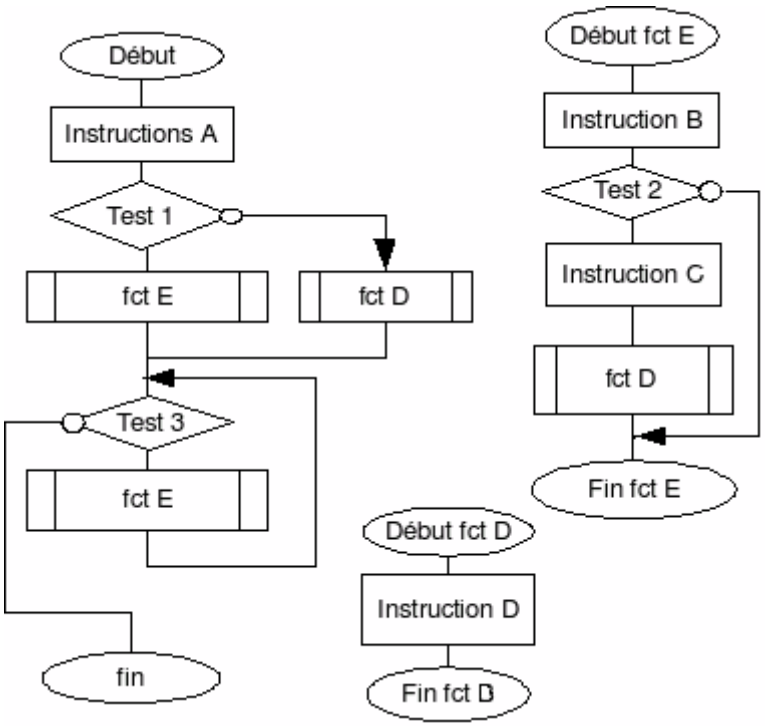
CORRIGE INSTRUCTIONS COMPOSEES EN PYTHON (I) : BOUCLE FOR, TESTS, BOUCLE WHILE.

Me signaler toute erreur éventuelle !



- *Logiciels et sites internet* : Editeur et console Python (Thonny, VS Code etc.) ; www.franceioi.org.
- *Pré-requis pour prendre un bon départ :*

	☹	😐	☺	☺☺
Variables : initialisation, affectation, auto-affectation, incrémentation, etc.				
Fonctions d'entrée sortie.				
Expressions booléennes (expressions vraies ou fausses).				
Schémas ensemblistes.				

Ce cours Python fonctionne en pédagogie inversée. Ce livret est donc un post-cours complémentaire aux exos de France IOI, et doit être fait juste après les chapitres correspondants sur France IOI :

Exercices France IOI	*Chapitres de ce livret*
Niveau 1 Chapitres 2-6	Chapitre I : Boucles For.
Niveau 1 Chapitres 5-6-7.	Chapitre II : Tests conditionnels.
Niveau 1 Chapitre 8.	Chapitre III : Boucles While.
	Chapitre IV : Compléments Boucles.

NOM et prénom : …… *Première spécialité NSI*

I. REPETITIONS DONT LE NOMBRE EST CONNU A L'AVANCE.

L'une des raisons d'être de l'Informatique est de faire faire aux machines ce qui est emmerdant pour les humains. En particulier les choses répétitives et ennuyantes.

Il existe 2 types de répétitions :

- les répétitions dont le nombre est connu à l'avance.
- les répétitions dont le nombre n'est pas connu à l'avance, car dépendantes d'une certaine condition.

Le type de répétition le plus simple est celui dont *le nombre est connu à l'avance*.

En Python (mais aussi dans bien d'autres langages : langage C, *PHP, Go etc.*), l'instruction qui permet d'exécuter en boucle un nombre précis de fois une ou plusieurs actions s'appelle la boucle *For*.

A. Syntaxe de la boucle déterminée For :

Modèle	*Explications*
for Variable de boucle **in** Séquence **:** Corps de la boucle Suite du programme	• **for** Variable de boucle **in** Séquence **:** Entête de la boucle constituée des mots réservés *for* (sans majuscule) et *in*. Se termine obligatoirement par « : ». • Séquence : C'est une suite itérative d'éléments, semblable à une liste. Ex : (0 , 1 , 2) ; ['MDR', 'GG','bouh'] ; [k, 'same as', 1 > 0] etc. (Nous reviendrons au cours 3 sur ce type Séquence.) **Le nb de tours de boucle (nb de répétitions) est donné par le *nombre d'éléments* dans Séquence.** • Variable de boucle (parfois appelé indice de boucle) : Nom au choix du programmeur. Ex : loop, k , nb_tours etc. **A chaque tour de boucle, Variable de boucle reçoit l'élément de Séquence correspondant au numéro de tour.** • Corps de la boucle ou Bloc d'instructions : Ce seront la ou les instructions qui seront exécutées et répétées. Ce bloc est obligatoirement *indenté* à droite. • Suite du programme : Après la dernière instruction du dernier tour de boucle, le programme passe à la suite, c-à-d à la **première instruction non indentée** qui n'est donc plus dans le corps de la boucle.
Traduction en Français	
Pour le nb de fois où Variable est affectée aux éléments de Séquence, exécuter : Corps de la boucle. Suite du programme	

Exemples d'entêtes de boucle For et du nombre de tours de boucle associé			
for element in ['nb', 3 , k] : *3 éléments dans ['nb', 3 , k]* *→ 3 tours.*	for lettre in 'abracadabra' : *11 lettres dans abracadabra* *→ 11 tours*	for loop in range(5) : *→ 5 tours*	for nb in range(p) : *→ p tours*

B. Fonction range( ) :

On remarque dans les 2 derniers exemples page précédente la présence de la fonction « range( ) ».

La fonction range( ) permet de construire des suites (arithmétiques) d'entiers selon les schémas suivants :

Syntaxe	*Entrée(s)*	*Sortie*	*Exemples à vérifier à la console*
• range(f)	• f entier positif.	• similaire à ⟦ 0 ; f ⟦, la liste des entiers **de 0 inclus jusqu'à l'entier f exclu.** • similaire à ⟦ 0 ; f – 1 ⟧.	• range(4) → similaire à [0 , 1 , 2 , 3]. • range(3) → similaire à [0 , 1 , 2]. • range(k) → similaire à [0 , 1 , etc. , k – 1]. • range(1) → similaire à [0]. • range(0) → similaire à [ ] liste vide.
• range(d , f)	• d et f entiers positifs ou négatifs. • d ≤ f.	• similaire à ⟦ d ; f ⟦, la liste des entiers **de d inclus jusqu'à f exclu.** • similaire à ⟦ d ; f – 1 ⟧.	• range(2 , 5) → similaire à [2 , 3 , 4]. • range(1 , 4) → similaire à [1 , 2 , 3]. • range(-1 , 2) → similaire à [-1 , 0 , 1]. • range(-1 , k) → similaire à [-1 , etc , k – 1] • range(4 , 1) → similaire à [ ] liste vide.
• range(d , f , p)	• d , f et p entiers positifs ou négatifs.	• similaire à la liste de tous les entiers partant **de d inclus,** itéré (augmenté) à chaque fois du **pas p**, jusqu'**à f exclu.**	• range(2, 9, 3) → similaire à [2 , 5 , 8]. • range(2, 8, 3) → similaire à [2 , 5]. • range(5, 0, -2) → similaire à [5 , 3 , 1]. • range(-1, -3, -1) → similaire à [-1 , -2]. • range(1, 3, -1) → similaire à [ ] liste vide.

