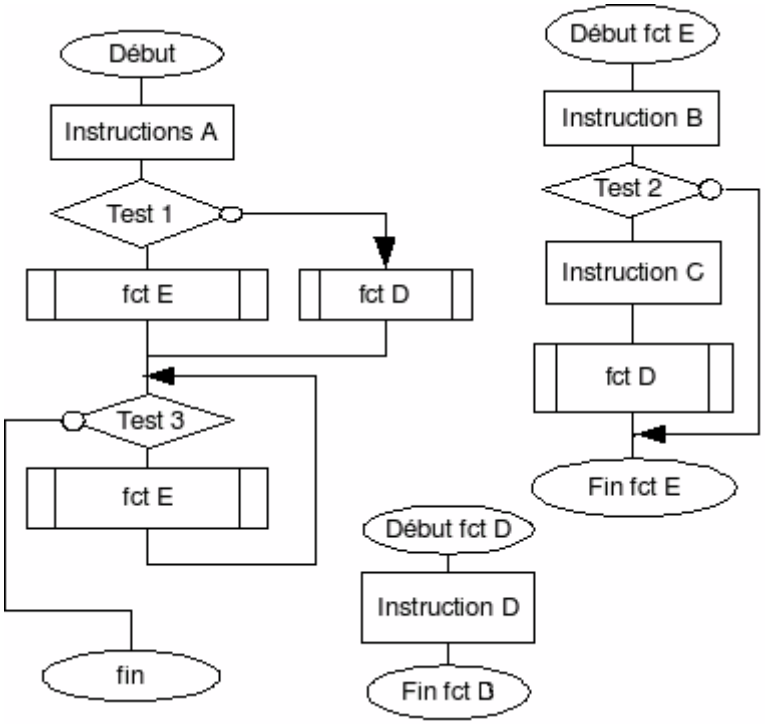
INSTRUCTIONS COMPOSÉES EN PYTHON (1) : BOUCLE FOR, TESTS, BOUCLE WHILE.

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.

La répétition la plus simple est celle dont le nombre ……..……

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

A. Syntaxe de la boucle déterminée For :

<table>
<tr><th>Modèle</th><th>Explications</th></tr>
<tr><td rowspan="2">for Variable de boucle in Séquence :

Corps de la boucle
à répéter

Suite du programme</td><td rowspan="2">• for Variable de boucle in Séquence :
Entête de la boucle constituée des mots réservés ……..... et ……
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'……..…. 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 ……..……. 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 1ère instruction non indentée qui n'est donc plus dans le corps de la boucle.</td></tr>
<tr><td>Traduction en Français
Pour le nombre de fois où Variable est affectée aux éléments de Séquence, exécuter :
Corps de la boucle.
Suite du programme</td></tr>
<tr><td colspan="2">Exemples d'entêtes de boucle For et du nombre de tours de boucle associé</td></tr>
<tr><td colspan="2">
<table>
<tr><td>for element in ['nb', 3 , k] :
→ ..... tours</td><td>for lettre in 'abracadabra' :
→ ..... tours</td><td>for loop in range(5) :
→ ..... tours</td><td>for nb in range(p) :
→ ..... tours</td></tr>
</table>
</td></tr>
</table>

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	*Sortie*	*Exemples à vérifier à la console*
range(fin) • fin est un entier positif.	• similaire à ⟦ 0 ; fin ⟦, c-à-d la liste des entiers **de 0 inclus jusqu'à l'entier fin exclue.** • similaire à ⟦ 0 ; fin – 1 ⟧.	• range(4) similaire à [0 , 1 , 2 , 3] • range(3) similaire à [……..…..] • range(k) similaire à [……] • range(1) similaire à […….] • range(0) similaire à [
range(début , fin) • début et fin sont des entiers des positifs ou négatifs. • début ≤ fin.	• similaire à ⟦ début ; fin ⟦, la liste des entiers **de début inclus jusqu'à fin exclue.** • similaire à ⟦ début ; fin – 1 ⟧.	• range(2 , 5) similaire à [2 , 3 , 4] • range(1 , 4) similaire à [……..] • range(-1 , 2) similaire à [……..…...] • range(-1 , k) similaire à […….....….] • range(4 , 1) similaire à [
range(début , fin , pas) • début , fin et pas sont des entiers positifs ou négatifs.	• similaire à la liste de tous les entiers partant **de début inclus, itéré (augmenté) à chaque fois du pas p, jusqu'à fin exclue.**	• range(1, 10, 2) similaire à [1 , 3 , 5 , 7 , 9] • range(2, 8, 3) similaire à […..…….....] • range(5, 0, -2) similaire à [……..……...] • range(-1, -3, -1) similaire à […..…….....] • range(1, 3, -1) similaire à [

