CODE TON PREMIER JEU EN PYTHON

Du concept au jeu interactif complet (Console + Interface Graphique)

C Premiers Pas - vCODE #4

Les bases solides d'aujourd'hui sont les succès de demain

📑 Table des matières

Introduction : Ton premier vrai jeu !

Le problème concret

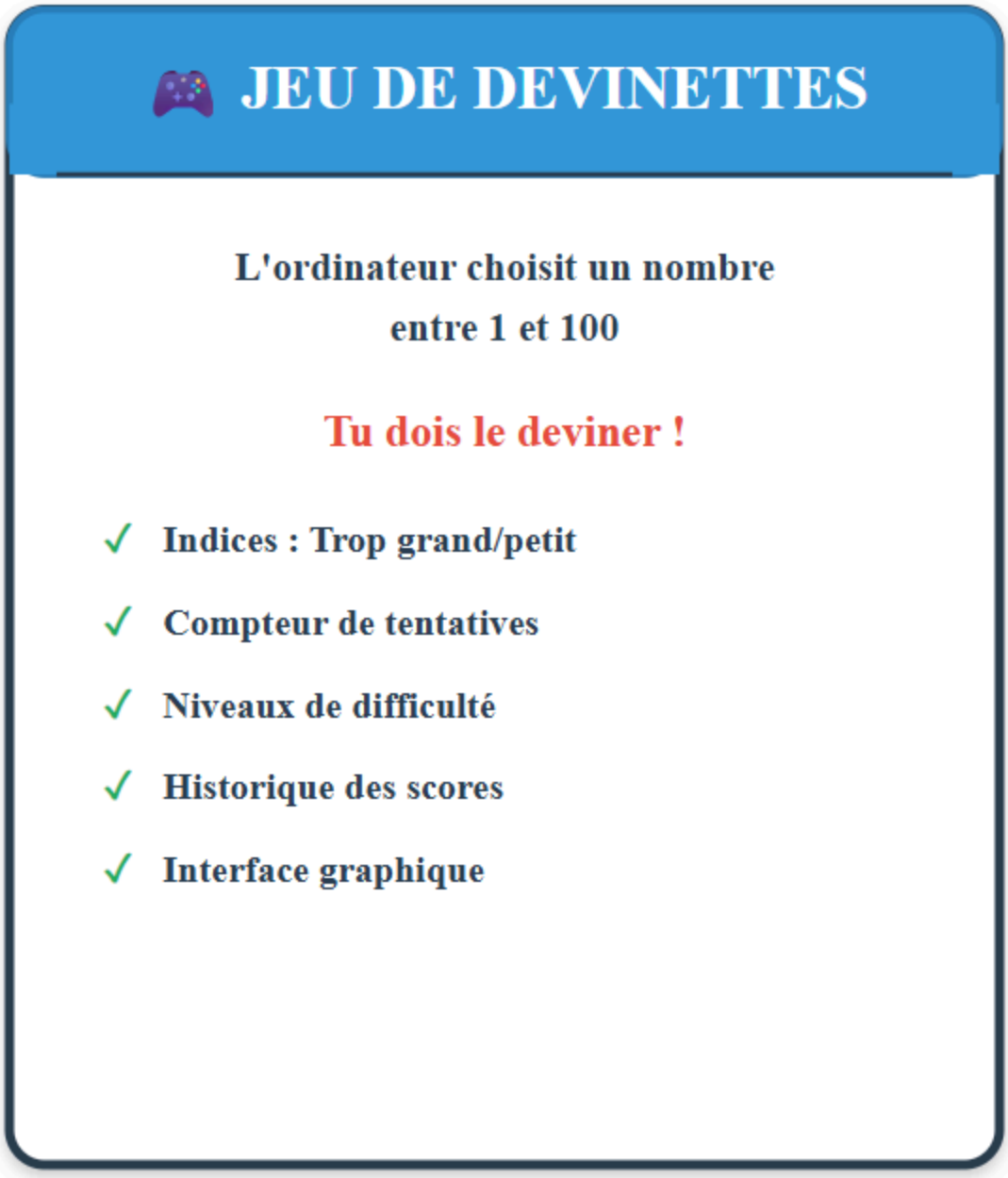
Tu joues à des jeux sur ton téléphone, ton ordinateur, ta console ? **Et si TU créais TON PROPRE JEU ?** Pas un jeu compliqué avec des graphismes 3D, mais un vrai jeu fonctionnel que tu auras programmé toi-même de A à Z !