Remarques sur la fonction range( )

1. Lorsque les arguments entre parenthèses ne vérifient pas les conditions d'entrée, la fonction « range( ) » renvoie l'équivalent d'une liste vide.
2. range(f) = range(0 , f) = range(0 , f , 1)

Exemples d'entêtes de boucle For et nombres de tours associés

for k in range(2 , 5) : *range(2 , 5) → [2 , 3 , 4]* *donc 3 tours.*	for j in range(k + 1) : *range(k+1) → ⟦0 ; k⟧* *donc (k + 1) tours.*	for nbs in range(2 , 10, 3) : *range(2, 10, 3) → [2 , 5 , 8] donc 3 tours.*	for p in range(3,-7,-2) : *range(3 , -7 , -2) → [3, 1, -1 , -3 , -5] donc 5 tours.*

C. Exercices classiques utilisant la boucle For :

❶ Sans utiliser l'ordinateur, quels sont les scripts qui affichent 2 , 4 , 6 en colonne ? Vérifier à l'ordinateur.

for k in range(3) : k = k + 2 print (k) 2 3 4 *range(3) → [0 , 1 , 2]* *k = k + 2 donnera successivement 2 , 3 , 4.*	for k in range(1 , 3) : k = k * 2 print (k) 2 4 *range(1 , 3) → [1 , 2]* *k = k * 2 donnera successivement 2 , 4.*	for k in range(6) : if k % 2 == 1 : print (k + 1) 2 4 6 *range(6) → [0,1,2,3,4,5]* *if k%2 == 1 → impairs.* *Puis avec le k + 1…*	for k in range(6) : if k % 2 == 0 : print (k) 0 2 4 *range(6) → [0,1,2,3,4,5]* *if k%2 == 0 sélectionne les pairs.*

for k in range(3) : k = k * 2 print (k + 2) 2 4 6 *range(3) → [0 , 1 , 2]* *k = k * 2 donne 0 , 2 , 4.* *Puis avec le k + 2…*	for k in range(2 , 6 , 2) : print (k) 2 4 *range(2, 6, 2) → [2 , 4]*	for k in range(2 , 7 , 2) : print (k) 2 4 6 *range(2,7,2) → [2 , 4, 6]*	for k in range(2 , 8 , 2) : print (k) 2 4 6 *range(2,8,2) → [2 , 4, 6]*

❷ En utilisant la variable de boucle et la fonction range( ), écrire un script permettant d'afficher la ou les :

Table de multiplication de 3. (boucle simple)

```
for nombre in range(1 , 11) :
        print (nombre * 3)
```

Tables de multiplication de 1 à 10. (2 boucles imbriquées)

```
for tablede in range(1 , 11) :
        print(f"Voici la table de {tablede} : ")
        for nombre in range(1 , 11) :
                print(nombre * tablede)
        print( )
```

❸ Ecrire un programme qui affiche :

La moyenne de plusieurs notes rentrées par un utilisateur. (boucle simple)

```
nb_notes = int(input('Combien de notes ?')
somme = 0
for k in range(nb_notes) :
        note = int(input('Entrez la note : ')
        somme = somme + note
moyenne = somme / nb_notes
print(f"La moyenne des {nb_notes} notes est {moyenne}.")
```

Pour chaque élève de la classe, la moyenne de ses notes. On suppose qu'on a la liste_élèves et pour chaque élève on a la liste_notes. (2 boucles imbriquées)

```
for eleve in liste_eleves :
        somme = 0
        for note in liste_notes :
                somme = somme + note
                moyenne = somme / nb_notes
                print(f"La moyenne de {eleve} est {moyenne}.")
```

D. 4 remarques sur la boucle For :

1. La boucle For est aussi appelée « **boucle bornée** », « **boucle déterminée** » etc. car le nombre de répétitions est connu à l'avance.

2. La boucle bornée For existe dans tous les langages de programmation ! Elle présente une particularité en Python. Ainsi, comparons sur le même exemple les entêtes des boucles For en Python, en C et en Java :

Python	*Langage C*	*Java*
for i in [5 , 6 , 7 , 8] :	for (i = 5 ; i <= 8 ; i = i + 1) {	for (int i = 5 ; i <= 8 ; i = i + 1) {
A chaque tour, l'indice de boucle i est affecté à un élément de la séquence *[5 , 6 , 7 , 8]*.	L'indice de boucle i entier est initialisé à *5* puis incrémenté (autoaffecté et augmenté) de *1* à chaque tour jusqu'à ce que i valle *8*.	
On parle de **boucle séquencée**.	On parle de **boucle incrémentée ou boucle itérative**.	

Citer 2 avantages de la boucle séquencée (on dit parfois boucle listée) par rapport à la boucle incrémentée :

- plus simple à écrire.
- **offre directement le parcourt d'une séquence.**
- **la séquence peut être bien autre chose qu'une suite d'entiers : une suite de mots, de fichiers, etc, et même une chaîne de caractères !**
- etc.

3. **Une variable incrémentée dans le corps d'une boucle peut faire office d'accumulateur**.
   En effet, à chaque tour de boucle, cette variable voit sa valeur augmenter comme une somme partielle.

Application : Pour chacun des 3 scripts suivants, dire quel est l'accumulateur et quelle est sa finalité.

Somme = 0 for k in range(11) : Somme = Somme + k**2	Dépense = 0 for j in range(12) : achat = int (input( )) Dépense = Dépense + achat	Pop = 10 789 for jour in range(28) : décès = int(input( )) Pop = Pop – décès
Accumulateur : Somme. *Calculer la somme des carrés des entiers de 0 à 10.*	*Accumulateur : Dépense.* *Calculer la dépense totale pour 12 achats.*	*Déccumulateur : Pop.* *Calculer la population restante en enlevant les décès chaque jour du mois de février.*

4. **Une variable incrémentée de 1 dans le corps d'une boucle peut faire office de compteur.**

5. L'auto-affectation de variables dans le corps d'une boucle (bornée ou non) est en fait la traduction algorithmique d'un concept mathématique plus général : les suites récurrentes.

E. 4 conseils sur la boucle For :

Conseils	*Exemple à ne pas faire !*	*Commentaire*
• **Eviter d'initialiser une variable interne à une boucle dans la boucle elle-même !** Comportement non attendu assuré !	for k in range(11) : Somme = 0 Somme = Somme + k**2	A chaque tour, Somme est réinitialisée à *0* ! Combien vaudra Somme en sortie ? Somme = *10**2 = 100*.
• **Eviter de faire modifier la séquence de l'entête de boucle dans le corps de la boucle ! (cas 1)** On peut se retrouver avec une boucle for infinie ! Ce qui est paradoxal !	Sequence = [0] for k in Sequence : k = k + 1 Sequence.extend([k])	Combien de tours de boucle sont attendus initialement ? 1 et non 0 ! A chaque tour, Séquence va être étendu de la valeur k ! Que vaut Séquence ? *[0 , 1 , 2 , 3 , etc. .....]*
• **Eviter de faire modifier la séquence de l'entête de boucle dans le corps de la boucle ! (cas 2)** On peut se retrouver avec un comportement non attendu !	n = 2 for k in range(n) : n = n+1	A cause de l'incrémentation de n dans la boucle, range(n) s'agrandit-t-il indéfiniment (donc boucle *infinie* ?). En fait non ! *range(n) n'est évalué que la 1ère fois où il est rencontré et vaudra range(2).* Combien vaut n à la fin ? *4.*
• ***Contrôler attentivement le début et la fin de la boucle.*** Pour éviter comportements inattendus et/ou traitements incomplets !	for k in range(1 , 10) : print (2*k)	Est-ce que ce script affiche bien la table de multiplication de 2 ? *Non car range(10) = ⟦1 ; 9⟧.* *Il manquera 2*0 et 2*10.*

Ai-je tout compris ? Boucles bornées For	☹	😐	☺	☺☺
Définition et syntaxe de la boucle For.				
Fonction range( ).				
Evaluer le nombre de tours de boucles.				
Avantages de la boucle listée (séquencée) sur la boucle incrémentée.				
Accumulateur dans une boucle.				
Appliquer la boucle For dans un cas basique de répétition finie.				
Appliquer la boucle For dans un cas de boucles imbriquées.				
Contrôler attentivement les début et fin de boucle.				
Les pièges de la boucle For.				

II. TESTS CONDITIONNELS ET ALTERNATIVES.

Le déroulement de l'exécution d'un programme (on parle de flux ou flot d'exécution des instructions) n'est quasiment jamais un long fleuve tranquille bien linéaire !

