

Conditions et boucles

Tout ce dont vous avez besoin pour ce module, dans un seul document. Cochez au fur et à mesure.

U6-S1-M2

module

Élève

document

MaisonDesJeux

scénario 1

—

format et durée

Ce qu'il faut savoir avant de commencer	4
Comment lancer un programme Python ?	4
Que rend input() ?	4
Que rend une comparaison ?	4
Comment afficher un nombre proprement ?	5
Que donnent // et % ?	5
Vérifiez-vous	5
Structures de contrôle	6
Capsule 1 sur 4 Choisir un chemin : if, elif, else	6
Capsule 2 sur 4 Combiner des conditions : and, or, not	8
Capsule 3 sur 4 Répéter un nombre connu de fois : for	10
Capsule 4 sur 4 Répéter tant qu'il le faut : while	12
Ce qu'on retient	15
Structures de contrôle	16
L'essentiel	16
Les mots à connaître	16
La méthode pour tracer une boucle	16
Un exemple concret	17
Les pièges	17
Quinze exercices et un outil de tournoi	18
PARTIE A — Choisir	19
PARTIE B — Répéter avec for	23
PARTIE C — Répéter avec while , réparer	26
PARTIE D — L'outil de tournoi	31
POUR ALLER PLUS LOIN	34
CE QU'IL FAUT RETENIR	34
Calcul des cotisations	35
CE QUE VOUS RENDEZ	35
PARTIE A — Lire les règles de Marc	37
PARTIE B — Écrire le programme	39

PARTIE C — Les cas limites	41
PARTIE D — Commenter pour Bruno	42
Tous les documents du module	45
A1 — Le dossier de travail du labo	45
A2 — Les quinze énoncés	46
A3 — Les trois boucles à tracer	51
A4 — Les quatre programmes en panne	52
A5 — Les règles de constitution des équipes de tournoi	53
A6 — Extrait de la documentation de Python	55
A7 — Le point de départ du TP	57
A8 — Les règles de cotisation, écrites par Marc Oliveira	59
A9 — Les vingt foyers de test	60
A10 — Le gabarit du jeu de tests	61

Ce qu'il faut savoir avant de commencer

Ce qu'il faut savoir avant de commencer. Lisez-le si vous avez un doute. Environ 10 minutes.

Comment lancer un programme Python ?

On écrit le programme dans un fichier `.py`, dans VS Code. On l'enregistre avec Ctrl+S. Puis on tape dans le terminal PowerShell :

```
python ex01.py
```

Un fichier non enregistré n'est pas celui que Python lit.

Que rend `input()` ?

`input()` pose une question et attend la réponse au clavier. Il rend **toujours du texte**, même si on tape des chiffres.

Pour calculer, on convertit : `int()` donne un nombre entier.

```
age = int(input("Votre âge : "))
```

Que rend une comparaison ?

Une comparaison rend `True`, vrai, ou `False`, faux.

Écriture	Se lit	Exemple	Résultat
<code>==</code>	égal à	<code>16 == 18</code>	<code>False</code>
<code>!=</code>	différent de	<code>16 != 18</code>	<code>True</code>
<code><</code> , <code><=</code>	plus petit, plus petit ou égal	<code>18 <= 18</code>	<code>True</code>
<code>></code> , <code>>=</code>	plus grand, plus grand ou égal	<code>16 > 18</code>	<code>False</code>

Attention : `=` range une valeur dans une variable. `==` compare deux valeurs.

Comment afficher un nombre proprement ?

Avec une chaîne formatée : un `f` devant les guillemets, et la variable entre accolades. `:.2f` affiche deux chiffres après le point.

```
prix = 12.5
print(f"Cotisation : {prix:.2f} €")
```

Affiche : `Cotisation : 12.50 €`.

Que donnent `//` et `%` ?

`//` donne le quotient de la division entière : combien de fois. `%` donne le reste.

```
17 // 4   donne   4
17 % 4    donne   1
```

Vérifiez-vous

- ☐ 1. Léa tape `17` à la question `input("Âge : ")`. Quel est le type de la valeur reçue ?
- ☐ 2. Que vaut `18 >= 18` ?
- ☐ 3. Que donnent `10 // 3` et `10 % 3` ?

Réponses

- 1. Du texte, un `str` : `"17"`. Il faut `int()` pour en faire un nombre.
- 2. `True` : 18 est égal à 18, donc « plus grand ou égal » est vrai.
- 3. `3` et `1` : 3 fois 3 font 9, il reste 1.

Structures de contrôle

COURS-U6-S1-02

Léa Fournier est animatrice bénévole à MaisonDesJeux, une association de jeux de société. Samedi, c'est le tournoi. Dix-sept joueurs sont inscrits.

À l'entrée, Léa encaisse les cotisations à la main. Elle fait payer 25 € à un garçon de 17 ans : il aurait dû payer 12 €.

Ensuite, elle forme les équipes sur une feuille. Elle oublie un joueur.

Léa soupire : « Un programme, ça ne se tromperait pas. »

« Mais comment il saurait quel prix choisir ? Et comment il ferait pour les dix-sept joueurs ? »

À la fin de la séance, vous saurez écrire un programme qui choisit et qui répète. Vous saurez aussi suivre une boucle à la main, avant de la lancer.

LE PLAN DE LA SÉANCE

- 1 Choisir un chemin : `if`, `elif`, `else`.
- 2 Combiner des conditions : `and`, `or`, `not`.
- 3 Répéter un nombre connu de fois : `for`.
- 4 Répéter tant qu'il le faut : `while`.

LA QUESTION DE DÉPART

À la caisse du cinéma, le prix dépend de vous. Comment le caissier choisit-il le bon prix ?

CAPSULE 1 SUR 4

Choisir un chemin : `if`, `elif`, `else`

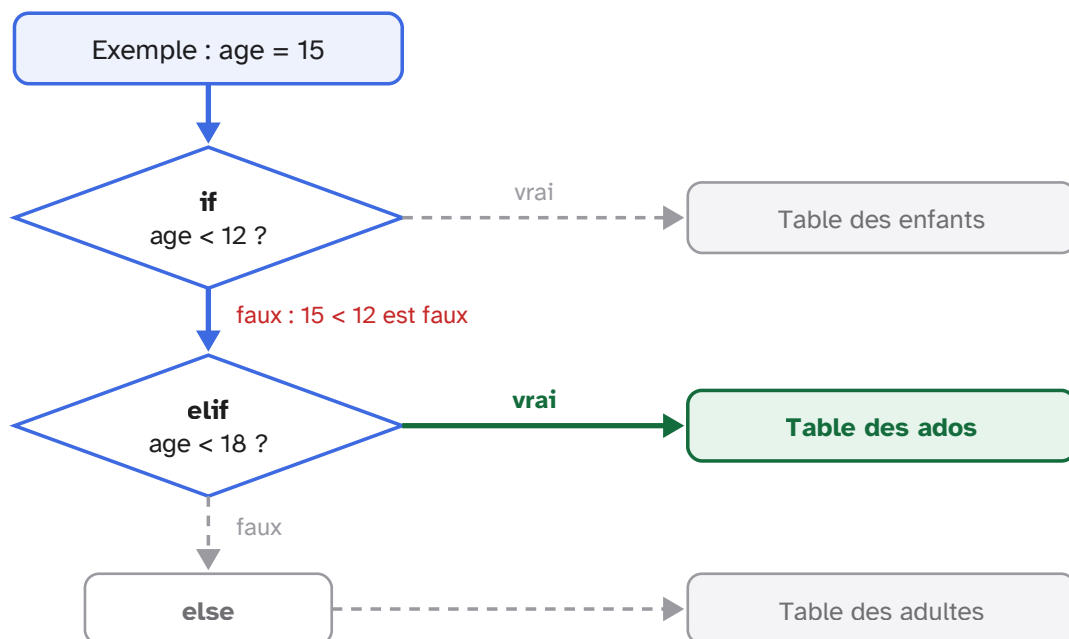
Jusqu'ici, un programme exécute ses lignes une par une, de haut en bas. Toutes, à chaque fois. Comment lui faire choisir entre deux prix ?

On lui donne une **CONDITION** : une question dont la réponse est vrai ou faux. Si c'est vrai, Python fait un bloc de lignes. Sinon, il en fait un autre, ou rien.

C'est comme le caissier du cinéma. Il ne fait pas tout : il choisit selon votre réponse. Le programme de Léa doit choisir le prix de la même façon.

- 1 On écrit `if`, la condition, puis deux-points.
- 2 Les lignes à faire si c'est vrai sont **décalées de quatre espaces**. Ce décalage forme un **BLOC**.
- 3 `elif`, pour « sinon, si », pose une autre condition. On peut en mettre plusieurs.
- 4 `else`, pour « sinon », prend tous les cas qui restent. Il n'a pas de condition.
- 5 Python teste **de haut en bas**. Il fait le bloc de la **première** condition vraie, puis il saute tout le reste.

```
if age < 12:
    print("Table des enfants")
elif age < 18:
    print("Table des ados")
else:
    print("Table des adultes")
```



Python prend le premier « vrai », puis saute tout le reste.

Schéma • Python teste les conditions de haut en bas et prend le premier chemin vrai

On peut aussi mettre un `if` dans le bloc d'un autre `if`. On dit que les conditions sont **imbriquées**. La seconde question n'est posée que si la première a répondu oui.

ON LE FAIT ENSEMBLE — L'ENTRÉE DU TOURNOI

Léa veut afficher le prix d'entrée selon l'âge : moins de 18 ans, 12 € ; sinon, 25 €.

1. Quelle condition écrit-on après `if` ?
2. Que met-on dans le bloc du `if` ?
3. Faut-il un `elif` ?
4. Que s'affiche-t-il pour 18 ans ?

Exemple

Pourquoi

✓	<code>if age < 12: puis elif age < 18:</code>	Les cas les plus précis d'abord : chaque âge arrive au bon bloc.
✗	<code>if age >= 12: avant elif age >= 18:</code>	Un adulte de 30 ans a aussi 12 ans ou plus. Il prend le premier bloc. La table des adultes n'est jamais atteinte.
✗	<code>print("Table des ados")</code> collé à gauche, sous le <code>if</code>	Sans le décalage, la ligne n'est pas dans le bloc. Python affiche une <code>IndentationError</code> .

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Avec le programme des tables ci-dessus, qu'affiche-t-il pour 12 ans ?
2. Pourquoi `else` n'a-t-il jamais de condition ?