Un jeu qui :

- ⇨ Est amusant à jouer
- ⇨ Défie ton intelligence
- ⇨ Garde un historique de tes performances
- ⇨ A une vraie interface graphique
- ⇨ **Et surtout : que TU as créé entièrement**

La solution : Ce qu'on va construire

Aujourd'hui, tu vas créer un **jeu de devinettes intelligent** :



Ce que tu vas apprendre

Compétence	Ce que ça t'apporte
Génération aléatoire	Créer de l'imprévisibilité dans tes programmes
Comparaisons	Prendre des décisions basées sur des valeurs
Compteurs	Suivre des statistiques et des scores
Logique de jeu	Construire des systèmes interactifs intelligents
Interface graphique (Tkinter)	Créer de vraies fenêtres professionnelles

Résultat final attendu

À la fin de cette session, tu auras créé **DEUX versions** du même jeu :

✓ **Version console** : Rapide, efficace, professionnelle
✓ **Version graphique** : Avec fenêtre, boutons, couleurs
✓ **Compris la logique de jeu** : Comment les jeux fonctionnent
✓ **Maîtrisé Tkinter** : Créer des interfaces graphiques
✓ **Développé ta créativité** : Personnaliser et améliorer

Prérequis et concepts clés

Avant de commencer, assurons-nous que tu maîtrises les bases vues dans vCODE #3 :

Révision rapide

Concept	Rappel	Exemple
Variables	Boîtes pour stocker des valeurs	`score = 0`
input()	Demander à l'utilisateur	`reponse = input("Ton nombre : ")`
print()	Afficher du texte	`print("Bravo !")`
if/elif/else	Prendre des décisions	`if score > 10: print("Excellent")`
while	Répéter jusqu'à une condition	`while continuer == True:`
int() / float()	Convertir en nombre	`nombre = int("42")`

Nouveaux concepts d'aujourd'hui

1. 🎲 Le module random

C'est quoi ? Un outil Python qui permet de générer de l'aléatoire (hasard).

```
import random

# Générer un nombre aléatoire entre 1 et 100
nombre_secret = random.randint(1, 100)
```

Analogie : C'est comme lancer un dé. Tu ne sais pas ce que tu vas obtenir !

Fonction	Ce qu'elle fait	Exemple
`random.randint(a, b)`	Nombre entier aléatoire entre a et b (inclus)	`random.randint(1, 6)` → peut donner 1, 2, 3, 4, 5 ou 6
`random.choice(liste)`	Choisit un élément au hasard dans une liste	`random.choice(["rouge", "bleu", "vert"])`
`random.shuffle(liste)`	Mélange les éléments d'une liste	Comme mélanger des cartes

2. Les opérateurs de comparaison

Tu les as déjà vus, mais aujourd'hui on va les utiliser **intensivement** !

```
nombre_utilisateur = 50
nombre_secret = 42

if nombre_utilisateur > nombre_secret:
    print("C'est trop grand !")
elif nombre_utilisateur < nombre_secret:
    print("C'est trop petit !")
else:
    print("Bravo, tu as trouvé !")
```

Les 6 opérateurs de comparaison :

Opérateur	Signification	Exemple	Résultat
`>`	Plus grand que	`50 > 42`	`True`
`<`	Plus petit que	`30 < 42`	`True`
`==`	Égal à	`42 == 42`	`True`
`!=`	Différent de	`50 != 42`	`True`
`>=`	Plus grand ou égal	`42 >= 42`	`True`
`<=`	Plus petit ou égal	`30 <= 42`	`True`

3. 🔢 Les compteurs

Un compteur, c'est une variable qui augmente ou diminue à chaque action.

```
tentatives = 0  # Initialisation

# À chaque essai :
tentatives = tentatives + 1  # Incrémentation

# OU version courte :
tentatives += 1
```

Analogie : Imagine un tableau de score au basketball. Chaque panier marqué, on ajoute des points !

Opération	Code long	Code court	Résultat
Ajouter 1	`compteur = compteur + 1`	`compteur += 1`	Incrémentation
Enlever 1	`compteur = compteur - 1`	`compteur -= 1`	Décrémentation
Multiplier par 2	`compteur = compteur * 2`	`compteur *= 2`	Doublement
Remettre à zéro	`compteur = 0`	`compteur = 0`	Reset

PARTIE 1 : Jeu en console (1h15)

On va construire le jeu **étape par étape**, du plus simple au plus complet !