Suivant les données en entrée, il y a plusieurs façons différentes de traiter ces données. Le flux empruntera donc des embranchements selon les choix à faire. Ces choix seront conditionnés soit par des choix externes (choix utilisateurs, choix conditionnés par des capteurs etc.), soit par des résultats internes au programme.

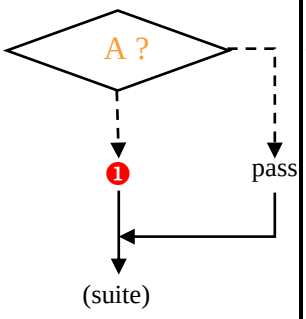
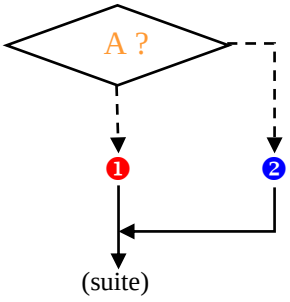
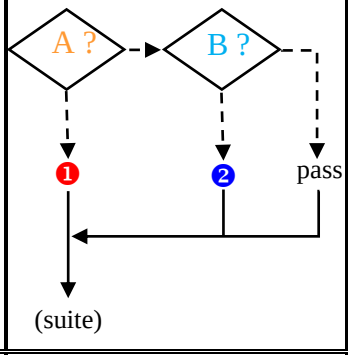
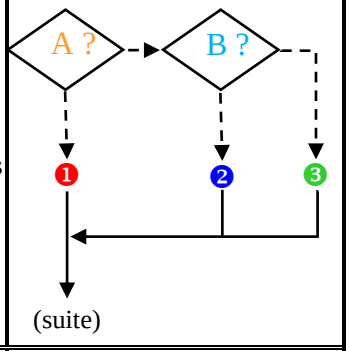
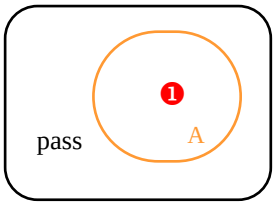
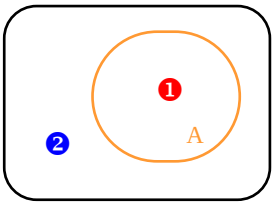
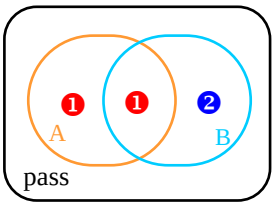
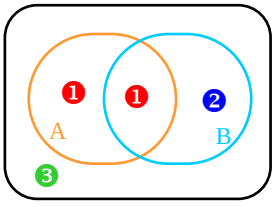
Le traitement conditionnel des données est l'autre structure de base de contrôle du flux (avec les boucles) qui permet d'adapter un algorithme-programme à quasiment n'importe quelle situation.

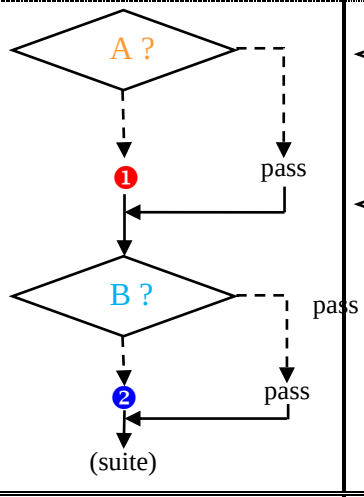
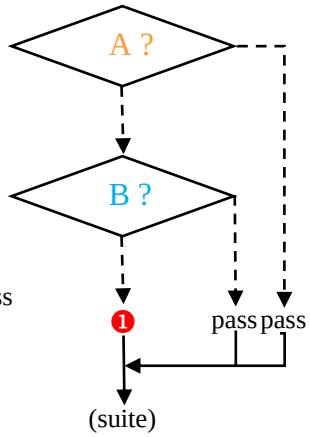
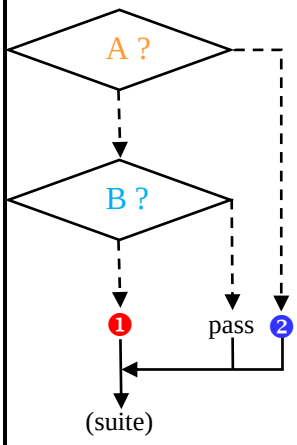
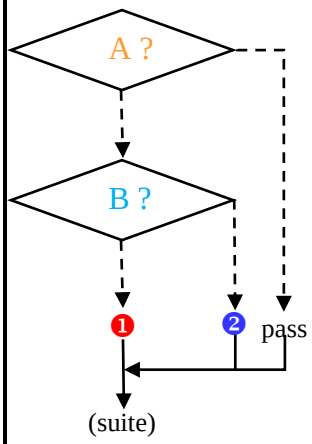
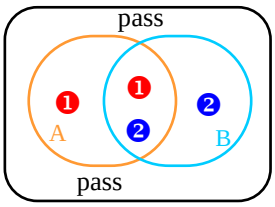
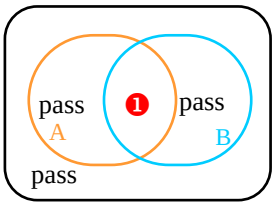
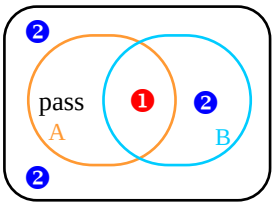
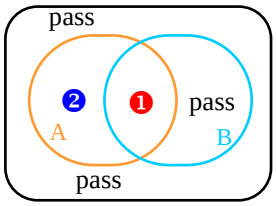
A. Syntaxe des tests conditionnels :

Modèle	*Explications*
if Condition A : Bloc 1 elif Condition B : Bloc 2 else : # Bloc 3 Suite du programme	1. **if** Condition A : obligatoire ! • Entête du test, mot réservé « if » (si). Ne pas oublier les « : ». • Condition A : **expression booléenne** plus ou moins simple. • Si Condition A vraie, exécution du bloc 1 en entier, puis après saut à Suite du programme. 2. elif Conditions B : *facultatif, possibilité de plusieurs elif.* • Suite du test, mot réservé « elif » (« sinon si »). Ne pas oublier les « : ». • Condition B : expression booléenne plus ou moins simple. • Si Condition B vraie (**sachant que A faux**), exécution du bloc 2 en entier, puis après saut à Suite du programme. 3. else : # *facultatif* • Terminaison du test, mot réservé « else » (« autrement »). Ne pas oublier les « : ». • **Aucune condition avant les « : » !** Else traite tous les autres cas possibles des if et elif donc condition interdite avant les « : ». • **Par contre, très bonne habitude d'écrire la condition contraire résultante *en remarque après le « # ».*** • Si A faux puis B faux, exécution du bloc 3 en entier, puis après Suite du programme. 4. Suite du programme : • Quel que soit le bloc conditionnel exécuté, le programme passe à la suite, c-à-d à la 1ère instruction **non indentée** en dehors du test.
Traduction en Français	
Si A vraie, faire Bloc 1, puis Suite du programme. Sinon Si B vraie avec A faux, faire Bloc 2, puis Suite du Programme. Dans tous les autres cas, faire Bloc 3, puis Suite du programme.	

Suivant la présence ou non de elif et/ou else, nous auront affaire à des types différents de tests : conditionnelle simple (if), alternative simple (if else), alternative multiple (if elif else) etc.