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 …….
2. range(fin) = range(…. , fin) = range(… , fin , …)

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

for k in range(2 , 5) : → …… tours	for j in range(k + 1) : → …….. tours	for nbs in range(2 , 10, 3) : → …… tours	for p in range(3,-7,-2) : → …… 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)	for k in range(1 , 3) : k = k * 2 print (k)	for k in range(6) : if k % 2 == 1 : print (k + 1)	for k in range(6) : if k % 2 == 0 : print (k)
for k in range(3) : k = k * 2 print (k + 2)	for k in range(2 , 6 , 2) : print (k)	for k in range(2 , 7 , 2) : print (k)	for k in range(2 , 8 , 2) : print (k)

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

❸ Ecrire un programme qui affiche :

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

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 [……...	L'indice de boucle i entier est initialisé à …… puis incrémenté (autoaffecté et augmenté) de …… à chaque tour jusqu'à ce que i valle …..	
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 :

-
-

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 qui 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 : …….	accumulateur : …….	déccumulateur : ……..……..

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 dans la boucle elle-même une variable apparaissant dans la boucle !**
Comportement non attendu assuré !
	for k in range(11) :
	  Somme = 0
	  Somme = Somme + k**2
		A chaque tour, Somme est réinitialisée à …….. !
		Combien vaudra Somme en sortie ?
		Somme = ……

• **Eviter de 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 ? ……
		A chaque tour, Séquence va être étendu de la valeur k ! Que vaut Séquence ? ……..

• **Eviter de 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-il indéfiniment (donc boucle ……... ?). En fait non ! Combien vaut n à la fin ? …..
• ***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 ?

Ai-je tout compris ? Boucles 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 le corps d'une boucle.				
Compteur dans le corps d'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.

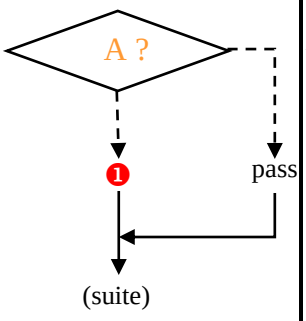
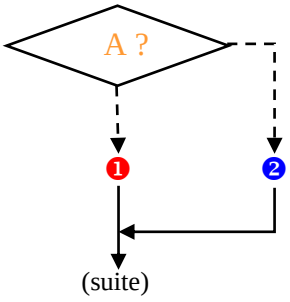
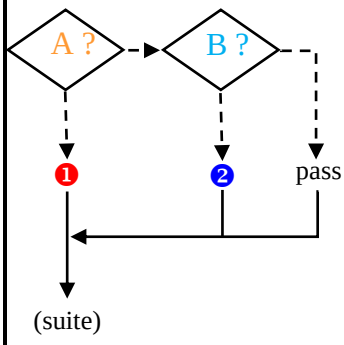
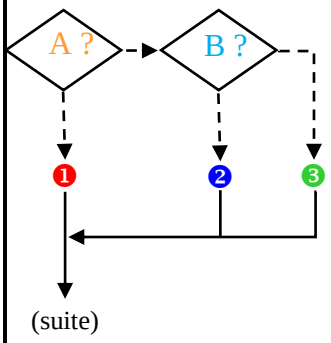
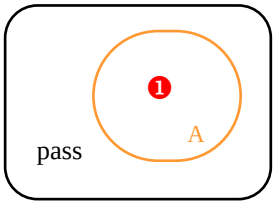
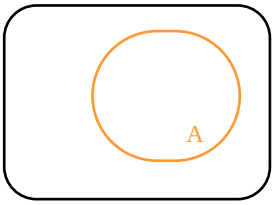
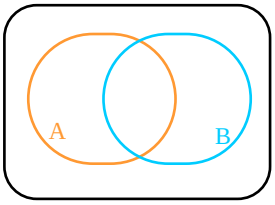
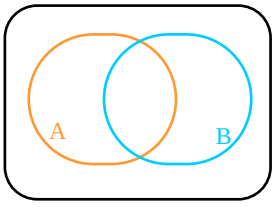
Le traitement conditionnel des données est l'autre structure de base de contrôle du flux (avec les boucles) qui permet d'adapter un algorithme un programme à quasiment n'importe quelle situation.