ÉTAPE 1 : Générer un nombre aléatoire

Objectif

Faire en sorte que l'ordinateur choisisse un nombre secret au hasard.

Le code (Version découverte)

```
# Jeu de devinettes v1.0 - Génération aléatoire

# Importer le module random
import random

# L'ordinateur choisit un nombre secret entre 1 et 100
nombre_secret = random.randint(1, 100)

# Afficher le nombre (juste pour vérifier que ça marche)
print("🎲 L'ordinateur a choisi un nombre entre 1 et 100")
print("DEBUG : Le nombre secret est", nombre_secret)
```

Explication ligne par ligne

Ligne 4 :

```
import random
```

→ NOUVEAU : IMPORTER UN MODULE !

Ce qui se passe :

- `import` = "importer" ou "charger"
- `random` = un ensemble de fonctions pour générer de l'aléatoire
- Python charge ce module pour qu'on puisse l'utiliser

🤔 C'est quoi un module ?

Analogie	Explication
Boîte à outils	Un module = une boîte contenant des outils (fonctions)
Bibliothèque	Comme une bibliothèque contient des livres, un module contient des fonctions
Application	Comme installer une app sur ton téléphone pour avoir de nouvelles fonctionnalités

Ligne 7 :

```
nombre_secret = random.randint(1, 100)
```

→ **Génération du nombre aléatoire !**

Décomposition :

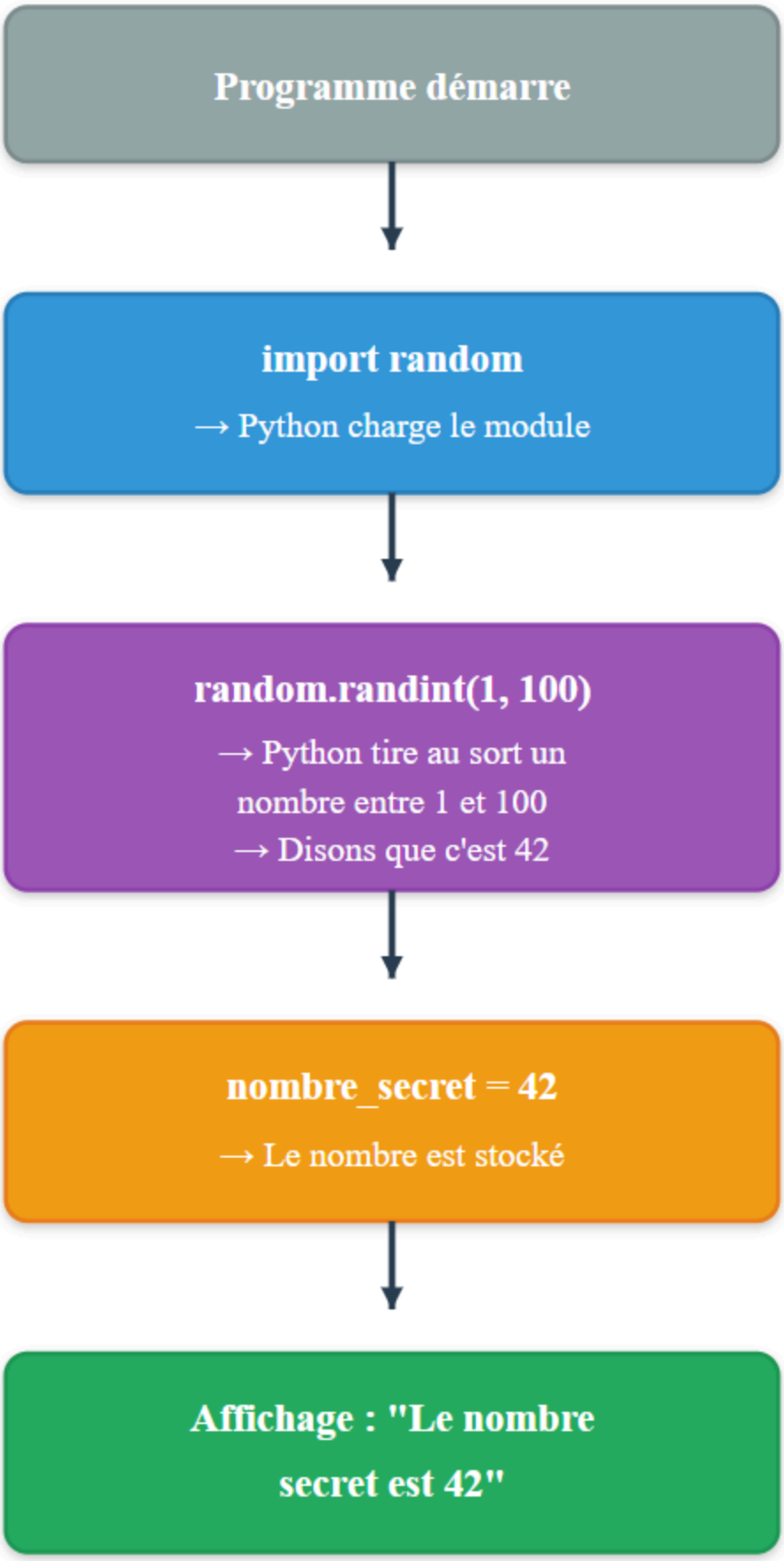
- `random.` = on utilise le module random
- `randint(1, 100)` = génère un **int**eger (entier) **rand**om (aléatoire) entre 1 et 100
- Le résultat est stocké dans `nombre_secret`