B. Différents types et combinaisons de tests : 2 tableaux.

Type de Test	**Conditionnelle simple incomplète.**	**Conditionnelle simple complète ou Alternative simple**	Alternative multiple incomplète.	**Alternative multiple complète.**
En Python	***if***	***if*** ***else***	*if* *elif* ⟺ *else* *if*	***if*** ***elif*** ⟺ ***else :*** ***else*** ***if*** ***else :***
Syntaxe A, B conditions. ❶, ❷, ❸ instructions	if A : ❶ (suite)	if A : ❶ else : # A faux ❷ (suite)	if A : ❶ elif B : ❷ (suite)	if A : ❶ elif B : ❷ else : # A faux et B faux. ❸ (suite)
Algorigramme				
Schémas ensembliste				
Condition pour obtenir	Sorties possibles ❶ | pass A | $\bar{A}$ (not A)	Sorties possibles ❶ | ❷ A | $\bar{A}$ (not A)	Sorties possibles ❶ | ❷ | pass A | $\bar{A}$ et B | ni dans A ni dans B	Sorties possibles ❶ | ❷ | ❸ A | $\bar{A}$ et B | ni dans A ni dans B
Remarques	Sortie incomplète : la sortie pour $\bar{A}$ (non A) n'est pas traitée (pass).	Sorties complètes : partition selon 2 cas possibles : A et $\bar{A}$.	Sortie incomplète : la sortie pour $\overline{A \text{ ou } B}$ n'est pas traitée.	Sorties complètes : partition selon 3 cas possibles : A , $\bar{A}$ et B , $\overline{A \text{ ou } B}$.
Exemple en Python	if k = = 1 : print ('a')	if k = =1 : print ('a') else : print ('b')	if k > 8 : print ('a') elif k < 5 : print ('b')	if k > 8 : print('a') elif k < 5 : print ('b') else : print ('c')
Sortie pour	k = 1 | k = 2 a | pass (rien)	k = 0 | k = *1* *b* | a	k = 10 | k = *6* | k = -7 *a* | pass | b	k = 5 | k = 7 | k = 9 *c* | c | *a*

Type de Test	2 conditionnelles simples incomplètes enchaînées.	2 conditionnelles simples incomplètes imbriquées.	1 conditionnelle imbriquée dans 1 alternative.	1 alternative imbriquée dans 1 conditionnelle.
En Python	*if* *if*	*if* *if*	*if* *if* *else*	*if* *if* *else*
Syntaxe A, B conditions. ❶, ❷ instructions	if A : ❶ if B : ❷ (suite)	if A : if B : ❶ (suite) ⇔ if (A and B) : ❶	if A : if B : ❶ else : # ❷ (suite) ⇔ if A and B : ❶ if not A : ❷ (suite)	if A : if B : ❶ else : # ❷ (suite) ⇔ if A and B : ❶ if A and not B : ❷ (suite)
Algorigramme				
Schémas ensembliste				
Condition pour obtenir	Sorties possibles ❶ | ❷ | ❶ ❷ | pass A et $\bar{B}$ | $\bar{A}$ et B | A et B | $\bar{A}$ et $\bar{B}$	Sorties possibles ❶ | pass A et B | $\bar{A}$ ou $\bar{B}$ non A ou non B	Sorties possibles ❶ | pass | ❷ A et B | A et $\bar{B}$ | $\bar{A}$	Sorties possibles ❶ | ❷ | pass A et B | A et $\bar{B}$ | $\bar{A}$
Remarque	Sortie incomplète : la sortie pour *(ni A et ni B)* n'est pas traitée.	Sortie incomplète : la sortie pour *(pas dans A ou pas dans B)* n'est pas traitée.	Sortie incomplète : la sortie pour *(dans A mais pas dans B)* n'est pas traitée.	Sortie incomplète : la sortie pour *non A* n'est pas traitée.
Exemple en Python	if k > 5 : print ('a') if k % 3 = = 0 : print ('b')	if k > 1 : if k < 4 : print ('a')	if k > 1 : if k < 5 : print ('a') else : print('b')	if k > = 1 : if k < 5 : print ('a') else : print('b')
Sortie pour	k = 3 | k = 6 | k = 7 | k = 2 *b* | *a* *b* | *a* | pass	k = 0 | k = 4 | k = 2 *pass* | *pass* | a	k = 3 | k = 5 | k = 0 a | *pass* | *b*	k = 1 | k = 5 | k = -1 *a* | b | pass

C. Où sont les difficultés dans les tests ?

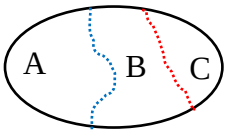
1. Difficulté mathématique : partition de l'ensemble des cas possibles.

➢ Tous les cas possibles à traiter constitue un ensemble qu'il va falloir souvent découper en plusieurs parties suivant les différents cas à traiter.

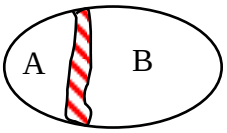
Un ensemble correctement découpé s'appelle en Mathématique **une partition**.

Le partitionnement (ou régionnement) d'un ensemble repose sur 2 règles mathématiques :

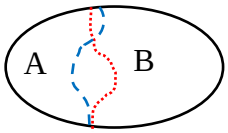
- Le recouvrement de l'ensemble total par l'ensemble des parties.
- La bonne jointure des parties entre elles.



Bonne partition !
Les parties A, B et C recouvrent bien l'ensemble total, sans chevauchement.
Tous les cas seront traités une seule fois !



Mauvaise partition !
Les parties A et B ne recouvrent qu'en partie l'ensemble total.
Les cas dans la bande ne seront pas traités !



Mauvaise partition !
Les parties A et B se chevauchent.
Les cas à l'intersection seront traités plusieurs fois au lieu d'une seule !

Mathématique : **Partionnement d'un ensemble en plusieurs parties.**	→	Informatique : **Plan de traitement de l'ensemble des différents cas à traiter et leur catégorisation.**
Recouvrement de l'ensemble total par les parties.	→	Existence d'au moins un traitement pour chaque cas : tous les cas auront (au moins) un traitement attendu !
Bonne jointure sans chevauchement des parties.	→	Unicité du traitement pour chaque cas : tous les cas seront traités (au plus) 1 fois et non pas 2 fois ou plus !
Recouvrement total + Bonne jointure	→	Chaque cas possible est traité de manière attendue 1 et 1 seule fois (mais pas forcément de manière individualisée).
Parmi les différents types de tests des 2 tableaux précédents, citer ceux qui assurent une partition de l'ensemble des cas à traiter : *if else (Alternative simple), if elif else (Alternative multiple complète).*		

➢ Application : Dans chacun des tests suivants, citer la ou les valeurs qui ne seront pas traitées (recouvrement partiel) ou qui seront traitées plusieurs fois (chevauchement des conditions) :

k = int(input( )) if k > = 5 : bloc ① if k < 7 : bloc ②	k = int(input( )) if k < 5 : bloc ① if k > 5 : bloc ②	k lettre minuscule if k < = "x" : bloc ① if k > = "a" : bloc ②	k lettre minuscule if k < = "y" : if k > = "b" : bloc ②
Recouvrement partiel □ Chevauchement x Soucis en *6.*	Recouvrement partiel x Chevauchement □ Soucis en *5.*	lettres non traitées : *y, z.* Chevauchement pour les lettres *entre a et x.*	lettres non traitées : *a, z.* *Pas de chevauchement.*
Remarque : Quand une condition contient des inégalités, toujours se poser la question « inégalités strictes ou non strictes ? » pour éviter les problèmes de chevauchement ou de recouvrement partiel !			

2. Difficulté d'ordre logique : écriture simplifiée des tests.

① Simplification du type de test :

➢ Le 2ème tableau p.9 montre que les 3 derniers types de tests avec des if imbriqués peuvent être simplifiés en un type de test plus simple, c-à-d sans imbrications de if.

Pour cela, il faut être capable de composer plusieurs conditions en une seule plus complexe. Puis s'assurer de l'équivalence du partitionnement résultant *en s'aidant par exemple du schéma ensembliste*.

➢ Application : Simplifier les tests suivants en tests plus simples, *sans if imbriqués* :

if condition1 : if condition2 : bloc ②	if k < = "x" : if k > = "a" : bloc ②	if k < = "x" : bloc ① if k > = "a" : bloc ②	if k > 4 : if k < = 10 : bloc ① else : bloc ②	if k > 4 : if k < = 10 : bloc ① else : bloc ②
équivaut à	équivaut à	équivaut à	équivaut à	équivaut à
if condition1 and condition2 : *bloc ②*	*if 'a' <= k <= 'x' :* *bloc ②*	*if k <= 'x' :* *bloc ①* *if 'a' <= k <= 'x' :* *bloc ②*	*if 4 < k <= 10 :* *bloc ①* *if not(k > 4) :* *bloc ②*	*if 4 < k <= 10 :* *bloc ①* *if (k > 4 and k > 10) :* *bloc ②*

② Simplification de la condition d'un test : utilisation de la négation.