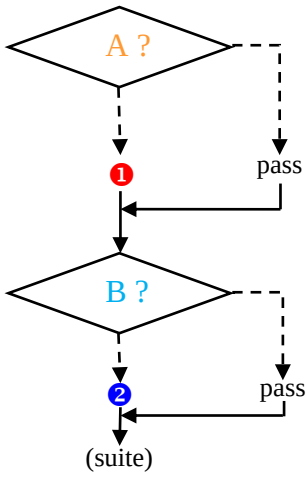
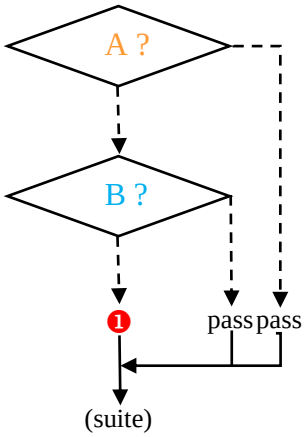
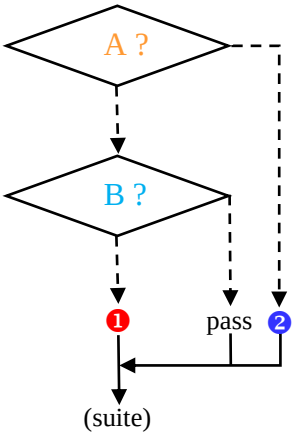
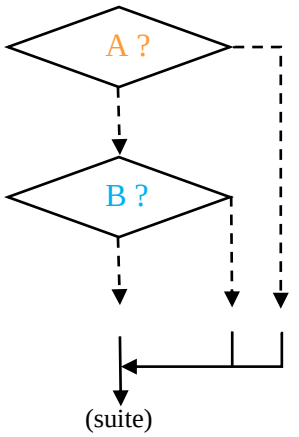
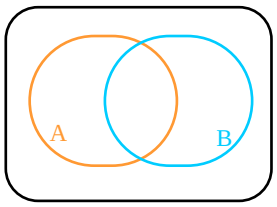
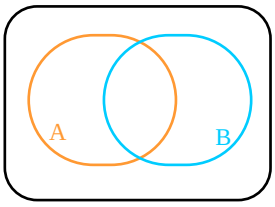
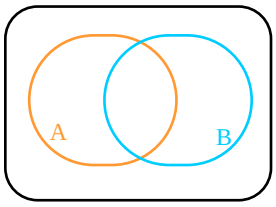
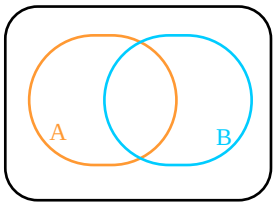
A. Syntaxe des tests conditionnels :

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 …….. (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 ……..…… (« 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 ……. (« 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* ⟺ *if* *else* *if*	***if*** ***elif*** ⟺ ***else***
Syntaxe A, B conditions. ❶, ❷, ❸ instructions	if A : ❶ (suite)	if A : ❶ else : # ❷ (suite)	if A : ❶ elif B : ❷ (suite)	if A : ❶ elif B : ❷ else : # ❸ (suite)
Algorigramme				
Schémas ensembliste				
Condition pour obtenir	Sorties possibles ❶ | pass A | $\bar{A}$ (not A)	Sorties possibles ❶ | ❷	Sorties possibles ❶ | ❷ | pass	Sorties possibles ❶ | ❷ | ❸
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 ……..… n'est pas traitée.	Sorties complètes : partition selon 3 cas possibles : ……
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 = …. | a	k = 10 | k = …. | k = ... | pass | b	k = 5 | k = …. | k = 9 | c |

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	Sorties possibles ❶ pass	Sorties possibles ❶ pass ❷	Sorties possibles ❶ ❷ pass
Remarque	Sortie incomplète : la sortie pour …… n'est pas traitée.	Sortie incomplète : la sortie pour …… n'est pas traitée.	Sortie incomplète : la sortie pour …… n'est pas traitée.	Sortie incomplète : la sortie pour …… 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 = ... pass	k = 0 | k = 4 | k = …. a	k = … | k = 5 | k = 0 a	k = 1 | k = …. | k = …. b | pass

C. Où sont les difficultés dans les tests ?

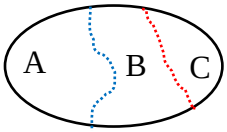
1. Difficulté mathématique : partition de l'ensemble des cas possibles.

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

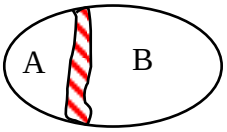
Un ensemble correctement découpé s'appelle en Mathématique **une partition**.