CAPSULE 2 SUR 4

Combiner des conditions : and, or, not

On sait poser une question à la fois. Mais Catane se joue de 3 à 4 joueurs : il faut deux questions en même temps. Comment les relier ?

On utilise un **OPÉRATEUR LOGIQUE**. Il relie deux conditions et donne une seule réponse, vrai ou faux. On obtient une **condition composée**.

Pensez à l'entrée d'un concert interdit aux mineurs : « 18 ans **et** un billet ». Il faut les deux. À la cantine, c'est « une carte **ou** un ticket » : un des deux suffit.

Opérateur	Vrai quand...	Exemple vrai
<code>and</code>	les deux conditions sont vraies	<code>joueurs >= 3 and joueurs <= 4</code> avec 3
<code>or</code>	au moins une condition est vraie	<code>jour == "mercredi" or jour == "samedi"</code> avec samedi
<code>not</code>	la condition est fausse	<code>not a_jour</code> quand <code>a_jour</code> vaut <code>False</code>

L'**ordre d'évaluation** compte aussi. Python calcule `not` d'abord, puis `and`, puis `or`. En cas de doute, on met des parenthèses.

`joueurs >= 3 and joueurs <= 4`



and : les deux conditions doivent être vraies. C'est juste.

`joueurs >= 3 or joueurs <= 4`



or : une seule condition suffit. Ici, elle est toujours vraie : c'est une erreur.

Schéma • Avec `and`, le nombre doit être dans l'intervalle ; avec `or`, une seule condition suffit

ON LE FAIT ENSEMBLE — L'EMPRUNT D'UN JEU

Un adhérent emprunte un jeu s'il est à jour de cotisation et s'il a moins de 2 jeux chez lui.

1. Quelles sont les deux conditions ?
2. Faut-il les deux, ou une seule ?
3. Écrivez la ligne du `if`.
4. Que donne-t-elle pour « oui » et 2 jeux ?

Exemple	Pourquoi
✓ <code>if joueurs >= 3 and joueurs <= 4:</code>	Les deux limites doivent être respectées : c'est <code>and</code> .
✗ <code>if joueurs >= 3 or joueurs <= 4:</code>	Tout nombre est soit plus grand que 3, soit plus petit que 4. La condition est toujours vraie.
✗ <code>if jour == "mercredi" or "samedi":</code>	Python lit <code>"samedi"</code> seul, qui compte comme vrai. Il faut répéter : <code>or jour == "samedi"</code> .

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Le local est fermé le lundi et le mardi. Écrivez la condition « le jour est lundi ou mardi ».
2. Avec `age = 20`, que vaut `age >= 18 and age < 26` ?

CAPSULE 3 SUR 4

Répéter un nombre connu de fois : for

Le programme sait maintenant choisir un prix. Mais Léa a dix-sept joueurs. Faut-il écrire dix-sept fois la même chose ?

Non : on écrit une **BOUCLE BORNÉE**. C'est une boucle qui fait un nombre de tours connu d'avance. En Python, elle s'écrit avec `for`.

Chaque tour s'appelle une **ITÉRATION**.

C'est comme distribuer les cartes à quatre joueurs. On sait d'avance qu'on fait le tour de la table quatre fois.

- 1 On écrit `for`, un nom de variable, `in`, puis la séquence à parcourir.
- 2 `range(1, 5)` donne les nombres 1, 2, 3, 4. **Le dernier nombre n'est jamais donné.**
- 3 On peut aussi parcourir un texte, lettre par lettre, ou une suite de valeurs entre crochets : `[12, 25, 55]`.
- 4 À chaque tour, la variable prend **la valeur suivante**. Puis Python fait le bloc décalé.

Avant de lancer une boucle, on la **TRACE** : on la fait à la main, tour par tour, dans un tableau.
Une ligne par tour, une colonne par variable.

```
total = 0
for n in range(1, 4):
    total = total + n
```

```
for n in range(1, 4):
    total = total + n
```

range(1, 4) donne : 1 2 3 4 jamais donné

Le tableau de traçage : une ligne par tour, une colonne par variable.

Tour	n	total
départ	-	0
1	1	0 + 1 = 1
2	2	1 + 2 = 3
3	3	3 + 3 = 6

fin : total = 6

Schéma • À chaque tour, la variable prend la valeur suivante, puis le bloc s'exécute

ON LE FAIT ENSEMBLE — TRACER LES COTISATIONS DE MERCREDI

`total = 0`, puis `for prix in [12, 25, 55]:` et, dans le bloc,
`total = total + prix.`

1. Combien de tours fait la boucle ?
2. Au tour 1, que valent `prix` et `total` ?
3. Au tour 2 ?
4. Au tour 3, puis à la fin ?

Exemple

Pourquoi

✓ `for table in range(1, 7):` pour les tables 1 à 6 range s'arrête juste avant 7 : on a bien 1 à 6.

Exemple	Pourquoi
✗ <code>for poste in range(1, 12):</code> pour les 12 postes	La boucle s'arrête à 11. Il manque un tour : c'est une erreur de borne.
✗ <code>total = 0</code> écrit dans le bloc de la boucle	Le total repart de 0 à chaque tour. À la fin, il ne contient que la dernière valeur.

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Quels nombres donne `range(2, 6)` ?
2. Tracez la boucle de « Comment ça marche » : que vaut `total` à la fin ?

CAPSULE 4 SUR 4

Répéter tant qu'il le faut : while

La boucle `for` sait combien de tours faire. Mais Léa ne sait pas combien de joueurs vont arriver ce soir. Comment répéter sans savoir combien de fois ?

On écrit une **BOUCLE CONDITIONNELLE**. Elle répète son bloc **tant qu'une condition est vraie**. En Python, elle s'écrit avec `while`.

C'est comme remplir un verre d'eau : on verse tant qu'il n'est pas plein. On ne compte pas les gorgées. On regarde le niveau à chaque fois.

- 1 Python **teste la condition**. Si elle est fausse dès le début, le bloc ne tourne jamais.
- 2 Si elle est vraie, il fait le bloc, puis **revient tester** la condition.
- 3 Dans le bloc, quelque chose **doit changer** la condition. Sinon, elle reste vraie pour toujours : c'est une **BOUCLE INFINIE**. On l'arrête avec Ctrl+C.
- 4 `break` sort de la boucle tout de suite. C'est une sortie anticipée.
- 5 `continue` saute la fin du tour et passe au tour suivant.

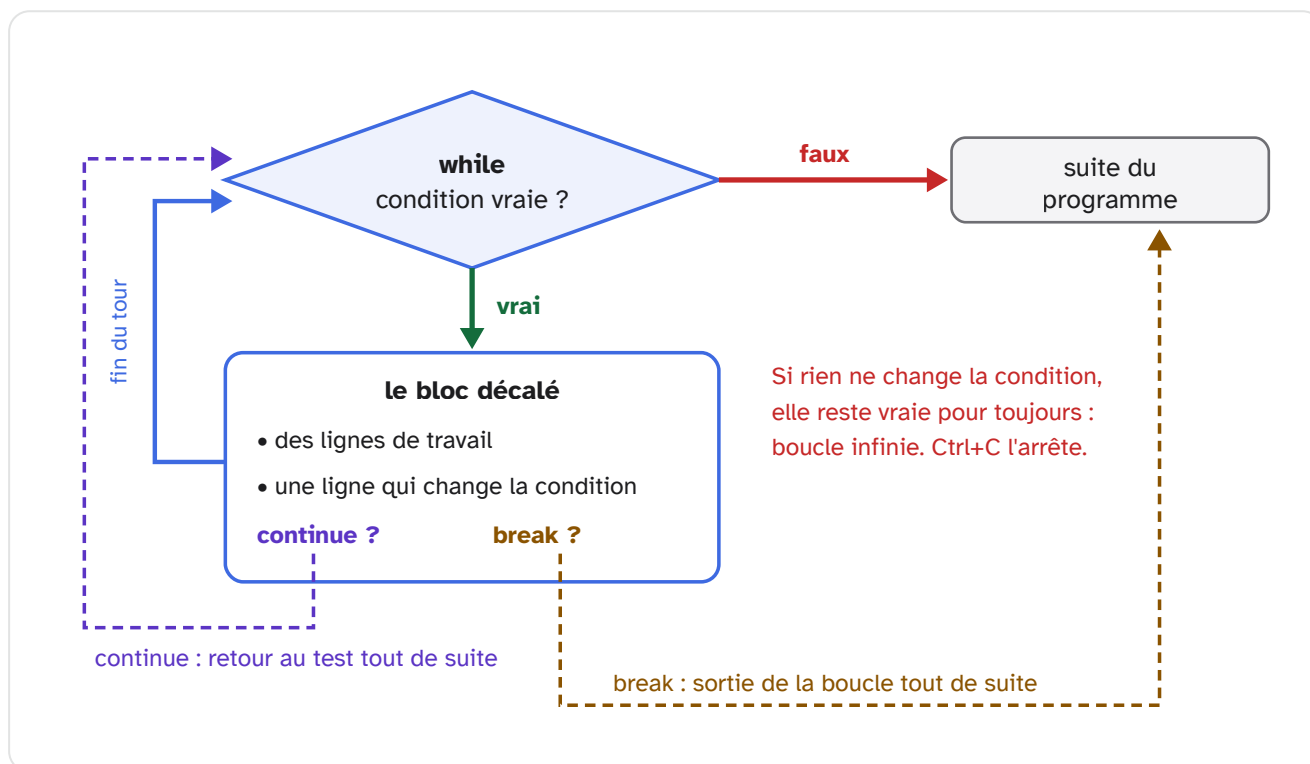


Schéma • La boucle while teste sa condition avant chaque tour ; break sort, continue repart au test

Comment choisir ? Vous savez combien de tours faire, ou vous avez une séquence à parcourir : `for` . Vous ne le savez pas, vous attendez qu'une chose arrive : `while` .

INSTANT HISTOIRE — 2008 : LES LECTEURS DE MUSIQUE BLOQUÉS LE 31 DÉCEMBRE

Le 31 décembre 2008, environ un million de lecteurs de musique Zune, de Microsoft, se bloquent tous en même temps.

2008 compte 366 jours. Ce jour-là, une boucle `while` qui calculait la date ne pouvait plus avancer. Sa condition restait vraie, et rien ne la changeait.

Le lendemain, tout remarche tout seul.

Selon vous, pourquoi personne n'avait vu le problème avant la vente ?