➢ La capacité à clairement identifier une condition puis exprimer sa négation est indispensable en Algorithmique-Programmation. En effet :

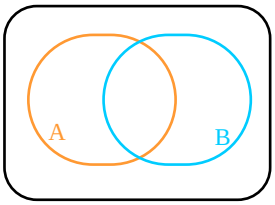
- cette capacité apparait implicitement dans « else : » : else correspond au reste donc ce qui n'est pas.
- lorsque la condition d'un test est une combinaison multiple de conditions simples, il est souvent bien plus facile d'écrire la condition contraire. Voir France IOI Niv 1 chap 7 exo 6.
  C'est comme en Probabilités, calculer la probabilité de l'évènement contraire est parfois bien plus facile que calculer celle de l'évènement direct.
- cette capacité sera aussi indispensable pour écrire la condition des boucles conditionnelles Tant que.

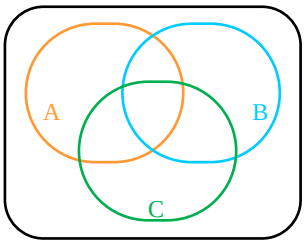
➢ **Il faut donc être capable par exemple d'écrire sans erreur la négation d'une condition multiple.**

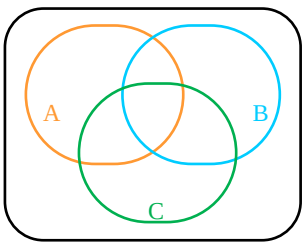
Notation : La négation (le contraire, le complémentaire) d'un ensemble A peut s'écrire $\bar{A}$.

Application : Ecrire la négation des conditions multiples suivantes. Aide : schémas ensemblistes !

Conditions	**A and B**	**A or B**	A and B and C	A or B or C	A or B and C $\Leftrightarrow$ *A or (B and C)*
Négations	$\bar{A}$ **or** $\bar{B}$	$\bar{A}$ **and** $\bar{B}$	$\bar{A}$ or $\bar{B}$ or $\bar{C}$	$\bar{A}$ and $\bar{B}$ and $\bar{C}$	$\bar{A}$ and $\overline{B \text{ and } C} \Leftrightarrow \bar{A}$ and ($\bar{B}$ or $\bar{C}$)







D. Quels types de tests privilégier ?

Au vu du C] précèdent (privilégier les tests à couverture complète et éviter les if imbriqués), on conclue :

Les 3 types de tests conditionnels à privilégier :

- La conditionnelle simple incomplète : if.
- La conditionnelle simple complète appelée aussi alternative simple : *if … else.*
- L'alternative multiple complète : *if .. elif .. else.*

E. Entraînement sur les tests :

❶ Un grand classique :

Ecrire un programme résolvant n'importe quelle équation de type $aX^2 + bX + c = 0$. Cas a = 0 ?

Il faut bien distinguer les cas sur les coefficients et sur le discriminant et privilégier les alternatives multiples complètes if elif else. Voici une solution possible

```
from math import sqrt  #import de la racine carrée
a = float(input("Entrer le coefficient a :"))
b = float(input("Entrer le coefficient b :"))
c = float(input("Entrer le coefficient c :"))
if a == 0 :
    if b ! = 0 :
    print(f"L'unique solution de l'équation du premier degré {b}X + {c} = 0 est -c/b = {-c/b}. ")
    else : # a = 0 et b = 0
        if c == 0 :
            print ("L'ensemble des solutions est ℝ.")
        else : # a = 0 et b = 0 et c ≠0
            print(« Aucune solution. »)
else :  # cas général où a différent de 0.
    delta = b**2 −4 * a * c
    if delta > 0 :
        sol1 = (-b −sqrt(delta))/(2 * a)
        sol2 = (-b + sqrt(delta))/(2 * a)
        print(f"Les 2 solutions réelles à l'équation {a}X^2 + {b}X + {c} =0 sont x1 ≈{sol1} et x2 ≈{sol2}. ")
    elif delta == 0 :
        print(f"1 solution double à l'équation {a}X^2 + {b}X + {c} =0 qui est x1 ≈{sol1}. ")
    else :   # delta négatif
        print( "Pas de solutions réelles. ")
```

❷ Soient 2 classes d'élèves caractérisés par leur taille et leur âge.

Pour chaque classe, on donne les tailles minimale et maximale, ainsi que les âges minimum et maximum.

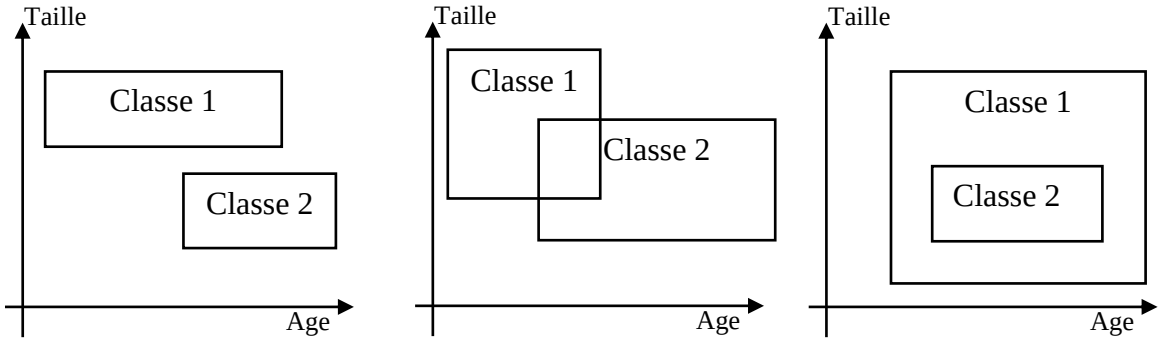
Ecrire un programme qui indique si oui ou non il est possible qu'il y ait des gens dans les 2 classes ayant même âge et même taille. Croquis ! (Aide : France IOI Niv 1 chap 7 exo 6 : intersection de 2 rectangles).

Analyse :

Comme d'habitude, on ne se lance pas dans la programmation sans avoir analyser la situation avec son cerveau, une feuille et un crayon à papier !

- *Illustrons la situation par un graphique : chaque classe est représentée sur le graphique par un rectangle de longueur [taille_min ; taille_max] et de hauteur [âge_min ; âge_max].*
  Répondre au problème revient à regarder la position des 2 rectangles l'un par rapport à l'autre.

Il n'y a que 3 cas possibles : non intersection, intersection et inclusion.



Les solutions au problème sont les 2 derniers cas : intersection et inclusion.

- *Considérons les coins inférieurs gauches de chaque rectangle. Il n'y a que 4 façons possibles de les placer l'un par rapport à l'autre. Puis il ne restera plus qu'à déterminer à quelles conditions en plus sur l'âge et la taille il y a intersection ou inclusion des 2 rectangles dans chaque cas :*
  - *Coin1 à gauche en haut du Coin2 : âge_min1 ≤ âge_min2 et taille_min1 ≥ taille_min2.*
    si (âge_max1 ≥ âge_min2 et taille_max2 ≥ taille_min1) alors il y a intersection voire inclusion.
  - *Coin1 à gauche en bas du Coin2 : âge_min1 ≤ âge_min2 et taille_min1 ≤ taille_min2.*
    si (âge_max1 ≥ âge_min2 et taille_max1 ≥ taille_min2) alors il y a intersection voire inclusion.
  - *Coin1 à droite en haut du Coin2 : âge_min1 ≥ âge_min2 et taille_min1 ≥ taille_min2.*
    si (âge_max2 ≥ âge_min1 et taille_max2 ≥ taille_min1) alors il y a intersection voire inclusion.
  - *Coin1 à droite en bas du Coin2 : âge_min1 ≥ âge_min2 et taille_min1 ≤ taille_min2.*
    si (âge_max2 ≥ âge_min1 et taille_max1 ≥ taille_min2) alors il y a intersection voire inclusion.

