts"))  
s=float(input("Choisir la somme à ajouter chaque année"))  
for i in range(1,a+1):  
    c=c*(1+t/100)+s  
print("la somme au bout de ",a,"années est :",c,"cfa.")
```

2) Tester l'algorithme pour un taux de 2,5% et un versement supplémentaire de 5 000 F CFA par an.

Pour un taux de 2,5% et 5000cfa d'ajouté chaque année, on obtient environ 97171.44cfa.

Exercices : Boucles « Tant que »

Ex 1 - Afficher les nombres entiers pairs inférieurs à un nombre choisi :

1	Lire n
2	Affecter à p la valeur 0
3	Tant que $n \geq p$
4	Afficher p
5	Affecter à p la valeur $p + 2$
6	Fin Tant que

1) Tester cet algorithme pour $n = 7$ à l'aide d'un tableau dans lequel on précisera les valeurs des différentes variables au niveau du « Fin Tant que ».
2) Programmer cet algorithme sur python et tester pour $n=18$, $n=47$...

1) Si $n=7$:

p	0	2	4	6	8
$n \geq p$	VRAI	VRAI	VRAI	VRAI	FAUX

Le programme va afficher 0 2 4 6.

```
2)  n=int(input("choisir un entier. "))
    p=0
    while n>=p:
        print(p)
        p=p+2
```

Pour $n=18$, le programme affiche 0 2 4 16 18.

Pour $n=47$, le programme affiche 0 2 4 44 46.

Ex 2 - Placement sur un compte épargne :

Fatou place 50 000 F CFA sur un compte épargne à 5% par an. Chaque année, les intérêts s'ajoutent au capital. Elle souhaite savoir au bout de combien d'année son capital dépassera 100 000 F CFA et quel sera alors son capital.

1	Affecter à a la valeur 0
2	Lire c
3	Tant que $c < 100\ 000$
4	Affecter à c la valeur $c \times 1.05$
5	Affecter à a la valeur $a + 1$
	Fin Tant que
	Afficher a
	Afficher c

1) Compléter cet algorithme pour répondre au problème.
2) Tester cet algorithme pour $c = 50\ 000$ à l'aide d'un tableau où on donnera les valeurs de a et c après chaque exécution de la ligne 3.
3) Programmer cet algorithme sur python et contrôler la réponse au problème posé.

1) Voir sur l'algorithme.

2)

c	50000	52500	55125	103946,41
a	0	1	2	15
$c < 100000$	VRAI	VRAI	VRAI	FAUX

```
3)  a=0
    c=float(input("Choisir le capital initial. "))
    while c<100000:
        c=c*1.05
        a=a+1
    print("la somme au bout de ",a,"années est :",c,"cfa.")
```

Ex 3 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 2 :

- 1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse choisir le taux de rémunération du compte épargne.

Langage naturel :

Affecter à a la valeur 0
Lire c et t
tant que c<100000
 Affecter à c la valeur $c \times (1+t/100)$
 a=a+1
Fin Pour
Afficher c

Python :

```
a=0
c=float(input("Choisir le capital initial."))
t=float(input("Choisir le taux d'intérêts"))
while c<100000:
    c=c*(1+t/100)
    a=a+1
print("la somme au bout de ",a,"années est :",c,"cfa.")
```

- 2) Tester l'algorithme pour un taux de 2% puis de 3%.

Pour un taux de 2%, le capital dépasse 100000cfa au bout de 24 années.

Pour un taux de 3%, le capital dépasse 100000cfa au bout de 36 années.

Ex 4 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 3 : Fatou compte aussi placer 2 000 F CFA de plus par an.

- 1) Modifier l'algorithme (langage naturel et python) pour savoir au bout de combien d'année son capital dépassera 100 000 F CFA et quel sera alors son capital.

Langage naturel :

Affecter à a la valeur 0
Lire c et t
tant que c<100000
 Affecter à c la valeur $c \times (1+t/100)+2000$
 a=a+1
Fin Pour
Afficher c

Python :