Ligne 11 :

```
print("DEBUG : Le nombre secret est", nombre_secret)
```

→ Le mot "DEBUG" indique que c'est une ligne temporaire qu'on va retirer plus tard.

Ce qui se passe vraiment



Résultat attendu

```
🎲 L'ordinateur a choisi un nombre entre 1 et 100
DEBUG : Le nombre secret est 67
```

Note : Le nombre change à chaque exécution !

Test interactif

Exécute le programme **3 fois de suite** et observe :

```
Exécution 1 : Le nombre secret est 23
Exécution 2 : Le nombre secret est 89
Exécution 3 : Le nombre secret est 42
```

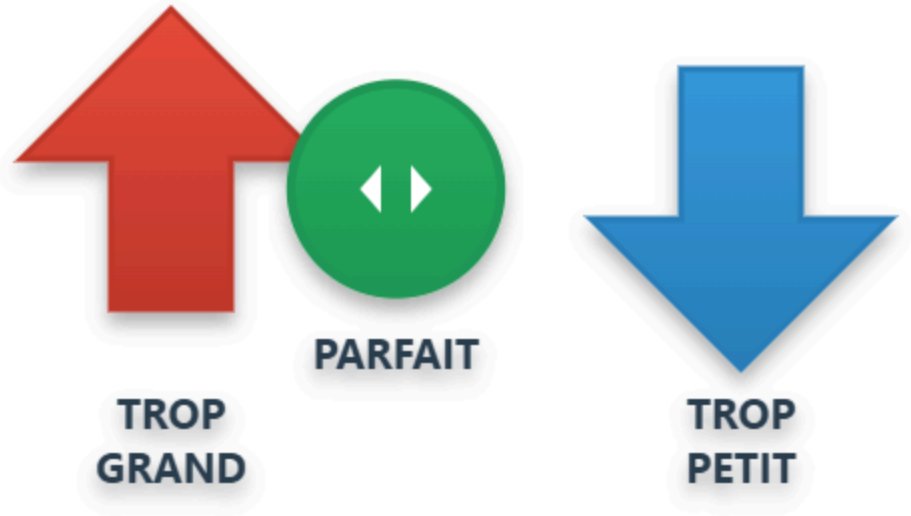
C'est vraiment aléatoire !

Le problème

✓ On peut générer un nombre aléatoire
☒ Mais on ne peut pas encore jouer
☒ Il faut pouvoir deviner et comparer !

Solution : Étape 2 - Les comparaisons !

ÉTAPE 2 : Comparer les nombres



Objectif

Permettre au joueur de proposer un nombre et lui dire s'il est trop grand, trop petit, ou correct.