  On voit que le raisonnement pour trouver directement s'il y a intersection (ou inclusion) n'est pas du tout évident !!
- *En fait ici, comme dit au ② p.11 (Simplification de la condition d'un test : utilisation de la négation), quand on est comme ici dans le cas d'une condition très complexe, il vaut mieux passer par la négation.*
  A savoir ici : quand est-ce que les rectangles ne se rencontrent pas !
  Et il n'y a que 4 cas possibles de non interception-non inclusion des 2 rectangles :

  Soit âge_max1 < âge_min2 *Soit âge_min1 > âge_max2*
  Soit taille_max1 < taille_min2 *Soit taille_min1 > taille_max2*

  Finalement, le contraire de ces 4 conditions combinées donnera l'intersection-inclusion. More simple !
- *Synthèse : compléter le programme python résultant de l'analyse.*

```
# Entrée des âges tailles minimum maximum pour les 2 classes.
# Tests.
if ..... :
    print( "Pas de gens dans les 2 classes ayant même âge et même taille. ")
else :    # .....
    print( "Il y a des gens dans les 2 classes qui ont le même âge et le même poids. ")
```

On pourrait améliorer le script en affichant les rectangles et en montrant l'intersection.

Ai-je tout compris ? Tests conditionnels.	☹	😐	☺	☺☺
Syntaxe des tests.				
Visualiser sur un schéma ensembliste conditions et traitements associés.				
Partitionner un ensemble de conditions.				
Simplifier une suite de if imbriqués.				
Les 3 types de test à privilégier.				
Utiliser les tests dans des cas simples et moins simples (if imbriqués).				

III. REPETITIONS DONT LE NB N'EST PAS CONNU A L'AVANCE.

➢ Dans la vraie vie, pas difficile de trouver des exemples d'actions répétées qui seront exécutées tant que certaine(s) condition(s) sont présentes. Ex : « Tant qu'il y a des voitures, ne pas traverser. », « Battre les œufs jusqu'à avoir une omelette. », « Bosser les maths tant que le niveau est faible. », etc.

Autre exemple ? *Bosser l'Informatique jusqu'à ce que le niveau soit bon etc.*

Connait-on à l'avance le nombre de fois où la ou les actions seront répétées ? *Non.*

➢ L'Algorithmique étant une traduction mécanique de la Vie, il eut donc été tout bonnement impensable qu'il n'existât pas la même structure de répétitions conditionnées en Algorithmique-Programmation !

En Python (mais aussi dans bien d'autres langages : *Langage C, Java, C#, Go etc.*), l'instruction qui permet d'exécuter en boucle, sous conditions, une ou plusieurs actions s'appelle la boucle *While.*

A. Syntaxe de la boucle indéterminée While :

<table>
<tr><th>Modèle</th><th>Explications</th></tr>
<tr><td>
while Condition(s) :

Corps de la boucle

Suite du programme
</td><td rowspan="2">
• while Condition(s) :
Entête de la boucle commençant par le mot réservé while.
Se termine obligatoirement par « : ».
Condition(s) est une expression booléenne plus ou moins simple qui conditionne l'exécution ou non du corps de boucle.
Cette expression doit être écrite comme
<u>la négation de la condition d'arrêt de la boucle</u>.
• Corps de la boucle ou Bloc d'instructions :
Ce seront la ou les instructions qui seront répétées tant que la condition est vraie.
Ce bloc est obligatoirement indenté vers la droite.
• Suite du programme :
Dès que la condition devient fausse, le programme sort de la boucle et passe directement à la 1ère instruction non indentée, sans exécuter le corps de boucle.
</td></tr>
<tr><td>Traduction en Français</td></tr>
<tr><td>
Tant que la condition est vérifiée, faire :
Corps de la boucle.
Dès que la condition n'est plus vérifiée, passer directement à :
Suite du programme.
</td></tr>
</table>

Exemple et commentaires	
k = 4 while k > 1 : print ('♥ Maths') k = k – 1 print('Super !')	Condition d'arrêt ? *k ≤ 1* Nb de tours de boucle ? *3.* Combien vaut k à la fin ? *1* Nb de fois la condition k > 1 a été testée ? *4 fois.* • **La condition dépend souvent de variables, à définir forcément avant le while !** • **Dans le corps de boucle, agir forcément sur les variables de la condition !**
Remarques	
• Dans une boucle While, le nombre de tours de boucle n'est pas *déterminé* car conditionné. • Puisque la condition est quand même vérifiée avant la sortie de boucle, alors cela signifie que **la condition sera toujours vérifiée 1 fois de plus que le nombre de tours de boucle**.	

B. Où sont les difficultés dans les boucles While ?

1. Ecrire correctement la condition dans l'entête après le mot while :

➢ Une boucle While doit continuer tant qu'il faut continuer et s'arrêter dès qu'il faut s'arrêter ! Cette tautologie (évidence) dit simplement qu'une boucle While s'arrête dès qu'une certaine condition d'arrêt est vérifiée et continue sinon. D'où :

La condition dans l'entête de la boucle While doit être la négation de sa condition d'arrêt !

➢ En pratique :

- Expliciter la ou les conditions de sortie (d'arrêt) de la boucle.
- Puis écrire dans l'entête de la boucle While la négation logique de ces conditions d'arrêt.
  Mettre juste le mot « not » devant cette condition d'arrêt si celle-ci est trop compliquée !

➢ Application : Compléter en Python les boucles suivantes dont on donne la condition d'arrêt en français.

Condition d'arrêt	k pair	k supérieur ou égal à 1 et p différent de m	f égal à h ou t inférieur à 3
Boucle while	while *k % 2 == 1 :* Bloc d'instructions	while *k < 1 or p == m :* Bloc d'instructions	while *f != h and t >= 3 :* Bloc d'instructions

Inversement, pour les boucles suivantes, écrire en bon français la condition d'arrêt.

Boucle while	while nb % 7 == 0 : Bloc d'instructions	while k % 3 != 0 or k > 8 : Bloc d'instructions	while type(k) == str : Bloc d'instructions
Condition d'arrêt	nb non multiple de 7.	k multiple de 3 et k inférieur ou égal à 8.	k n'est pas un caractère.

2. Le problème des boucles infinies inattendues :

➢ La boucle While donnée en exemple page précédente n'est pas un bon exemple ! En effet, le nombre de tours est en fait bien déterminé d'avance (de 4 inclus au départ jusqu'à 2 inclus à la fin en enlevant 1 à chaque tour, cela fait 3 tours) ! Donc il s'agissait en fait d'une boucle *For* déguisée en boucle *While*.

Dans une vraie boucle While, non seulement on ne connaît pas à l'avance le nombre de tours mais pire ce nombre de tours peut être infini de manière complètement inattendue !

Comment une boucle While devient-elle une boucle infinie ?

Il faut et il suffit que la condition dans l'entête soit encore et toujours *vraie* avant chaque tour de boucle !

Ex1 : `while 1 > 0 :` `Bloc d'instructions`	Ex2 : `k = 2` `while k < 3 :` `k = k – 1`	Ex3 : `k = 3` `while k > 1 or k < 2 :` `k = k – 1`
1 > 0 toujours vrai quelque soit les instructions contenues dans Bloc (sauf s'il contient un break).	k est un déccumulateur (k = k – 1) partant de 2. Donc sa valeur sera toujours inférieure à 3.	Il suffit que l'une ou l'autre des 2 conditions soit vraie ! C'est toujours le cas ici !

Ces boucles infinies *inattendues* sont cause de nombreux bugs : derrière un programme qui freeze peut se cacher une boucle infinie inattendue.

Dans ce cas, on parle aussi de boucle divergente, de non-terminaison, de programme bouclant indéfiniment.

➢ Application : Quelles sont les boucles seront divergentes et pourquoi ? Puis vérifier à l'ordinateur.