ON LE FAIT ENSEMBLE — RANGER 10 JEUX DANS DES CAISSES DE 3

`jeux = 10`, `caisses = 0`, puis `while jeux > 0`: et, dans le bloc, `jeux = jeux - 3` et `caisses = caisses + 1`.

1. Avant le premier tour, la condition est-elle vraie ?
2. Après le tour 1 ?
3. Après les tours 2 et 3 ?
4. Que se passe-t-il ensuite ?

Exemple

Pourquoi

✓ <code>while code != "MDJ2026":</code> avec un <code>input()</code> dans le bloc	La saisie change <code>code</code> à chaque tour : la boucle finira par s'arrêter.
✗ <code>while table <= 5:</code> sans <code>table = table + 1</code> dans le bloc	<code>table</code> ne change jamais : la condition reste vraie. Boucle infinie.
✗ <code>break</code> pour sauter la table 7	<code>break</code> arrête toute la boucle : les tables 8, 9, 10 ne sont jamais affichées. Il faut <code>continue</code> .

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Afficher les tables 1 à 10 : `for` ou `while` ? Et demander un âge jusqu'à ce qu'il soit possible ?
2. Que fait `break` dans une boucle ?

Ce qu'on retient

SI VOUS NE DEVEZ RETENIR QUE TROIS CHOSES

1. Écrivez les conditions de la plus précise à la plus large : Python prend la première qui est vraie.
2. Nombre de tours connu, `for` ; nombre inconnu, `while`, avec une ligne qui change la condition.
3. Tracez une boucle à la main, tour par tour, avant de la lancer.

Revenons à Léa. Pour le prix, une condition `if age < 18` choisit 12 € ou 25 €, sans erreur. Pour les dix-sept joueurs, une boucle `for` fait les équipes, et personne n'est oublié.

Léa n'a plus qu'à taper le nombre de joueurs. Le programme fait le reste, et le refait à chaque tournoi.

Structures de contrôle

COURS-U6-S1-02

L'essentiel

- **Condition** : une question vrai ou faux. `if` fait un bloc si elle est vraie.
- `if`, `elif`, `else` : Python teste de haut en bas et fait le bloc de la première condition vraie.
- **Condition composée** : `and` (les deux), `or` (au moins une), `not` (le contraire).
- **Boucle** `for` : un nombre de tours connu, ou une séquence à parcourir.
- **Boucle** `while` : on répète tant que la condition est vraie.

Les mots à connaître

Mot	En clair
Bloc	Les lignes décalées de 4 espaces sous un <code>if</code> , un <code>for</code> ou un <code>while</code> .
Imbriquées	Un <code>if</code> placé dans le bloc d'un autre <code>if</code> .
Itération	Un tour de boucle.
<code>range(a, b)</code>	Les nombres de <code>a</code> à <code>b - 1</code> .
Tracer	Faire la boucle à la main, une ligne de tableau par tour.
Boucle infinie	Un <code>while</code> dont la condition reste toujours vraie. Ctrl+C l'arrête.
<code>break</code>	Sort tout de suite de la boucle.
<code>continue</code>	Passe tout de suite au tour suivant.

La méthode pour tracer une boucle

1. **Dessinez** un tableau : une colonne « tour », une colonne par variable.

2. **Écrivez** la valeur de départ de chaque variable.
3. **Faites** le bloc à la main : une ligne par tour.
4. **Vérifiez** la condition, ou la valeur suivante, avant chaque tour.
5. **Lancez** le programme, puis comparez avec votre tableau.

Un exemple concret

Compter les joueurs de 12 ans ou plus.

```
joueurs = 0
for age in [16, 9, 42]:
    if age >= 12:
        joueurs = joueurs + 1
```

Tour	age	age >= 12 ?	joueurs
départ			0
1	16	vrai	1
2	9	faux	1
3	42	vrai	2

Conclusion : à la fin, `joueurs` vaut 2.

Les pièges

- **×** `if age >= 12` **avant** `elif age >= 18` → mettez le cas le plus précis en premier.
- **×** `or` **à la place de** `and` pour un intervalle → « de 3 à 4 », c'est `and`.
- **×** `range(1, 12)` **pour 12 tours** → écrivez `range(1, 13)`.
- **×** Un `while` **sans ligne qui change la condition** → boucle infinie.
- **×** `break` **à la place de** `continue` → `break` arrête toute la boucle.

Quinze exercices et un outil de tournoi

LABO-U6-S1-02

Contexte — Vous êtes prestataire bénévole chez MaisonDesJeux, une association de jeux de société. Le jour du tournoi, Léa Fournier, animatrice, forme les équipes à la main. Une erreur arrive vite : un joueur oublié, une équipe trop petite.

Vous allez lui écrire un outil qui forme les équipes sans erreur. Avant, vous vous entraînez sur quinze exercices courts, puis vous réparez quatre programmes de Bruno Sanchez, un autre bénévole.

RÈGLE 1A DE LA SÉANCE

Scénario 1 : pas d'IA pendant ce labo. Vous avez les annexes, la documentation de Python et le web classique.

Si vous avez un doute, demandez au formateur ou à votre binôme.

À la fin de ce labo, vous saurez écrire des conditions et des boucles, et suivre une boucle à la main. Vous saurez aussi trouver pourquoi un programme fait autre chose que prévu.

Comment ça se valide — à chaque point de contrôle, vous montrez votre résultat au formateur. Il valide, puis vous passez à la suite.

Les mots de la séance

Mot	Ce que ça veut dire
Condition	Une question vrai ou faux, après <code>if</code> , <code>elif</code> ou <code>while</code> .
Bloc	Les lignes décalées de 4 espaces sous un <code>if</code> , un <code>for</code> ou un <code>while</code> .
Opérateur logique	<code>and</code> , <code>or</code> , <code>not</code> : ils relient deux conditions.
Boucle <code>for</code>	Une boucle qui fait un nombre de tours connu, ou parcourt une séquence.
Boucle <code>while</code>	Une boucle qui tourne tant que sa condition est vraie.
Tracer	Faire une boucle à la main, une ligne de tableau par tour, avant de la lancer.
Erreur de logique	Le programme tourne sans message d'erreur, mais il ne fait pas ce qu'on voulait.

Ce dont vous avez besoin

- Votre poste Windows, avec Python, VS Code et PowerShell.
- [Annexe A1](#) : le dossier de travail, dans le fichier [A1_conditions_boucles.zip](#).
- [Annexe A2](#) : les quinze énoncés.
- [Annexe A3](#) : les trois boucles à tracer.
- [Annexe A4](#) : les quatre programmes en panne.
- [Annexe A5](#) : les règles des équipes de tournoi.
- [Annexe A6](#) : un extrait de la documentation de Python.
- Une feuille et un stylo, pour tracer les boucles.

En binôme — le pilote tape, le lecteur lit les énoncés et les annexes à voix haute, et vérifie chaque résultat. On inverse à chaque partie.

Le plan du labo

Partie	Étapes	Ce que vous obtenez	Temps
A — Choisir	1 à 6	les exercices 1 à 5	25 min
B — Répéter avec <code>for</code>	7 à 12	les exercices 6 à 10, deux boucles tracées	30 min
C — Répéter avec <code>while</code> , réparer	13 à 20	les exercices 11 à 15, une boucle tracée, quatre pannes réparées	35 min
D — L'outil de tournoi	21 à 24	un outil vérifié de 0 à 30 participants	30 min

PARTIE A — Choisir

Étapes 1 à 6, environ 25 min. À la fin, les exercices 1 à 5 tournent.

Besoin d'un rappel ? Revoir le cours : « [Choisir un chemin : if, elif, else](#) ».

01 Installer le dossier de travail

☐ Téléchargez `A1_conditions_boucles.zip` depuis l'[annexe A1](#).

☐ Faites un clic droit sur le fichier, puis « Extraire tout ». Avec « Parcourir », choisissez votre dossier « Documents ».

Vous devez voir : un dossier `conditions_boucles` dans « Documents ». Il contient `ex01.py` à `ex15.py`, `tournoi.py`, `verifier_equipes.py` et deux dossiers, `pannes` et `traces`.

Si vous voyez un dossier `conditions_boucles` dans un autre dossier `conditions_boucles` : utilisez celui qui contient directement `ex01.py`.

02 Ouvrir le dossier dans VS Code

☐ Ouvrez VS Code. Menu « Fichier », puis « Ouvrir le dossier ». Choisissez `Documents\conditions_boucles`.

☐ Ouvrez un terminal : menu « Terminal », puis « Nouveau terminal ». Tapez `python --version`.

Vous devez voir : `Python 3.` suivi d'un numéro. L'invite du terminal finit par `conditions_boucles>`.

Si vous voyez « python n'est pas reconnu », ou une fenêtre du Microsoft Store : tapez `py` à la place de `python` pendant tout le labo.

03 Exercice 1, fait avec vous

Voici le premier exercice, fait en entier. Les autres se font de la même façon : énoncé dans l'[annexe A2](#), code dans le fichier, essais.

- ☐ Ouvrez `ex01.py`. Sous les deux lignes de commentaire, tapez :

```
age = int(input("Votre âge : "))
if age < 18:
    print("Tarif jeune")
else:
    print("Tarif adulte")
```

- ☐ Enregistrez avec Ctrl+S. Lancez `python ex01.py` deux fois : tapez 17, puis 18.

Vous devez voir : `Tarif jeune` avec 17, puis `Tarif adulte` avec 18.

Si vous voyez `IndentationError` : une ligne `print` n'est pas décalée. VS Code décale tout seul après les deux-points. Sinon, mettez quatre espaces devant.

04 Exercices 2 et 3 : and et or

- ☐ Écrivez l'exercice 2 dans `ex02.py`. Faites les quatre essais de l'[annexe A2](#).

- ☐ Écrivez l'exercice 3 dans `ex03.py`. Faites les deux essais.

Vous devez voir : « possible » seulement pour 3 et 4 joueurs. `Le local est ouvert` pour samedi, `Le local est fermé` pour lundi.

05 Exercice 4 : if , elif , else

☐ Écrivez l'exercice 4 dans `ex04.py` . Faites les quatre essais : 11, 12, 17, 18.

Vous devez voir : `Table des enfants` , `Table des ados` , `Table des ados` , `Table des adultes` .

Pourquoi ? Vous avez écrit `elif age < 18` . Pourquoi n'avez-vous pas besoin d'écrire aussi `age >= 12` dans cette condition ?

Votre réponse :

06 Exercice 5 : deux conditions à la fois

☐ Écrivez l'exercice 5 dans `ex05.py` . Faites les trois essais : oui et 1, oui et 2, non et 0.