Le code (Version débutant)

```
# Jeu de devinettes v2.0 - Avec comparaison

import random

# L'ordinateur choisit un nombre secret
nombre_secret = random.randint(1, 100)

print("🎮 JEU DE DEVINETTES")
print("=" * 30)
print("J'ai choisi un nombre entre 1 et 100.")
print("À toi de le deviner !\n")

# Le joueur propose un nombre
proposition = int(input("📌 Ton nombre : "))

# Comparer avec le nombre secret
if proposition > nombre_secret:
    print("☒ C'est trop grand !")
elif proposition < nombre_secret:
    print("☒ C'est trop petit !")
else:
    print("✓ BRAVO ! Tu as trouvé !")
```

Explication ligne par ligne

Ligne 14 :

```
proposition = int(input("📌 Ton nombre : "))
```

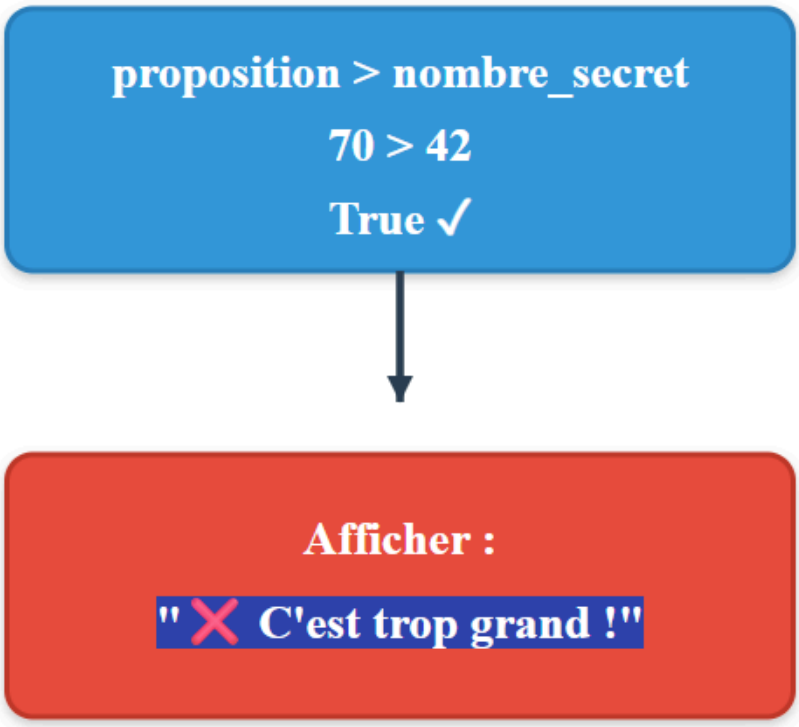
→ On demande au joueur un nombre et on le convertit **directement** en entier (int).

Lignes 17-22 : La logique de comparaison

```
if proposition > nombre_secret:
    print("☒ C'est trop grand !")
```

Ce qui se passe :

Exemple : nombre_secret = 42, proposition = 70



Les 3 cas possibles :

Cas	Condition	Message
Trop grand	`proposition > nombre_secret`	"C'est trop grand !"
Trop petit	`proposition < nombre_secret`	"C'est trop petit !"
Trouvé	`proposition == nombre_secret`	"BRAVO !"

Ce qui se passe vraiment

Scénario 1 : Nombre trop grand

```
Nombre secret : 42
Joueur tape : 70

Test 1 : 70 > 42 ? OUI
→ Affiche "C'est trop grand !"
→ Ne teste pas les autres conditions (elif/else)
```

Scénario 2 : Nombre trop petit

```
Nombre secret : 42
Joueur tape : 20

Test 1 : 20 > 42 ? NON
Test 2 : 20 < 42 ? OUI
→ Affiche "C'est trop petit !"
```

Scénario 3 : Nombre trouvé

```
Nombre secret : 42
Joueur tape : 42

Test 1 : 42 > 42 ? NON
Test 2 : 42 < 42 ? NON
Test 3 : else (c'est égal)
→ Affiche "BRAVO !"
```

Résultat attendu

Partie 1 : Trop grand

```
🎮 JEU DE DEVINETTES
═══════════════════
J'ai choisi un nombre entre 1 et 100.
À toi de le deviner !

📌 Ton nombre : 75
☒ C'est trop grand !
```

Partie 2 : Trop petit

```
📌 Ton nombre : 25
☒ C'est trop petit !
```

Partie 3 : Trouvé

```
📌 Ton nombre : 42
✓ BRAVO ! Tu as trouvé !
```

Le problème

✓ On peut comparer et donner des indices
☒ Mais on ne peut faire qu'**UNE SEULE tentative**
☒ Pas de compteur, pas de statistiques

Solution : Étape 3 - Ajouter un compteur !