```
a=0
c=float(input("Choisir le capital initial."))
t=float(input("Choisir le taux d'intérêts"))
while c<100000:
    c=c*(1+t/100)+2000
    a=a+1
print("la somme au bout de ",a,"années est :",c,"cfa.")
```

- 2) Tester l'algorithme pour un taux de 2% ; 3% ; 5%.

Pour un taux de 2%, le capital dépasse 100000cfa au bout de 15 années.

Pour un taux de 3%, le capital dépasse 100000cfa au bout de 13 années.

Pour un taux de 5%, le capital dépasse 100000cfa au bout de 10 années.

Ex 5 - Placement sur un compte épargne (suite) :

Reprendre l'exercice 4 :

1) Modifier l'algorithme (langage naturel et python) pour que l'utilisateur puisse entrer la somme qu'il veut placer en plus par an.

Langage naturel :

Affecter à a la valeur 0

Lire c , t et s

tant que c<100000

Affecter à c la valeur $c \times (1+t/100)+s$

a=a+1

Fin Pour

Afficher c

Python :

a=0

c=float(input("Choisir le capital initial."))

t=float(input("Choisir le taux d'intérêts"))

s=float(input("Choisir la somme à ajouter"))

while c<100000:

c=c*(1+t/100)+s

a=a+1

print("la somme au bout de ",a,"années est :",c,"cfa.")

2) Tester l'algorithme pour un taux de 2,5% et un versement supplémentaire de 5 000 F CFA par an.

Pour un taux de 2,5% et 5000cfa d'ajouté chaque année, on dépasse 100000cfa au bout de 8 ans.

Exercices : Fonctions

Exercice 1 :

```
def carre(n):  
    return n*n
```

Soit le programme python suivant :

- Quel est le nom de la fonction définie dans le programme ci-dessus ? **carre.**
- Cette fonction a-t-elle des paramètres ? Si oui combien ? et lesquels ? **Oui, un : n.**
- Que fait cette fonction ? **Elle calcule le carré du nombre n.**
- Taper le programme sur Python et donner le résultat obtenu pour n=8 puis pour n=6,2
Il faut taper `carre(8)` et `carre(6.2)` dans la console et on obtient 64 et 38,44.

Exercice 2 :

Soient les deux programmes python suivants :

```
def premiers_carres(k):  
    for i in range(k):  
        print carre(i)
```

```
def premiers_carres(k):  
    i = 0  
    while i < k:  
        print carre(i)  
        i = i + 1
```

- 1) Dans chacun de ces deux programmes est définie une fonction . Que fait la fonction du 1^{er} programme ? la fonction du 2^e programme ? Que remarquez-vous ? ***Ces 2 fonctions ont le même rôle : elles calculent les carrés des nombres de 0 à k-1 à l'aide de la fonction de l'exercice 1.***
- 2) Taper chacun de ces deux programmes sur Python et donner le résultat obtenu pour chacun d'eux pour n=5 puis pour n=12
Il faut taper premiers_carres(5), premiers_carres(12) dans la console (ne pas oublier de taper aussi la fonction carre!).
Dans les 2 cas on a pour premiers_carres(5) : 0 1 4 9 16
Puis pour premiers_carres(12) : 0 1 4 9 16 25 36 49 64 81 100 121
- 3) Que faut-il rajouter au programme si on veut qu'il demande d'abord un entier k à l'utilisateur puis qu'il affiche par la suite les k premiers carrés.

```
def carre(n):  
    return(n*n)
```

```
def premiers_carres(k):  
    for i in range(k):  
        print(carre(i))
```

```
n=int(input("choisir un nombre entier"))  
premiers_carres(n)
```

Exercice 3 :

Écrire une fonction qui prend en paramètres deux nombres et renvoie le plus grand des deux nombres.

```
def plus_grand(x,y):  
    if x>y:  
        return(x)  
    else:  
        return(y)
```

Puis taper plus_grand(2,3) dans la console pour tester par exemple.

Exercice 4 :

Écrire une fonction qui prend en paramètre un entier strictement positif k et qui renvoie la somme des k premiers carrés non nuls.

```
def somme_carres(n):  
    for i in range(1,n+1):  
        print(i**2)
```

Puis taper somme_carres(5) dans la console pour tester par exemple.