Vous devez voir : `Emprunt accepté` pour le premier essai, `Emprunt refusé` pour les deux autres.

POINT DE CONTRÔLE A

☐ Montrez au formateur les exercices 1 à 5. Répondez à sa question sur l'un d'eux.

☐ Le formateur a validé. Inversez les rôles, puis passez à la partie B.

PARTIE B — Répéter avec `for`

Étapes 7 à 12, environ 30 min. À la fin, les exercices 6 à 10 tournent, et vous avez tracé deux boucles.

Besoin d'un rappel ? Revoir le cours : [« Répéter un nombre connu de fois : `for` »](#).

07 Tracer la boucle 1, faite avec vous

On trace une boucle **avant** de la lancer. Voici la première, faite en entier.

☐ Lisez la boucle 1 dans l'[annexe A3](#). Suivez le tableau ligne par ligne :

Tour	<code>n</code>	<code>total</code> à la fin du tour
------	----------------	-------------------------------------

départ		0
--------	--	---

1	1	$0 + 1 = 1$
---	---	-------------

2	2	$1 + 2 = 3$
---	---	-------------

3	3	$3 + 3 = 6$
---	---	-------------

4	4	$6 + 4 = 10$
---	---	--------------

☐ Lancez `python traces\trace1.py`.

Vous devez voir : `Total : 10`, comme la dernière ligne du tableau. `range(1, 5)` s'arrête à 4 : il y a quatre tours.

08 Trouver dans la doc

☐ Ouvrez l'[annexe A6](#), partie « `range` : une suite de nombres ».

☐ Notez les nombres donnés par `range(1, 7)` : _____

☐ Notez les nombres donnés par `range(10, 0, -1)` : _____

Vous devez avoir : deux suites. Aucune ne contient le nombre écrit en deuxième dans `range`.

09 Exercices 6 et 7 : range

☐ Écrivez l'exercice 6 dans `ex06.py`, avec la première suite de l'étape 8. Lancez-le.

☐ Écrivez l'exercice 7 dans `ex07.py`, avec la seconde suite. Lancez-le.

Vous devez voir : `Table 1` à `Table 6`. Puis les nombres de 10 à 1, et `C'est parti !`.

10 Exercices 8 et 9 : parcourir une séquence

☐ Écrivez l'exercice 8 dans `ex08.py`. Lancez-le.

☐ Écrivez l'exercice 9 dans `ex09.py`. Lancez-le.

Vous devez voir : les cinq lettres de DIXIT, puis `Nombre de I : 2`. Puis `Total encaissé : 129 €`.

Si vous voyez `Total encaissé : 55 €` : la ligne `total = 0` est dans le bloc de la boucle. Placez-la avant le `for`.

11 Exercice 10 : une saisie à chaque tour

☐ Écrivez l'exercice 10 dans `ex10.py`. Lancez-le avec 3 tables : 4, 4 et 5 joueurs.

Vous devez voir : Joueurs en tout : 13.

Pourquoi ? Le nombre de tables est saisi au clavier. Pourquoi est-ce quand même une boucle `for`, avec un nombre de tours connu ?

Votre réponse :

12 Tracer la boucle 2

☐ Lisez la boucle 2 dans l'[annexe A3](#). Remplissez le tableau à la main, **sans lancer le programme**.

Tour	age	age >= 12 ?	joueurs à la fin du tour
------	-----	-------------	--------------------------

départ			0
--------	--	--	---

1			
---	--	--	--

2			
---	--	--	--

3			
---	--	--	--

4			
---	--	--	--

Tour age age >= 12 ? joueurs à la fin du tour

5

☐ Lancez `python traces\trace2.py`. Comparez avec la dernière ligne de votre tableau.

Vous devez voir : le même nombre que dans votre tableau. Sinon, cherchez le tour où vous vous êtes trompé.

POINT DE CONTRÔLE B

☐ Montrez au formateur votre tableau de l'étape 12 et les exercices 6 à 10.

☐ Le formateur a validé. Inversez les rôles, puis passez à la partie C.

PARTIE C — Répéter avec `while`, réparer

Étapes 13 à 20, environ 35 min. À la fin, les exercices 11 à 15 tournent, et les quatre programmes de Bruno font ce qu'ils doivent.

Besoin d'un rappel ? Revoir le cours : [« Répéter tant qu'il le faut : while »](#).

13 Tracer la boucle 3

☐ Lisez la boucle 3 dans l'[annexe A3](#). Remplissez le tableau à la main, sans lancer le programme. Ajoutez des lignes si besoin.

Tour	jeux > 0 ? avant le tour	jeux à la fin du tour	caisses à la fin du tour
départ		10	0

Tour `jeux > 0` ? avant le tour `jeux` à la fin du tour `caisses` à la fin du tour

1

2

3

4

☐ Lancez `python traces\trace3.py`. Comparez.

Vous devez voir : la même ligne que la fin de votre tableau.

Pourquoi ? À la fin, `jeux` est négatif. Pourquoi la boucle a-t-elle fait un tour de plus quand il restait 1 jeu ?

Votre réponse :

14 Exercices 11 et 12 : redemander

☐ Écrivez l'exercice 11 dans `ex11.py`. Essai : -3, puis 150, puis 17.

☐ Écrivez l'exercice 12 dans `ex12.py`. Essai : `mdj`, puis `1234`, puis `MDJ2026`.

Vous devez voir : `Âge enregistré : 17`. Puis `Coffre ouvert en 3 essais`.

15 Exercices 13 et 14 : compter les tours, sortir avec break

- ☐ Écrivez l'exercice 13 dans `ex13.py` . Lancez-le.
 - ☐ Dans l'[annexe A6](#), partie « `break` et `continue` », lisez ce que fait `break` . Écrivez l'exercice 14. Essai : Léa, Bruno, Nadia, puis fin.
-

Vous devez voir : `Il faut 9 semaines . Puis Joueurs inscrits : 3 .`

16 Exercice 15 : sauter un tour avec continue

- ☐ Écrivez l'exercice 15 dans `ex15.py` . Lancez-le.
 - ☐ Remplacez `continue` par `break` . Relancez. Notez ce qui change :

 - ☐ Remettez `continue` .
-

Vous devez voir : neuf lignes avec `continue` , sans la table 7. Seulement six lignes avec `break` .

17 Panne 1, faite avec vous

Les quatre programmes de Bruno n'affichent aucun message d'erreur. Leur code et la liste des natures de défaut sont dans l'[annexe A4](#).

- ☐ Lancez `python pannes\panne1.py` . Le terminal se remplit sans fin. Appuyez sur Ctrl+C pour l'arrêter.
-

Vous devez voir : des lignes `Table 1 prête` à l'infini, puis `KeyboardInterrupt` après Ctrl+C.

- ☐ Lisez la ligne d'exemple de la grille. Elle est faite pour vous.

Panne	Ce que j'observe	Nature du défaut	Ma correction
1	<code>Table 1 prête</code> s'affiche sans fin	Boucle infinie	ajouter <code>table = table + 1</code> dans le bloc du <code>while</code>
2	_____	_____	_____
3	_____	_____	_____
4	_____	_____	_____

- ☐ Corrigez `pannes\panne1.py`, enregistrez, relancez.

Vous devez voir : `Table 1 prête` à `Table 5 prête`, puis `Toutes les tables sont prêtes.`

18 Panne 2

- ☐ Lancez `python pannes\panne2.py` avec 30, puis avec 8. Notez ce que vous observez dans la grille.
- ☐ Choisissez la nature du défaut dans l'[annexe A4](#). Corrigez, puis relancez avec 30, 15 et 8.

Vous devez voir : `Table des adultes`, `Table des ados`, `Table des enfants`.

19 Panne 3

☐

Lancez `python pannes\panne3.py` . Comptez les postes affichés. Remplissez la ligne 3 de la grille.

☐ Corrigez, puis relancez.

Vous devez voir : Poste numéro 1 à Poste numéro 12 , puis Fin de la liste.

20 Panne 4

☐ Lancez `python pannes\panne4.py` avec 3, puis avec 6, puis avec 1. Remplissez la ligne 4 de la grille.

☐ Corrigez, puis relancez avec 6.

Vous devez voir : Pas de partie de Catane pour 6 et pour 1, Partie de Catane possible pour 3.

Pourquoi ? Aucune des quatre pannes n'affichait de message d'erreur. Pourquoi Python ne les a-t-il pas vues ?

Votre réponse :

POINT DE CONTRÔLE C

☐ Montrez au formateur votre tableau de l'étape 13, votre grille des pannes et les quatre programmes réparés.

☐ Le formateur a validé. Inversez les rôles, puis passez à la partie D.

PARTIE D — L'outil de tournoi

Étapes 21 à 24, environ 30 min. À la fin, `tournoi.py` forme les équipes, et l'outil de vérification ne trouve aucun problème.

21 Faire les équipes à la main

☐ Lisez les règles de l'[annexe A5](#).

☐ Pour 10 participants, remplissez la grille. Pour chaque numéro, `i % nb` donne l'équipe du calcul.

`n` `nb = (n + 2) // 4` Équipe 1, `i % nb` vaut 0 Équipe 2, `i % nb` vaut 1 Équipe 3, `i % nb` vaut 2

10 _____

Vous devez avoir : trois équipes, de 3 ou 4 membres. Chaque numéro de 1 à 10 apparaît une seule fois.

22 Le début de l'outil : refuser ou compter les équipes

☐ Dans `tournoi.py`, écrivez la saisie de `n`. Si `n` est plus petit que 3, affichez `Tournoi impossible : il faut au moins 3 participants.`

☐ Sinon, calculez `nb` et affichez une ligne comme `10 participants, 3 équipes`. Lancez avec 2, puis avec 10.

Vous devez voir : le message de refus pour 2. Pour 10, `10 participants, 3 équipes`.

23 La boucle des équipes

L'outil doit afficher chaque équipe sur une seule ligne. Il faut une boucle dans une boucle.

- ☐ Dans l'[annexe A6](#), partie « `print` : afficher sans aller à la ligne », trouvez comment rester sur la même ligne. Notez-le : _____
- ☐ Traduisez ce plan en Python, dans le bloc du `else`. Chaque ligne du plan donne une ligne de code, avec le même décalage.

```
pour chaque équipe e, de 0 à nb - 1 :  
    afficher Équipe, puis e + 1, puis deux-points, sans aller à la ligne  
    pour chaque numéro i, de 1 à n :  
        si i % nb vaut e :  
            afficher i, sans aller à la ligne  
    terminer la ligne de l'équipe
```

Aide : « pour chaque » s'écrit avec `for` et `range`. « Sans aller à la ligne », c'est `end=" "`.
« Terminer la ligne », c'est `print()`.