ÉTAPE 3 : Compter les tentatives

Objectif

Compter combien de fois le joueur essaie avant de trouver le nombre.

Le code (Version débutant)

```python
# Jeu de devinettes v3.0 - Avec compteur de tentatives

import random

# L'ordinateur choisit un nombre secret
nombre_secret = random.randint(1, 100)

print("🎮 JEU DE DEVINETTES")
print("=" * 30)
print("J'ai choisi un nombre entre 1 et 100.")
print("À toi de le deviner !\n")

# Initialiser le compteur à 0
tentatives = 0

# Le joueur propose un nombre
proposition = int(input("📌 Ton nombre : "))
tentatives += 1  # On compte cette tentative

# Comparer
if proposition > nombre_secret:
    print(f"☒ C'est trop grand ! (Tentative n°{tentatives})")
elif proposition < nombre_secret:
    print(f"☒ C'est trop petit ! (Tentative n°{tentatives})")
else:
    print(f"✓ BRAVO ! Tu as trouvé en {tentatives} tentative(s) !")
```

Explication ligne par ligne

Ligne 14 :

```python
tentatives = 0
```

→ **Initialisation du compteur.** On commence à zéro.

Ligne 18 :

```python
tentatives += 1
```

→ **Incrémentation.** Chaque fois qu'on arrive ici, on ajoute 1 au compteur.

** Les opérateurs d'assignation composée**

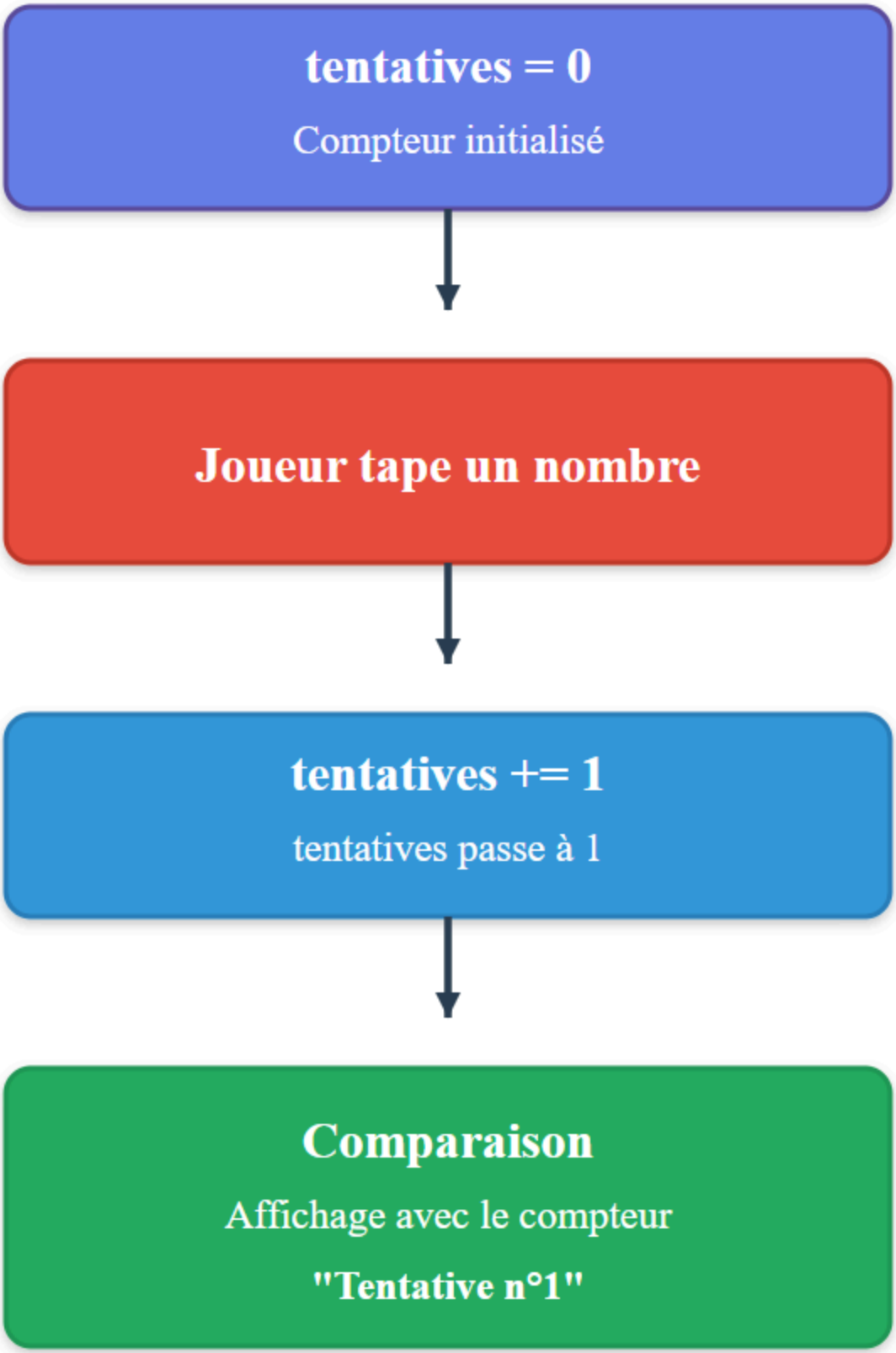
Opération	Code long	Code court
Ajouter 1	`tentatives = tentatives + 1`	`tentatives += 1`
Ajouter 5	`tentatives = tentatives + 5`	`tentatives += 5`
Soustraire 1	`score = score - 1`	`score -= 1`
Multiplier par 2	`points = points * 2`	`points *= 2`