`k = 2` `while k > 3 :` `k = k + 1`	`k = 3.1` `while type(k) != int :` `k = k – 0.1`	`k = 4` `while k > 3 or k < 3 :` `k = k – 1`	`k = 1` `while k != 0.0 :` `k = k – 0.1`	`Séquence = [0]` `for k in Séquence :` `Séquence.extend([1])`
boucle non exécutée : *Dès le début,* *2 < 3.*	*boucle infinie.* *k sera toujours un float, même lorsque k = 3.0 qui n'est pas de type int.*	*boucle finie.* *Après le 1er tour, k = 3* *→ sortie de boucle.*	*Attention ! Boucle inifnie.* *k ne vaudra jamais 0.0. Voir cours Représentation des nombres réels.*	*Attention ! Boucle infinie.* *On a vu p.6 que le extend étendait à l'infini Séquence et donc la boucle For devenait infinie ! Shocking !*

➢ Attention : On parle bien ici des boucles infinies *inattendues*, dues à des erreurs de programmation !

A contrario, les boucles infinies *attendues* sont importantes en programmation : elles apparaissent par exemple lorsqu'on veut faire tourner indéfiniment des processus ; ou en programmation évènementielle (par exemple afficher indéfiniment une page jusqu'à l'évènement cliquer sur la fermeture de page).

Ces boucles infinies attendues sont en général de la forme : `while True :`
`Bloc avec break à l'intérieur.`

C. Conseils pour les boucles While :

Conseils	*Exemple ou contre-exemple*	*Commentaires*
• **Bien initialiser les variables conditionnelles et les variables internes avant le while.** Sinon non-entrée dans la boucle ou NameError assurés !	`Somme = int(input( ))` `while Somme < 10 :` `k = k + 1` `Somme = Somme + k**2`	Pour quelles valeurs entrées, aura-t-on une non-entrée dans la boucle ? *Toute valeur ≥ 10.* Pour une entrée < 10, qu'aura-t-on ? *NameError : k is undefined.*

• **Eviter d'initialiser une variable interne dans la boucle elle-même !** Comportement non attendu assuré !	Somme = 0 while Somme < 10 : Somme = 0 Somme = Somme + 1	A chaque tour, Somme est réinitialisée à *0* ! Que se passe-t-il alors ? Somme vaut toujours 0 puis 1 etc.
• ***Dans le corps de boucle, avoir obligatoirement des instructions qui agissent sur la condition, afin de la rendre fausse à un moment donné.*** Sinon grand risque de boucle infinie !	Séquence = [0] while Séquence != [ ] : print(Séquence)	Séquence est-elle modifiée par le corps de la boucle ? *Non !* Que se passe-t-il alors ? *La condition est toujours vraie* *→ boucle infinie.*
• ***Contrôler attentivement le début et la fin de la boucle.*** Pour éviter comportements inattendus et/ou traitements incomplets !	a,N = int(input()) , int(input()) mult = 0. while a * mult < N : mult = mult + 1 print(a * mult)	Obtient-on bien le plus grand multiple de a contenu dans N ?
• Non. On obtient le multiple qui dépasse N, à cause de la condition qui est testée toujours 1 fois de plus que le nb de tours de boucle. Il faut donc prendre mult – 1 dans le print( ). *• Attention aussi dans la condition du while : la condition d'arrêt de la boucle est : a * mult > N. Donc la condition du while doit en être la négation c-à-d : a * mult <= N.*		

D. Entraînement sur les boucles While :

❶ Soit un entier N ≠ 0, écrire un script qui renvoie l'exposant (noté exp) de la plus grande puissance de 2 contenue dans N. Ex : pour l'entrée N = 35, le programme renverra exp = 5. En effet, 35 est entre 2^5 et 2^6.

❷ Soit N un entier écrit en base 10. Ecrire un programme qui renvoie le nombre de chiffres qui compose N. Aide : suite de divisions euclidiennes (divisions entières) par la base qui vaut ici 10 + un compteur.

❸ Soit un entier généré au hasard entre 15 et 123. Ecrire un programme qui joue au jeu du « Plus grand Plus petit » : le joueur humain propose un nombre et le programme répond par « plus grand » ou « plus petit » jusqu'à ce que le joueur humain trouve le nombre généré au début et affiche le nb de coups effectués.

❹ Soient m et n, 2 entiers. Ecrire un programme qui calcule leur pgcd selon l'algorithme d'Euclide.

❶ Cela revient à trouver exp tel que : $2^{exp} \geq N$.

```
Entrer N. Et exp ← 0.
Tant que 2**exp < N :    # contraire de 2^exp ≥ N.
        exp ← exp + 1
# dès que ça dépasse ou égal, c'est le bon !
Afficher (exp)
```

❷ Entrer N. compteur_chiffres ← 0

```
Quotient ← N
Tant que Quotient > 0 :
        Quotient ← Quotient // 10
        compteur_chiffres ← compteur_chiffres + 1
Si N == 0 :
    Afficher (1)
Sinon Afficher (compteur_chiffres)
```

❸ from random import randint

nb_cherché ← randint(15 , 123)

N ← nb_cherché + 1 # initialisation du nb proposé à nb_cherché + 1 pour être sûr qu'il est différent de nb_cherché et ainsi rentrer dans la boucle.

Nb_coups ← 0

Tant que N ≠ nb_cherché :

    Entrer N l'entier proposé

    Nb_coups ← Nb_coups + 1

    Si N > nb_cherché :

        Afficher (Trop grand)

    Si N < nb_cherché

        Afficher (Trop petit)

Afficher(Vous avez trouvé {nb_cherché} en {Nb_coups} coups.)

❹ Voir Terminale Corrigé Contrôle 1 2018-2019 Exercice 4.

Ai-je tout compris ? Boucles conditionnelles While.	☹	😐	☺	☺☺
Syntaxe de la boucle While.				
Ecrire correctement la condition dans l'entête à partir de conditions d'arrêt.				
Problème des boucles infinies inattendues et comment les éviter.				
Contrôler le début et la fin de boucle.				
Utiliser la boucle While dans des cas simples.				

IV. COMPLEMENTS SUR LES BOUCLES FOR ET WHILE.

A. Instructions complémentaires pour les boucles :

Voici des instructions facultatives qui permettent d'assouplir et/ou raffiner le comportement d'une boucle.

1. Break : instruction de sortie conditionnée d'une boucle.

Syntaxe du Break	*Explications sur l'instruction Break*
for Variable de boucle in Sequence : Bloc facultatif d'instructions ① if Condition(s) : break Bloc facultatif d'instructions ② Suite du programme	• Interne à une boucle For (ou While). • Associé à Conditions, interrompt l'exécution de la boucle et passe directement à Suite du programme. • Permet de fabriquer des **boucles itérativo-conditionnelles : les boucles For-Break.** Ce sont des boucles qui doivent s'exécuter a priori un nombre déterminé de fois mais qui peuvent s'interrompre lors d'un certain évènement. • Exemple : Traitement normal (blocs ①②) sur plusieurs heures mais abandon (break) du reste du traitement (bloc ②) lors du déclenchement d'une alarme (Condition(s)).

Remarques
• Marche aussi avec les boucles While selon la même syntaxe. • Utilisé en particulier avec des boucles While infinies attendues de type while True : Permet de simuler les boucles de type (Faire … jusqu'à ce que) existant parfois dans d'autres langages. • Toute boucle For-break est plus ou moins facilement remplaçable par une boucle While mais qui pourrait être moins claire surtout au niveau du conditionnement.

2. Else : instruction confirmative associée à une boucle avec Break.

Syntaxe du Else associé à une boucle	*Explications sur la clause Else associée à une boucle*
for Variable de boucle in Séquence : Bloc d'instructions facultatif ① if Condition(s) : break Bloc d'instructions facultatif ② else : Bloc d'instructions ③ Suite du programme	• Associée à une boucle For (ou While)-break. Facultative. • **Bloc ③ du else exécuté que si le break n'a pas été activé.** • Permet de compléter les boucles itérativo-conditionnelles (boucles For-Break) avec des instructions supplémentaires dans le cas de terminaison « normale (sans break) » de la boucle. • Exemple : Traitement normal sur une journée (blocs ① et ②) puis confirmation que tout est OK (bloc ③). Mais abandon (break) de la boucle lors du déclenchement d'une alarme (Condition(s)) et passage directement à Suite du programme, sans confirmation (sans bloc ③).