- ☐ Lancez avec 12.

Vous devez voir : exactement l'affichage de l'[annexe A5](#) pour 12 participants.

Si vous voyez une équipe à qui il manque le dernier numéro : votre `range` s'arrête à `n - 1`. Relisez l'étape 8.

24 Vérifier l'outil

- ☐ Lancez l'outil de vérification :

```
python verifier_equipes.py
```

☐ Corrigez chaque problème signalé, puis relancez, jusqu'à 0 problème.

☐ Lancez votre outil avec 7, 13 et 25. Notez le nombre de membres de chaque équipe :

n	Membres par équipe
---	--------------------

7	
---	--

13	
----	--

25	
----	--

Vous devez voir : 31 cas vérifiés (0 à 30 participants), 0 problème(s).

Pourquoi ? Vous avez essayé votre outil avec 2, 10 et 12 participants. Pourquoi est-ce utile de le lancer aussi de 0 à 30 ?

Votre réponse :

POINT DE CONTRÔLE D

☐ Montrez au formateur `verifier_equipes.py` avec 0 problème, et votre outil lancé avec 17 participants.

☐ Le formateur a validé. Le labo est terminé.

POUR ALLER PLUS LOIN

10 à 15 min, si vous avez fini avant la fin de la séance.

Léa veut aussi savoir combien d'équipes ont 5 membres.

☐

Dans la boucle des équipes, comptez les membres de chaque équipe dans une variable, remise à 0 au début de chaque équipe.

☐

Après la boucle, affichez `Nombre d'équipes de 5 : ...`. Pour 17 participants, vous devez voir `Nombre d'équipes de 5 : 1`.

☐

Relancez `verifier_equipes.py` : il doit toujours afficher 0 problème.

CE QU'IL FAUT RETENIR

1. Les conditions s'écrivent de la plus précise à la plus large : Python prend la première vraie.
2. `range(a, b)` s'arrête à `b - 1` : vérifiez toujours le dernier tour.
3. Tracez une boucle à la main avant de la lancer. Un `while` a toujours une ligne qui change sa condition.

Calcul des cotisations

TP-U6-S1-01

Contexte — Marc Oliveira, le trésorier de MaisonDesJeux, encaisse les cotisations depuis six ans. Il calcule tout de tête, et il se trompe parfois. Le mois dernier, une famille de quatre a payé deux fois le forfait.

Marc vous a écrit ses règles, avec ses mots : « Je préfère que le programme fasse exactement comme moi. » Vous traduisez ses règles en conditions, vous écrivez le programme, puis vous prouvez qu'il marche avec un jeu de tests. Bruno Sanchez reprendra le programme après vous : il doit pouvoir le comprendre sans vous appeler.

RÈGLE IA DE LA SÉANCE

Scénario 1 : pas d'IA pendant ce TP. Vous avez les annexes, la documentation de Python et la recherche web classique.

Notez dans votre mini-rapport chaque coup de pouce demandé.

CE QUE VOUS RENDEZ

Ce que vous rendez	Format	Nom du fichier
Le dossier <code>cotisations</code> , complet, avec <code>journal_tests.txt</code>	ZIP	NOM_Prenom_TP-U6-S1-01_cotisations.zip
Le mini-rapport	PDF	NOM_Prenom_TP-U6-S1-01_rapport.pdf

Le mini-rapport : deux pages au maximum. Il prouve que c'est **votre** travail : chaque rubrique s'appuie sur ce qui s'est passé sur **votre** poste. Un texte général, sans traces de votre poste, ne compte pas.

1. **Mes captures** : les deux captures demandées dans les parties B et C. Prenez-les en plein écran : l'heure de la barre des tâches doit se voir. Une ligne de légende sous chacune.
2. **Mon journal** : faites le parcours imposé de la partie C, étape 6. Recopiez les lignes ÉCHEC de ce lancement, prises dans `journal_tests.txt`. Expliquez chacune en une phrase.

3. **Mes erreurs** : deux erreurs que vous avez vraiment rencontrées pendant le TP, dans le tableau ci-dessous.
4. **Une ligne de mon code** : recopiez la ligne de votre programme qui décide du forfait famille. Expliquez-la avec vos mots, en trois lignes au maximum.
5. **Mes cas limites** : votre grille de la partie C, étape 1.
6. **Ce qui manque** : ce qui ne marche pas, ce que vous avez essayé, et les coups de pouce utilisés.

Voici une ligne d'exemple pour la rubrique 3 :

Ce que j'ai vu, recopié	Ce qui le causait	Ce que j'ai changé
ÉCHEC attendu 12.50 obtenu 25.00 pour le cas 7	J'avais écrit <code>mois > 1</code> : janvier n'avait pas le demi-tarif.	J'ai écrit <code>mois >= 1</code> .

Vos deux erreurs :

Ce que j'ai vu, recopié	Ce qui le causait	Ce que j'ai changé

Comment ce sera évalué

Critère	Ce qui est attendu
Le programme et ses conditions	Les règles de Marc traduites en conditions, dans le bon ordre. Les vingt foyers de référence passent.
Le jeu de tests et les commentaires	Vos propres cas limites, calculés à la main et lancés. Un programme commenté que Bruno comprend seul.

Critère	Ce qui est attendu
Présentation	Document daté, nommé selon la convention, titres clairs, tableaux lisibles

Sans traces de votre poste, c'est-à-dire sans vos captures, votre journal ni vos erreurs, les deux premiers critères ne dépassent pas « Fragile ».

Et à l'oral — le formateur pointera une ligne de votre programme ou de votre mini-rapport. Il vous demandera de l'expliquer, ou de refaire devant lui une erreur que vous racontez.

Ce dont vous disposez

- Votre poste Windows, avec Python, VS Code et PowerShell.
- [Annexe A7](#) : le point de départ du TP. Le dossier se télécharge depuis l'[annexe A7](#), dans le fichier [A7_cotisations.zip](#).
- [Annexe A8](#) : les règles de cotisation, écrites par Marc Oliveira.
- [Annexe A9](#) : les vingt foyers de test.
- [Annexe A10](#) : le gabarit du jeu de tests.

Le plan du TP

Partie	Ce que vous obtenez	Temps conseillé
A — Lire les règles de Marc	les conditions et leur ordre	30 min
B — Écrire le programme	les vingt foyers de référence réussis	45 min
C — Les cas limites	votre jeu de tests, lancé	35 min
D — Commenter pour Bruno	un programme qu'un autre comprend seul	20 min
Mini-rapport		20 min

Bloqué ? Demandez un coup de pouce. Ça ne fait pas perdre de points.

PARTIE A — Lire les règles de Marc

Environ 30 min.

Marc écrit comme il parle. Ses règles ne sont pas dans l'ordre où le programme doit les tester. Avant d'écrire une seule ligne de code, vous les traduisez en conditions.

Besoin d'un rappel ? Revoir le cours : « [Choisir un chemin : if, elif, else](#) ».

☐

1. Lisez l'[annexe A8](#) en entier. Soulignez sur votre feuille chaque phrase qui contient un nombre.

☐

2. Remplissez la grille. Une ligne par règle. La première est faite pour vous montrer.

Règle de Marc, avec ses mots

Condition en Python

Ce qu'elle change

« un jeune, c'est jusqu'à 17 ans compris »

`age < 18`

12 € pour cette personne, sinon 25 €

Règle de Marc, avec ses mots

Condition en Python

Ce qu'elle change

Forfait famille

Demi-tarif

Pour le demi-tarif, utilisez la variable `mois` décrite dans l'[annexe A7](#). Elle vaut 1 pour janvier, 12 pour décembre.

☐

3. Trouvez dans l'[annexe A8](#) les deux phrases qui fixent l'ordre des calculs. Complétez :

Ma règle : on calcule d'abord _____ , puis

☐

4. Lisez les phrases sur le tarif senior et sur les trois enfants. Notez ce que le programme ne doit **pas** faire : _____

☐

5. Calculez à la main, avec vos règles, la cotisation de ces trois foyers de l'[annexe A9](#) : les cas 6, 9 et 17. Notez le calcul en deux lignes au maximum par cas. Comparez avec la colonne « Cotisation attendue ».

Ce qui doit marcher à la fin de la partie A

- ☐ La grille a trois lignes, avec une condition Python pour chaque règle.
- ☐ L'ordre des calculs est écrit en une phrase.
- ☐ Vos trois calculs à la main donnent les cotisations attendues de l'[annexe A9](#).

POINT DE CONTRÔLE A

- ☐ Le formateur a vu votre grille et votre phrase sur l'ordre des calculs. Passez à la partie B.

PARTIE B — Écrire le programme

Environ 45 min.

Le programme lit le foyer, calcule la cotisation avec vos conditions, puis l'affiche. L'outil de tests de l'[annexe A7](#) le lance vingt fois, sur les vingt foyers de référence.

Besoin d'un rappel ? Revoir le cours : [« Répéter un nombre connu de fois : for »](#).

- ☐ 1. Téléchargez [A7_cotisations.zip](#) depuis l'[annexe A7](#). Extrayez-le dans « Documents ». Ouvrez le dossier `Documents\cotisations` dans VS Code.
- ☐ 2. Lisez `cotisation.py` et la partie « Ce que `cotisation.py` doit faire » de l'[annexe A7](#). Notez l'ordre des trois saisies : _____

- ☐ 3. Écrivez la saisie du nombre de personnes. Puis une boucle `for` qui demande l'âge de chaque personne, et additionne son tarif dans une variable.
- ☐ 4. Écrivez le montant de base : le forfait famille, ou le total des tarifs de l'étape 3.
- ☐ 5. Placez les deux lignes déjà écrites, qui lisent la date, après vos saisies d'âges. Puis écrivez le demi-tarif avec la condition de votre grille.
- ☐ 6. Affichez la dernière ligne au format de l'[annexe A7](#), avec deux chiffres après le point.
- ☐ 7. Essayez à la main le cas 9 de l'[annexe A9](#) : `python cotisation.py`, puis 4, 40, 38, 10, 8 et `2026-02-01`.
- ☐ 8. Lancez les vingt foyers de référence. Corrigez jusqu'à « 20 sur 20 ».

```
python lancer_tests.py foyers_reference.csv
```

Ce qui doit marcher à la fin de la partie B

- ☐ L'essai à la main du cas 9 affiche `Cotisation : 27.50 €`.
- ☐ L'outil affiche `Résultat : 20 sur 20 réussis`.
- ☐ Capture 1 ajoutée au mini-rapport : le terminal avec les vingt lignes OK.