Ligne 22 :

```
print(f"☒ C'est trop grand ! (Tentative n°{tentatives})")
```

→ **f-string** : On affiche le numéro de la tentative dans le message.

Ce qui se passe vraiment



Résultat attendu

```
🎮 JEU DE DEVINETTES
═══════════════════
J'ai choisi un nombre entre 1 et 100.
À toi de le deviner !

📌 Ton nombre : 50
☒ C'est trop petit ! (Tentative n°1)
```

Si on relance et qu'on trouve tout de suite :

```
📌 Ton nombre : 42
✓ BRAVO ! Tu as trouvé en 1 tentative(s) !
```

Le problème

✓ On compte les tentatives
☒ Mais on ne peut faire qu'**UN SEUL essai** puis le programme s'arrête
☒ Il faut pouvoir continuer jusqu'à trouver !

Solution : Étape 4 - La boucle de jeu !

ÉTAPE 4 : Ajouter la boucle de jeu

Objectif

Permettre au joueur de faire autant de tentatives que nécessaire jusqu'à trouver le nombre.

Le code (Version débutant)

```python
# Jeu de devinettes v4.0 - Avec boucle complète

import random

# L'ordinateur choisit un nombre secret
nombre_secret = random.randint(1, 100)

print("🎮 JEU DE DEVINETTES")
print("=" * 30)
print("J'ai choisi un nombre entre 1 et 100.")
print("À toi de le deviner !\n")

# Initialiser les variables
tentatives = 0
trouve = False  # Le joueur n'a pas encore trouvé

# Boucle de jeu : tant qu'on n'a pas trouvé
while not trouve:

    # Le joueur propose un nombre
    proposition = int(input("📌 Ton nombre : "))
    tentatives += 1

    # Comparer
    if proposition > nombre_secret:
        print(f"☒ C'est trop grand ! (Tentative n°{tentatives})\n")
    elif proposition < nombre_secret:
        print(f"☒ C'est trop petit ! (Tentative n°{tentatives})\n")
    else:
        print(f"\n✓ BRAVO ! Tu as trouvé en {tentatives} tentative(s) !")
        trouve = True  # On arrête la boucle

print("\n🎉 Partie terminée !")
```

Explication ligne par ligne

Ligne 15 :

```python
trouve = False
```

→ **Variable booléenne.** C'est un "interrupteur" qui peut être True (vrai) ou False (faux).

🔘 Les booléens : True et False

Valeur	Signification	Exemple d'usage
`True`	Vrai	Le joueur a trouvé le nombre
`False`	Faux	Le joueur n'a pas encore trouvé

Ligne 18 :

```
while not trouve:
```

→ **NOUVEAU : Opérateur NOT !**

Décomposition :

- `not` = "pas" ou "non" en français
- `not trouve` = "tant que trouve est à False"
- Tant que `trouve` est False, la boucle continue
- Dès que `trouve` devient True, la boucle s'arrête