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

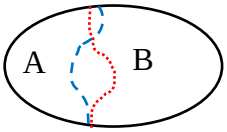
- Le recouvrement de l'ensemble total par l'ensemble des parties.
- La bonne jointure des parties entre elles.



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 : …….		

➢ 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 □ Soucis en ……..	Recouvrement partiel □ Chevauchement □ Soucis en ……..	lettres non traitées : …… Chevauchement pour les lettres de …….. à …..…	lettres non traitées : …… 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 k in Tab :
  if A != 4 :
  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

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 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
Négations					

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 : ……
- L'alternative multiple complète : ……

E. Entraînement sur les tests :

❶ Un grand classique :

Ecrire un programme qui résout n'importe quelle équation du second degré $aX^2 + bX + c = 0$. ⚠ cas a = 0 !

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).

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 ? ……..

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

➢ 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 : ……..……), l'instruction qui permet d'exécuter en boucle, sous conditions, une ou plusieurs actions s'appelle la boucle ……

A. Syntaxe de la boucle indéterminée While :

Modèle	*Explications*
while Condition(s) **:** Corps de la boucle à répéter Suite du programme	• **while** Condition(s) **:** Entête de la boucle commençant par le mot réservé ……..…… 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*** ***la négation de la condition d'arrêt de la boucle.*** • Corps de la boucle ou Bloc d'instructions : Ce seront la ou les instructions qui seront répétées tant que la condition est ☐ vraie ? ☐ fausse ? Ce bloc est obligatoirement ……. vers la droite. • Suite du programme : Dès que la condition devient ……..., le programme sort de la boucle et passe directement à la Suite du programme, *sans exécuter le corps de boucle*.
Traduction en Français	
Tant que la condition est vraie, faire : Corps de la boucle. Dès que la condition devient fausse, passer directement à : Suite du programme.	

Exemple et commentaires

```
k = 4
while k > 1 :
    print ('♥ Maths')
    k = k – 1
print('Super !')
```

Condition d'arrêt ? …..... Nb de tours de boucle ? …….

Combien vaut k à la fin ? …… Nb de fois la condition k > 1 a été testée ? .....

• **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 ……..… 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 ! Homer Simpson n'aurait pas dit mieux. Cette tautologie (évidence) dit simplement qu'une boucle While s'arrête dès que la condition d'arrêt est vérifiée et continue sinon. D'où :

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

➢ <u>En pratique :</u>

- 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 !

➢ <u>Application :</u> 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 ..... : Bloc d'instructions	while ..... : Bloc d'instructions	while ..... : 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			

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 …….. déguisée en boucle ……..…
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 !

<u>Comment une boucle While devient-elle une boucle infinie ?</u>

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

<u>Exemples :</u>

while 1 > 0 : Bloc d'instructions	k = 2 while k < 3 : k = k – 1	k = 3 while k > 1 or k < 2 : k = k – 1
1 > 0 toujours vrai quelques soient les instructions 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])
			Attention !	Attention !

➢ 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 Name error 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 ? …… Pour une entrée < 10, qu'aura-t-on ? ……..
• **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 …….. ! Que se passe-t-il alors ?
• ***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 ? ……. Que se passe-t-il alors ?
• ***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('multiple = ', mult)	S'affiche-t-il bien le plus grand multiple de a contenu dans 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 ……. + 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.

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 facultatif d'instructions ①
  if Condition(s) :
  break
  Bloc facultatif d'instructions 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 en 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.

B. Comparatif boucle For et boucle While :

	Boucle For	Boucle While
Autres noms.		
Indice de boucle.		
Parcours d'une séquence.		
Nombre de tours de boucle ?		
Exécution du corps de boucle.		
Break, Else, Continue.		
A utiliser lorsque :		

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 » ?
 	
• Y-a-t-il « indépendance » des boucles suivantes ?
`for k in range(10) :`
`  for j in [1, 2 , 5 , k]`
`  print ( j )`
	Les boucles suivantes sont-elles « indépendantes » ?
	`j = 10`
	`for k in range(10) :`
	`  while j > k :`
	`  j = j – 1`

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 » ? …… 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 = nb grands tours =	nb petits tours = nb grands tours =	nb petits tours = nb grands tours =	nb petits tours = nb grands tours =
N vaut …. et k vaut ….	N vaut …. et k vaut ….	N vaut …. et k vaut ….	N vaut …. et k vaut ….

Attention ! On voit que changer juste 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.

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

➢ 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

				Faire un tableau d'états de la boucle !

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.				

Après avoir vu les structures de contrôle fondamentales que sont les boucles et les tests, nous allons aborder dans le prochain livret une autre structure de contrôle fondamentale : les …….