POINT DE CONTRÔLE B

- ☐ Le formateur a vu « 20 sur 20 réussis » sur votre écran. Passez à la partie C.

PARTIE C — Les cas limites

Environ 35 min.

Un **cas limite**, c'est une valeur juste à la frontière d'une règle : 17 ans et 18 ans, 2 personnes et 3 personnes. C'est là que les programmes se trompent le plus. Vous trouvez vos propres cas limites, puis vous les testez.

- ☐ 1. Pour chaque règle de votre grille de la partie A, trouvez la frontière. Écrivez la valeur juste avant, et la valeur juste après. La première ligne est faite pour vous montrer.

Règle	Juste avant la frontière	Juste après la frontière
Tarif jeune ou adulte	17 ans : 12 €	18 ans : 25 €

Règle	Juste avant la frontière	Juste après la frontière
Forfait famille		
Demi-tarif, en janvier		
Demi-tarif, en fin de saison		

Pour la fin de saison, relisez la phrase de Marc sur le début et la fin de la saison, dans l'[annexe A8](#).

- ☐ 2. Comparez votre grille avec la colonne « Ce qui est testé » de l'[annexe A9](#). Notez la frontière que les vingt foyers ne testent pas : _____
- ☐ 3. Ouvrez `mes_tests.csv` dans VS Code. Lisez l'[annexe A10](#). Gardez la ligne d'exemple M1.
- ☐ 4. Ajoutez au moins six cas : deux par frontière de votre grille, sauf celle de l'âge, déjà en exemple. Calculez chaque attendu à la main **avant** de lancer.
- ☐ 5. Lancez votre jeu de tests. Si une ligne échoue, cherchez qui se trompe : votre programme ou votre calcul à la main.

```
python lancer_tests.py mes_tests.csv
```

- ☐ 6. **Le parcours imposé.** Dans votre programme, changez `>= 3` en `> 3` dans la condition du forfait famille. Enregistrez. Lancez les deux jeux de tests. Puis remettez `>= 3`, et relancez les deux jeux.

Ce qui doit marcher à la fin de la partie C

- ☐ `mes_tests.csv` contient au moins sept cas, dont M1, avec une colonne `ce_que_je_teste` remplie.
- ☐ Avec `>= 3`, les deux jeux de tests sont réussis en entier.
- ☐ Avec `> 3`, vos propres tests ont repéré l'erreur : au moins une ligne ÉCHEC.
- ☐ Capture 2 ajoutée au mini-rapport : le terminal avec votre jeu de tests réussi.

POINT DE CONTRÔLE C

- ☐ Le formateur a vu votre grille des cas limites et `journal_tests.txt` avec le parcours imposé. Passez à la partie D.

PARTIE D — Commenter pour Bruno

Environ 20 min.

Bruno reprendra votre programme quand les tarifs changeront. Il ne vous appellera pas. Vos commentaires doivent lui dire quelles règles vous avez retenues, et où elles sont dans le code.

- ☐ 1. En haut de `cotisation.py`, sous le titre, écrivez un bloc de commentaires avec ce modèle. Complétez chaque ligne.

```
# Règles retenues (Marc Oliveira, trésorier) :  
#   - tarif par personne : ...  
#   - forfait famille : ...  
#   - demi-tarif : ...  
# Ordre des calculs : ...  
# Entrées : ...  
# Sortie : ...
```

☐ 2. Au-dessus de chaque `if` et de la boucle, ajoutez un commentaire d'une ligne : ce que fait ce morceau, pas comment Python le lit. Par exemple, `# Montant de base : le forfait famille passe avant les tarifs par personne.`

☐ 3. Relancez les deux jeux de tests : vos commentaires ne doivent rien casser.

☐ 4. Donnez votre programme à lire à un autre binôme, sans rien expliquer. Il répond à l'écrit, en lisant seulement le code :

- à quel âge commence le tarif adulte ?
- quelle ligne changer si le forfait passe à 60 € ?
- à partir de quel mois le demi-tarif s'applique-t-il ?

Ce qui doit marcher à la fin de la partie D

☐ Le bloc « Règles retenues » est complet, sans ligne « ... ».

☐ Chaque `if` et la boucle ont leur commentaire.

☐ L'autre binôme a répondu juste aux trois questions, sans vous poser de question.

☐ Les deux jeux de tests sont toujours réussis.

POINT DE CONTRÔLE D

☐

Le formateur a vu les trois réponses de l'autre binôme. Rédigez votre mini-rapport.

Tous les documents du module

A1 — Le dossier de travail du labo

↓ Télécharger A1_conditions_boucles.zip

Source : fichiers écrits pour le module. Fichier [A1_conditions_boucles.zip](#) à télécharger ci-dessus. Utilisée dans : LABO étapes 1, 2, 7, 12, 13, 17 à 24.

Le fichier [A1_conditions_boucles.zip](#) contient un dossier `conditions_boucles` :

Élément	Ce que c'est
<code>ex01.py</code> à <code>ex15.py</code>	un fichier par exercice. Chacun contient seulement deux lignes de commentaire : le titre de l'exercice, et la place où écrire votre code.
<code>traces\trace1.py</code> à <code>traces\trace3.py</code>	les trois boucles à tracer à la main, décrites dans l' annexe A3
<code>pannes\panne1.py</code> à <code>pannes\panne4.py</code>	les quatre programmes en panne, décrits dans l' annexe A4
<code>tournoi.py</code>	l'outil de tournoi, à écrire. Il contient seulement trois lignes de commentaire.
<code>verifier_equipes.py</code>	l'outil qui vérifie votre <code>tournoi.py</code> . Mode d'emploi dans l' annexe A5 . Ne le modifiez pas.

Une ligne qui commence par `#` est un **commentaire** : Python ne l'exécute pas.

Le mode d'emploi

1. Téléchargez le zip, puis extrayez-le dans « Documents » : clic droit, « Extraire tout », puis « Parcourir ».
2. Ouvrez le dossier `Documents\conditions_boucles` dans VS Code.
3. Pour repartir de zéro, supprimez le dossier `conditions_boucles` et extrayez de nouveau le zip.

A2 — Les quinze énoncés

Source : énoncés écrits pour le module. Tarifs : document de scénario, § 5 « Volumétrie et règles métier ». Jeux et jours d'ouverture : document de scénario, § 2 et § 5. Utilisée dans : LABO étapes 3 à 6, 9 à 11, 14 à 16.

Pour chaque exercice : écrivez le code dans le fichier du même numéro, enregistrez, lancez avec `python exNN.py`. Faites chaque essai demandé.

Le résultat doit être celui de « Vous devez voir », au caractère près. Ce que vous tapez au clavier est écrit après le deux-points de la question.

Les exercices vont du plus simple au plus difficile :

- 1 à 5 : des conditions ;
- 6 à 10 : une boucle `for` ;
- 11 à 15 : une boucle `while`, `break` ou `continue`.

Exercice 1 — Tarif jeune ou adulte

Demandez l'âge. Moins de 18 ans : affichez `Tarif jeune`. Sinon : `Tarif adulte`.

Essais : 17, puis 18.

```
Votre âge : 17
Tarif jeune
```

```
Votre âge : 18
Tarif adulte
```

Exercice 2 — Une partie de Catane est-elle possible ?

Catane se joue de 3 à 4 joueurs. Demandez le nombre de joueurs.

De 3 à 4 : affichez `Partie de Catane possible`. Sinon : `Pas de partie de Catane`. Utilisez `and`.

Essais : 2, 3, 4, puis 5. Vous devez voir « possible » seulement pour 3 et 4.

```
Nombre de joueurs : 4
Partie de Catane possible
```

Exercice 3 — Le local est-il ouvert ?

Le local de l'association est ouvert le mercredi, le vendredi et le samedi. Demandez le jour, écrit en minuscules.

Affichez `Le local est ouvert` ou `Le local est fermé`. Utilisez `or`.

Essais : samedi, puis lundi.

```
Jour : samedi
Le local est ouvert
```

Exercice 4 — À quelle table ?

Pendant les soirées, Léa place les joueurs par âge :

- moins de 12 ans : table des enfants ;
- de 12 à 17 ans : table des ados ;
- 18 ans et plus : table des adultes.

Demandez l'âge, puis affichez la table. Utilisez `if`, `elif` et `else`.

Essais : 11, 12, 17, puis 18.

```
Âge du joueur : 12
Table des ados
```

Vous devez voir : `Table des enfants` pour 11, `Table des ados` pour 12 et 17, `Table des adultes` pour 18.

Exercice 5 — L'emprunt est-il accepté ?

Un adhérent peut emprunter un jeu s'il est à jour de sa cotisation et s'il a moins de 2 jeux chez lui. Demandez `oui` ou `non` pour la cotisation, puis le nombre de jeux déjà empruntés. Affichez `Emprunt accepté` ou `Emprunt refusé`.

Essais : oui et 1, oui et 2, non et 0.

```
Cotisation à jour ? (oui/non) : oui
Jeux déjà empruntés : 1
Emprunt accepté
```

Vous devez voir : `Emprunt refusé` pour les deux autres essais.

Exercice 6 — Les numéros des tables

Ce mercredi, six tables de jeu sont installées. Affichez-les, de `Table 1` à `Table 6`, avec une boucle `for` et `range`.

```
Table 1
Table 2
Table 3
Table 4
Table 5
Table 6
```

Exercice 7 — Le compte à rebours

Le tournoi commence. Affichez les nombres de 10 à 1, un par ligne, puis `C'est parti !`. Utilisez `range` avec un pas de `-1`.

```
10
9
...
1
C'est parti !
```

Les trois points remplacent les lignes 8 à 2 : votre programme les affiche toutes.

Exercice 8 — Épeler un jeu

Parcourez le texte `"DIXIT"` avec une boucle `for`. Affichez chaque lettre sur une ligne. Comptez les lettres `I`, puis affichez le nombre après la boucle.

```
D
I
X
I
T
Nombre de I : 2
```

Exercice 9 — Le total des cotisations encaissées

Ce mercredi, Marc a encaissé cinq cotisations : 12, 25, 25, 12 et 55 €. Parcourez la séquence `[12, 25, 25, 12, 55]` avec une boucle `for`, et additionnez.

```
Total encaissé : 129 €
```

Exercice 10 — Les joueurs de toutes les tables

Demandez le nombre de tables. Puis, pour chaque table, demandez le nombre de joueurs. Affichez le total à la fin.