3. Continue : instruction de saut de tour conditionné dans une boucle.

Syntaxe du Continue	*Explications sur l'instruction Continue*
for Variable de boucle in Sequence : Bloc d'instructions facultatif ① if Condition(s) : continue Bloc d'instructions facultatif ② Suite du programme	• Interne à une boucle For (ou While). • Associé à Conditions, interrompt le tour de boucle et passe directement au tour de boucle suivant, sans exécuter le bloc ②. • Permet de fabriquer un **autre type de boucles itérativo-conditionnelles : les boucles For-conditionnelles.** Ce sont des boucles qui doivent s'exécuter a priori un nombre déterminé de fois sauf pour certains cas précis. • Exemple : Traitement normal (blocs ①②) sur une journée sauf aux moments où il pleut (Condition) : pas de bloc ② (continue).

Remarques
• Marche aussi avec les boucles While selon la même syntaxe. • Toute boucle For contenant un Continue peut-être remplacée par une boucle While plus ou moins facilement conditionnée. • Continue s'apparente à l'instruction Go To du langage Basic qui permet d'aller à une certaine ligne.

4. Application :

En utilisant les instructions break, continue et else, écrire un programme qui demande à l'utilisateur d'entrer un certain nb de notes entre 0 et 20, s'arrêtera dès que la note est supérieure à 20, calculera leur moyenne, sans tenir compte d'éventuels zéros, et affichera s'il n'y a pas eu de break « Tout est Ok » et la moyenne.

```
nb_notes = int(input("Entrer le nombre de notes : "))
somme=0
for numero_note in range(nb_notes) :
    note = float(input(f"Entrer la note {numero_note + 1} : "))
    if note >=20 :
        break
    if note == 0 :
        nb_notes = nb_notes – 1
        continue
    somme = somme + note
else :          # Tout c'est bien passé : pas de break donc pas de note >= 20.
    moyenne = somme/nb_notes
    print(f"Tout est OK. Il y avait {nb_notes} notes non nulles. Leur moyenne est {moyenne}.")
```

Les notes nulles ne sont pas prises en compte dans le calcul de la somme totale (instruction continue) et il ne faut pas oublier de décrémenter le nb de notes pour le calcul de la moyenne.

B. Comparatif boucle For et boucle While :

	Boucle For	Boucle While
Autres noms.	*Boucle bornée, déterminée.*	*Boucle conditionnelle, non bornée, non déterminée.*
Indice de boucle.	*Présent d'office dans l'entête. C'est la variable de boucle.*	*Absent d'office. Mettre un accumulateur dans le corps de boucle.*
Parcours d'une séquence.	*Direct, dans l'entête. Du début jusqu'à la fin de la séquence.*	*Indirect. Conditionné.*
Nombre de tours de boucle ?	*Déterminé a priori = longueur de la séquence.*	*Indéterminé a priori.*
Exécution du corps de boucle.	*Automatique.*	*Conditionnée.*
Break, Else, Continue.	*Oui*	*Oui*
A utiliser	*Lorsque le nb de répétitions est facilement connu a priori.*	*Lorsque le nb de répétitions est vraiment inconnu a priori car dépendant d'une condition.*

C. Boucles imbriquées :

Définition : On appelle boucles imbriquées des boucles qui sont l'une dans l'autre.

Exemple d'une boucle While dans une boucle For :

```
for k in range(n) :
    while k < 10 :
        print ('ok')
```

2 types de boucles imbriquées :

Les boucles dont le nombre de tours de l'une ne dépend pas de l'autre.	Les boucles dont le nombre de tours de l'une dépend de l'autre.
• Dans l'exemple plus haut, les boucles sont-elles « indépendantes » ? *Non car le nb de tours du while dépend de la valeur de k dans l'entête de la boucle externe.*	`j = 10` `for k in range(10) :` `  while j > k :` `  j = j – 1` *Ici, la condition de la boucle interne dépend de k qui est dans la boucle externe.*
• `for k in range(10) :` `  for j in [1, 2 , 5 , k]` `  print ( j )` *Ici, le nb de tours de la boucle interne ne dépend pas de la valeur de k dans la boucle externe.*	*Nb total de tours = 1 × 10*

Nombre total de tours :

nb de tours de l'une × nb de tours de l'autre.	Pas de formule simple.

Imbriquer des boucles est indispensable en Algorithmique-Programmation, mais source de nombreuses erreurs surtout quand l'entête de la boucle intérieure dépend de la boucle extérieure.

➢ Exercice ❶ :

1. Les boucles suivantes sont-elles « indépendantes » ? *Evidemment que non !* Compléter.

N = k = 0 while k < = 9 : while k < = 8 : k = k + 1 N = N + 1 k = k + 1	N = k = 0 while k < = 9 : while k < 8 : k = k + 1 N = N + 1 k = k + 1	N = k = 0 while k < 9 : while k < = 8 : k = k + 1 N = N + 1 k = k + 1	N = k = 0 while k < = 9 : while k < 8 : k = k + 1 N = N + 1 k = k + 1
nb petits tours = 9 nb grands tours = 1	nb petits tours = 8 + 0 nb grands tours = 1 + 1	nb petits tours = 9 nb grands tours = 1	nb petits tours = 8 + 0 nb grands tours = 1 + 1
N vaut 9 et k vaut 10.	N vaut 8 et k vaut 10.	N vaut 1 et k vaut 10.	N vaut 2 et k vaut 10.

On voit que juste changer le caractère strict-large d'une inégalité ou le niveau d'indentation peut complètement changer le résultat attendu, et ce d'autant plus qu'il y a d'imbrications !

2. S'agit-il ici de vraies boucles While ou de boucles For déguisées ? Justifier.

Il s'agit de boucle For car on connait a priori le nombre de tours, même si parfois ce n'est pas évident !

3. Ecrire les 2ème et 3ème cas en remplaçant la boucle externe While par une boucle For.

On remplace l'entête « while etc. » par l'entête « for j in range(nb de grands tours) : » et on supprime l'incrémentation externe k = k + 1

➢ Exercice ❷ : Ecrire le calcul qui donne la valeur de l'accumulateur à la fin des scripts suivants :

Facile	Facile	Moins facile	Moins Facile	Difficile
j = 0, n entier > 0 for i in range(n) : j = j + 1	a = 0, n et t entiers for k in range(n) : for y in range(t) : a = a + 1	p = 0, n entier > 0 for k in range(n) : for y in range(k) : p = p + 1	b = 0, n entier > 0 for k in range(n) : for y in range(n – k) : b = b + 1	p = 0, n entier > 0 for k in range(n) : while p < k : p = p + 1
j vaut n	*a vaut t × n*	*p vaut 1 + 2 + 3 + …. + (n – 1)*	*b vaut (n – 1) + …… + 3 + 2 + 1*	*p vaut n – 1 (faire un tableau d'état)*

Correction du Difficile ! Utilisons un tableau d'état des variables :

Numéro du tour	*1er tour*	*2ème tour*	*3ème tour*	*4ème tour*	*dernier tour n*
Valeur de k	*0*	*1*	*2*	*3*	*n – 1*
Condition p < k	*0 < 0 False*	*0 < 1 True*	*1 < 2 True*	*2 < 3 True*	*n – 2 < n – 1 True*
Valeur de p	*0*	*1*	*2*	*3*	*n – 1*

Ai-je tout compris ? Compléments Boucles.	☹	😐	☺	☺☺
Instruction d'arrêt conditionné Break dans une boucle et clause confirmative Else associée à cette boucle.				
Instruction de saut de tour conditionné Continue dans une boucle.				
Différences entre boucles While et For.				
Utiliser les boucles For et While dans des cas complexes : boucles imbriquées.				