Essai : 3 tables, avec 4, 4 et 5 joueurs.

```
Nombre de tables : 3
Joueurs à la table 1 : 4
Joueurs à la table 2 : 4
Joueurs à la table 3 : 5
Joueurs en tout : 13
```

Exercice 11 — Redemander un âge possible

Demandez l'âge. Tant qu'il est plus petit que 0 ou plus grand que 120, affichez `Âge impossible, recommencez.`, puis redemandez. À la fin, affichez l'âge enregistré.

Essai : -3, puis 150, puis 17.

```
Votre âge : -3
Âge impossible, recommencez.
Votre âge : 150
Âge impossible, recommencez.
Votre âge : 17
Âge enregistré : 17
```

Exercice 12 — Le code du coffre à jeux

Le coffre à jeux s'ouvre avec le code `MDJ2026`. Demandez le code.

Tant qu'il est faux, redemandez. À la fin, affichez le nombre d'essais.

Essai : `mdj`, puis `1234`, puis `MDJ2026`.

```
Code du coffre : mdj
Code faux. Code du coffre : 1234
Code faux. Code du coffre : MDJ2026
Coffre ouvert en 3 essais
```

Exercice 13 — Économiser pour un jeu

Bruno met 5 € de côté chaque semaine pour acheter un jeu à 42 €. Avec une boucle `while`, calculez le nombre de semaines qu'il lui faut. Aucune saisie.

```
Il faut 9 semaines
```

Exercice 14 — Inscrire des joueurs jusqu'à « fin »

Léa inscrit les joueurs d'une soirée. On ne sait pas combien il y en aura.

Écrivez une boucle `while True` qui demande un prénom. Si le prénom est `fin`, sortez de la boucle avec `break`. Sinon, comptez un joueur de plus.

Affichez le total à la fin.

Essai : Léa, Bruno, Nadia, puis fin.

```
Prénom du joueur (fin pour arrêter) : Léa
Prénom du joueur (fin pour arrêter) : Bruno
Prénom du joueur (fin pour arrêter) : Nadia
Prénom du joueur (fin pour arrêter) : fin
Joueurs inscrits : 3
```

Exercice 15 — Sauter la table réservée

Le jour du tournoi, dix tables sont installées. La table 7 est réservée aux animateurs.

Affichez `Table N ouverte au public` pour chaque table, sauf la 7. Utilisez `continue`.

```
Table 1 ouverte au public
...
Table 6 ouverte au public
Table 8 ouverte au public
Table 9 ouverte au public
Table 10 ouverte au public
```

Vous devez voir neuf lignes, sans la table 7.

A3 — Les trois boucles à tracer

Source : programmes écrits pour le module. Fichiers dans le dossier `traces` du zip de l'[annexe A1](#). Utilisée dans : LABO étapes 7, 12 et 13.

Tracer une boucle, c'est la faire à la main avant de la lancer. On écrit une ligne de tableau à chaque tour, avec la valeur de chaque variable à la fin du tour. Ne lancez un programme qu'après avoir rempli son tableau.

Boucle 1 — `traces\trace1.py`

```
# Boucle 1 : additionner les nombres de 1 à 4
total = 0
for n in range(1, 5):
    total = total + n
print("Total :", total)
```

Boucle 2 — `traces\trace2.py`

```
# Boucle 2 : compter les joueurs de 12 ans ou plus
joueurs = 0
for age in [16, 9, 42, 17, 30]:
    if age >= 12:
        joueurs = joueurs + 1
print("Joueurs de 12 ans ou plus :", joueurs)
```

Boucle 3 — `traces\trace3.py`

```
# Boucle 3 : ranger 10 jeux dans des caisses de 3
jeux = 10
caisses = 0
while jeux > 0:
    jeux = jeux - 3
    caisses = caisses + 1
print("Caisses :", caisses, "- jeux :", jeux)
```

A4 — Les quatre programmes en panne

Source : programmes écrits pour le module. Fichiers dans le dossier `pannes` du zip de l'[annexe A1](#). Utilisée dans : LABO étapes 17 à 20.

Bruno Sanchez, un bénévole, a écrit ces quatre programmes. Aucun n'affiche de message d'erreur. Pourtant, aucun ne fait ce qu'il doit faire.

Ce sont des **erreurs de logique** : le code est bien écrit, mais il ne dit pas ce que Bruno voulait.

Chaque programme a un défaut d'une nature différente. Les quatre natures possibles :

Nature du défaut	Ce que ça veut dire
Boucle infinie	La condition du <code>while</code> reste toujours vraie : la boucle ne s'arrête jamais.
Conditions dans le mauvais ordre	Une condition placée trop tôt prend tous les cas. Celle d'après n'est jamais atteinte.
Erreur de borne	La boucle fait un tour de trop ou un tour de moins.
Mauvais opérateur logique	<code>or</code> à la place de <code>and</code> , ou l'inverse : la condition est vraie dans de mauvais cas.

Panne 1 — `pannes\panne1.py`

Ce qu'il doit faire : afficher `Table 1 prête` à `Table 5 prête`, puis `Toutes les tables sont prêtes.`

```
# Panne 1 : afficher les 5 tables de la soirée
table = 1
while table <= 5:
    print("Table", table, "prête")
print("Toutes les tables sont prêtes.")
```

Panne 2 — `pannes\panne2.py`

Ce qu'il doit faire : moins de 12 ans, table des enfants. De 12 à 17 ans, table des ados. 18 ans et plus, table des adultes.

```
# Panne 2 : placer un joueur à la bonne table selon son âge
age = int(input("Âge du joueur : "))
if age >= 12:
    print("Table des ados")
elif age >= 18:
    print("Table des adultes")
else:
    print("Table des enfants")
```

Panne 3 — `pannes\panne3.py`

Ce qu'il doit faire : afficher les douze postes de l'association, de Poste numéro 1 à Poste numéro 12.

```
# Panne 3 : afficher les 12 postes de l'association
for poste in range(1, 12):
    print("Poste numéro", poste)
print("Fin de la liste.")
```

Panne 4 — `pannes\panne4.py`

Ce qu'il doit faire : Catane se joue de 3 à 4 joueurs. Afficher « possible » seulement pour 3 ou 4 joueurs.

```
# Panne 4 : Catane se joue de 3 à 4 joueurs
joueurs = int(input("Nombre de joueurs : "))
if joueurs >= 3 or joueurs <= 4:
    print("Partie de Catane possible")
else:
    print("Pas de partie de Catane")
```

A5 — Les règles de constitution des équipes de tournoi

Source : document de scénario, § 5 « Bloc U6 », constitution d'équipes de tournoi. Outil `verifier_equipes.py` écrit pour le module, dans le zip de l'[annexe A1](#). Utilisée dans : LABO étapes 21 à 24.

MaisonDesJeux organise deux tournois par an. Le jour du tournoi, Léa Fournier forme les équipes avec ces règles. Elles ont été votées par le bureau de l'association.

Les règles

1. Chaque participant tire au sort un jeton numéroté dans un sac. Avec n participants, les jetons vont de 1 à n . Chaque numéro sort une seule fois.
2. Il faut au moins 3 participants. En dessous, le tournoi n'a pas lieu.
3. Le nombre d'équipes se calcule ainsi : $nb = (n + 2) // 4$.
4. Le participant qui a tiré le numéro i va dans l'équipe $i \% nb$. Ce calcul donne un nombre de 0 à $nb - 1$.
5. Sur l'écran, on numérote les équipes à partir de 1. L'équipe 0 du calcul s'affiche « Équipe 1 », l'équipe 1 s'affiche « Équipe 2 », et ainsi de suite.

Ce que garantissent ces règles

- Chaque équipe a 3, 4 ou 5 membres.
- Deux équipes ont au plus un membre d'écart : elles sont équilibrées.
- Personne n'est oublié, personne n'est dans deux équipes.

Des exemples pour vous vérifier

Participants n	Équipes nb	Membres par équipe
12	3	4, 4, 4
9	2	5, 4
6	2	3, 3
5	1	5

L'affichage attendu, ici pour 12 participants :

```
Nombre de participants : 12
12 participants, 3 équipes
Équipe 1 : 3 6 9 12
Équipe 2 : 1 4 7 10
Équipe 3 : 2 5 8 11
```

Chaque équipe tient sur une seule ligne : le mot `Équipe`, son numéro, deux-points, puis les numéros des jetons séparés par une espace.

L'outil de vérification `verifier_equipes.py`

À quoi il sert : il lance votre `tournoi.py` 31 fois, avec 0, 1, 2... jusqu'à 30 participants. Il vérifie à chaque fois les trois garanties ci-dessus, et le refus en dessous de 3 participants.

Comment le lancer, dans le dossier `conditions_boucles` :

```
python verifier_equipes.py
```

Ce qu'il affiche quand tout est juste :

```
31 cas vérifiés (0 à 30 participants), 0 problème(s).
```

Ce qu'il affiche quand quelque chose ne va pas, une ligne par cas faux, par exemple :

```
n = 2 : PROBLÈME, il faut refuser le tournoi, mais des équipes sont affichées
n = 10 : PROBLÈME, un numéro manque ou est en double
31 cas vérifiés (0 à 30 participants), 2 problème(s).
```

L'outil lit seulement les lignes qui commencent par `Équipe` et contiennent un deux-points. Respectez l'affichage attendu.

A6 — Extrait de la documentation de Python

Source : *documentation officielle de Python 3*, [docs.python.org](https://docs.python.org/3), « *Le tutoriel Python* », chapitres 3 et 4, et « *Fonctions natives* », traduite et raccourcie pour le module. Utilisée dans : LABO étapes 8, 15 et 23.

`range` : une suite de nombres

`range(fin)` donne les nombres de 0 à `fin - 1`. Le nombre `fin` n'est jamais donné.

`range(debut, fin)` donne les nombres de `debut` à `fin - 1`.

`range(debut, fin, pas)` avance de `pas` à chaque fois. Avec un pas négatif, la suite descend. Elle s'arrête avant d'atteindre `fin`.

Écriture

Nombres donnés

`range(4)`

0, 1, 2, 3

`range(2, 6)`

2, 3, 4, 5

Écriture	Nombres donnés
<code>range(0, 10, 3)</code>	0, 3, 6, 9
<code>range(5, 0, -1)</code>	5, 4, 3, 2, 1

print : afficher sans aller à la ligne

`print(valeur, ..., sep=" ", end="\n")` affiche les valeurs séparées par `sep`, puis écrit `end`.

Par défaut, `end` vaut `"\n"` : un retour à la ligne. Avec `end=" "`, `print` écrit une espace à la place.

Le prochain `print` continue sur la même ligne. Un `print()` vide termine la ligne.

```
for n in range(3):
    print(n, end=" ")
print()
```

Affiche : 0 1 2 , sur une seule ligne.

and, or, not

Opération	Résultat
<code>x or y</code>	vrai si au moins un des deux est vrai
<code>x and y</code>	vrai seulement si les deux sont vrais
<code>not x</code>	vrai si <code>x</code> est faux, faux si <code>x</code> est vrai

Ordre de priorité : `not` est calculé en premier, puis `and`, puis `or`. Ainsi, `a or b and c` se lit `a or (b and c)`. En cas de doute, mettez des parenthèses.

Python calcule de gauche à droite et s'arrête dès qu'il connaît le résultat. Dans `x and y`, si `x` est faux, `y` n'est pas calculé.

break et continue

`break` sort tout de suite de la boucle `for` ou `while` qui le contient. Le programme continue après la boucle.

`continue` arrête le tour en cours et passe au tour suivant de la boucle.

Arrêter un programme qui ne s'arrête pas

Dans un terminal, Ctrl+C arrête le programme en cours. Python affiche alors `KeyboardInterrupt`.

A7 — Le point de départ du TP

↓ Télécharger [A7_cotisations.zip](#)

Source : fichiers écrits pour le module. Foyers de test : document de scénario, § 5 « Bloc U6 ». Fichier [A7_cotisations.zip](#) à télécharger ci-dessus. Utilisée dans : TP parties A à D.

Le fichier [A7_cotisations.zip](#) contient un dossier `cotisations` :

Élément	Ce que c'est
<code>cotisation.py</code>	le programme à écrire. Il contient déjà des commentaires et les deux lignes qui lisent la date.
<code>foyers_reference.csv</code>	les vingt foyers de test de l' annexe A9 , prêts à lancer
<code>mes_tests.csv</code>	votre jeu de tests, à compléter. Il contient l'en-tête et une ligne d'exemple.
<code>lancer_tests.py</code>	l'outil qui lance un jeu de tests sur <code>cotisation.py</code> . Ne le modifiez pas.

Ce que `cotisation.py` doit faire

Il demande, dans cet ordre :

1. le nombre de personnes du foyer ;
2. l'âge de chaque personne, une par une ;
3. la date d'adhésion, écrite `AAAA-MM-JJ`, par exemple `2026-01-15`.

Sa dernière ligne affichée donne le montant avec deux chiffres après le point :

```
Cotisation : 12.50 €
```

Les deux lignes déjà écrites

```
date_adhesion = input("Date d'adhésion (AAAA-MM-JJ) : ")
mois = int(date_adhesion[5:7])
```

La première lit la date en texte. La seconde garde les caractères numéros 5 et 6 du texte, puis les convertit en nombre.

On compte à partir de 0 : dans "2026-01-15", les caractères 5 et 6 sont 01. La variable `mois` vaut donc 1.

Ne modifiez pas ces deux lignes. Placez-les après la lecture des âges : la date est demandée en dernier.

L'outil `lancer_tests.py`

Comment le lancer, dans le dossier `cotisations` :

```
python lancer_tests.py foyers_reference.csv
python lancer_tests.py mes_tests.csv
```

Pour chaque ligne du fichier, l'outil lance `cotisation.py` et tape à votre place le nombre de personnes, chaque âge, puis la date. Il lit le montant sur la ligne `Cotisation : ... €`, puis le compare à la colonne `attendu`.

Ce qu'il affiche, par exemple :

```
=== 2026-10-05 14:32:10 · mes_tests.csv · dupont sur PC-SALLE-12
  M1 OK      attendu 12.00 obtenu 12.00          exemple : 17 ans, encore
jeune
  M2 ÉCHEC   attendu 25.00 obtenu 12.00          18 ans : adulte
Résultat : 1 sur 2 réussis.
```

Ce que veulent dire les messages d'échec :

Message dans la colonne
« obtenu »

Ce qui se passe

un montant

le programme tourne, mais le montant est faux

`plante : ...`

le programme s'arrête sur une erreur, recopiée après les deux-points

`ne s'arrête pas`

le programme tourne encore après 5 secondes : boucle infinie, ou il attend une saisie de trop

pas de ligne Cotisation la ligne Cotisation : ... € n'est pas affichée, ou elle est mal écrite

Chaque lancement est ajouté à la fin du fichier `journal_tests.txt`, avec la date, l'heure, votre nom de session et le nom du poste. Ce journal sert pour votre mini-rapport.

Pour repartir de zéro : supprimez le dossier `cotisations`, puis extrayez de nouveau le zip. Gardez une copie de votre `cotisation.py` avant.

A8 — Les règles de cotisation, écrites par Marc Oliveira

Source : document de scénario, § 5 « Bloc U6 », règles de cotisation, et § 3 « Les interlocuteurs ». Texte rédigé pour le module dans les mots de Marc Oliveira. Début et fin de saison : écrits pour le module. Utilisée dans : TP parties A, C et D.

Bonjour,

Voici comment je calcule les cotisations. C'est moi qui les encaisse depuis six ans, alors je préfère que le programme fasse exactement comme moi.

La saison commence le 1er septembre et finit le 31 août de l'année suivante.

Quand quelqu'un s'inscrit après le Nouvel An, c'est-à-dire à partir du 1er janvier, il ne paie que la moitié. Ça vaut pour tout le monde, même pour les familles. Une famille qui s'inscrit en février paie donc la moitié du forfait.

Le prix normal dépend de l'âge le jour où la personne s'inscrit. Les jeunes paient 12 €. Attention, pour moi un jeune, c'est jusqu'à 17 ans compris. Quelqu'un qui a 18 ans le jour de l'inscription paie le tarif adulte, 25 €.

On me demande souvent un tarif senior. Il n'y en a pas : un adhérent de 70 ans paie 25 €, comme les autres adultes.

Pour les familles : si un foyer inscrit trois personnes ou plus en même temps, il paie un forfait de 55 € pour tout le foyer. Pas 55 € chacun : 55 € en tout. Peu importe l'âge des membres. Même si trois enfants à 12 € coûteraient moins cher, c'est le forfait qui s'applique : c'est la règle votée en assemblée générale. À deux personnes, ce n'est pas une famille : chacun paie son tarif.

Le demi-tarif se calcule toujours en dernier, sur le montant du foyer.

Merci, Marc Oliveira, trésorier

A9 — Les vingt foyers de test

Source : document de scénario, § 5 « Bloc U6 », vingt foyers de test. Fichier

`foyers_reference.csv` dans le zip de l'[annexe A7](#). Utilisée dans : TP parties A, B et C.

Un **foyer**, ce sont les personnes d'une même maison qui s'inscrivent ensemble. Souvent, un foyer compte une seule personne. Les âges sont ceux du jour de l'inscription.

Tous les foyers s'inscrivent pendant la saison qui va du 1er septembre 2025 au 31 août 2026.

Cas	Âges du foyer	Date d'adhésion	Cotisation attendue	Ce qui est testé
1	30	2025-09-15	25.00 €	adulte plein tarif

Cas	Âges du foyer	Date d'adhésion	Cotisation attendue	Ce qui est testé
2	16	2025-09-20	12.00 €	jeune
3	17	2025-10-01	12.00 €	borne : 17 ans = jeune
4	18	2025-10-01	25.00 €	borne : 18 ans = adulte
5	42, 40, 12	2025-09-10	55.00 €	famille de 3 : forfait
6	45, 43	2025-09-10	50.00 €	2 adultes, pas famille
7	35	2026-01-01	12.50 €	borne : 1er janvier = demi-tarif
8	35	2025-12-31	25.00 €	borne : 31 décembre = plein tarif
9	40, 38, 10, 8	2026-02-01	27.50 €	famille + demi-tarif
10	16	2026-03-15	6.00 €	jeune + demi-tarif
11	50, 20, 19	2025-09-05	55.00 €	famille, âges mêlés
12	70	2025-09-05	25.00 €	adulte âgé, pas de tarif senior
13	17, 15	2025-09-12	24.00 €	2 jeunes
14	17, 15	2026-01-20	12.00 €	2 jeunes + demi-tarif
15	60, 58, 30, 28	2025-09-08	55.00 €	famille de 4
16	25	2026-01-02	12.50 €	adulte + demi-tarif
17	12, 10, 8	2025-09-14	55.00 €	famille de 3 jeunes
18	44, 22	2025-12-20	50.00 €	2 adultes plein tarif
19	44, 22	2026-01-10	25.00 €	2 adultes + demi-tarif
20	80, 55, 30	2026-01-05	27.50 €	famille + demi-tarif

A10 — Le gabarit du jeu de tests

Source : gabarit écrit pour le module. Fichier `mes_tests.csv` dans le zip de l'[annexe A7](#).
Utilisée dans : TP partie C.

Un **jeu de tests** est une liste de cas. Pour chaque cas, on écrit les données d'entrée et le résultat attendu, calculé à la main, avant de lancer le programme. Un cas qui ne teste rien de nouveau ne sert à rien : chaque ligne teste une chose précise.

Le format : une ligne par cas, cinq colonnes séparées par un point-virgule. La première ligne est l'en-tête : ne la changez pas.

```
cas;ages;date;attendu;ce_que_je_teste
M1;17;2025-10-01;12.00;exemple : 17 ans, encore jeune
```

Colonne	Ce qu'on y écrit	Exemple
cas	un nom court : M1, M2, M3...	M2
ages	les âges du foyer, séparés par une espace	42 40 12
date	la date d'adhésion, AAAA-MM-JJ	2026-08-31
attendu	la cotisation calculée à la main, avec un point et deux chiffres	27.50
ce_que_je_teste	en quelques mots, le cas visé	famille de 3 en août

Les règles du gabarit

- Un point-virgule entre chaque colonne, aucun ailleurs.
- Pas d'espace avant ou après un point-virgule.
- Pas de symbole € dans la colonne **attendu**.
- Enregistrez le fichier dans VS Code avec Ctrl+S. Ne l'ouvrez pas dans un tableur : il risque de changer les points-virgules ou les nombres.