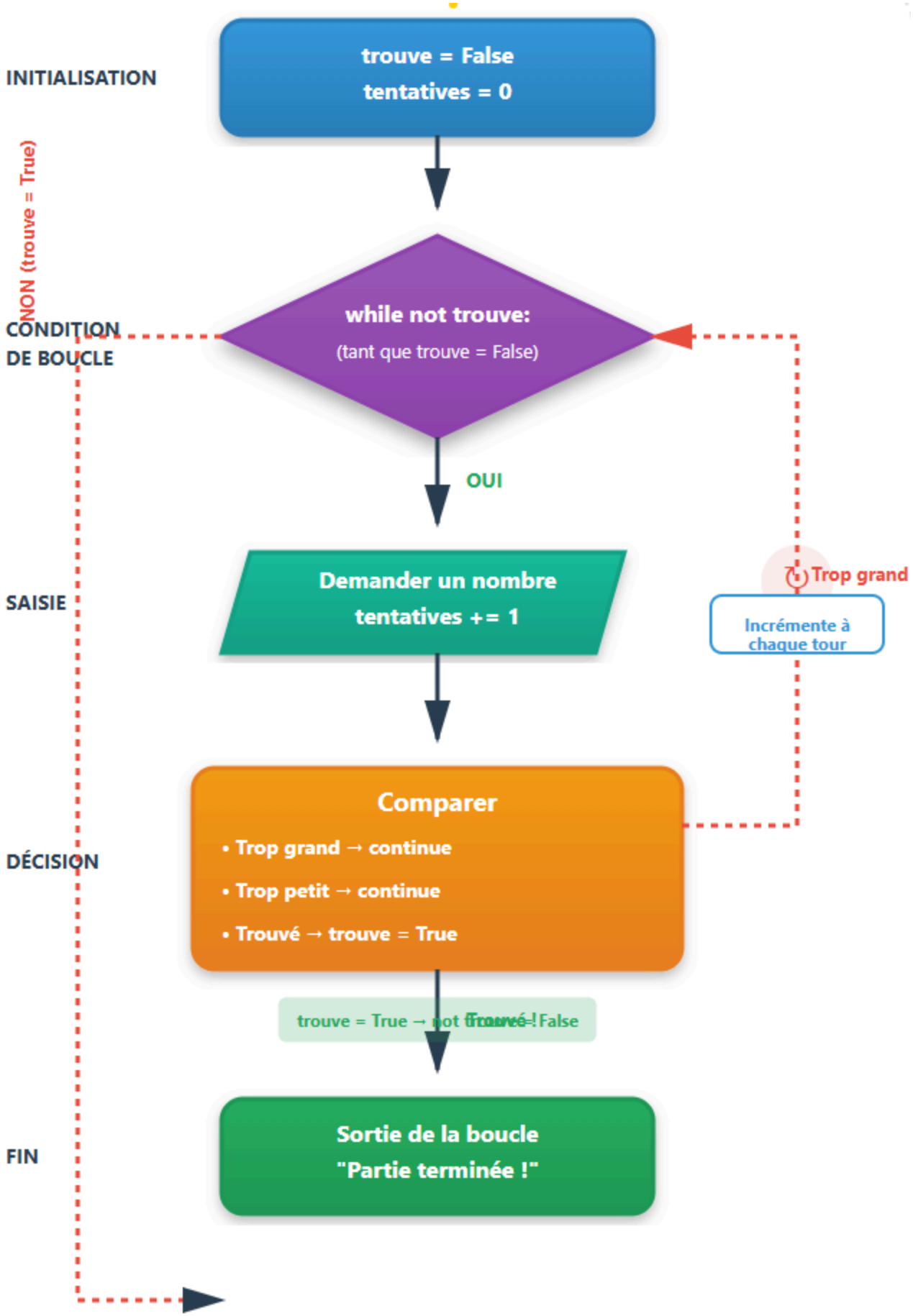
Expression	Valeur de `trouve`	Résultat de `not trouve`	La boucle ?
`not trouve`	False	True	Continue ✓
`not trouve`	True	False	S'arrête ✗

Ligne 31 :

```
trouve = True
```

→ Quand le joueur trouve, on change `trouve` à True, ce qui arrête la boucle !

Ce qui se passe vraiment



Résultat attendu

```
🎮 JEU DE DEVINETTES
═══════════════════
J'ai choisi un nombre entre 1 et 100.
À toi de le deviner !

📌 Ton nombre : 50
☒ C'est trop petit ! (Tentative n°1)

📌 Ton nombre : 75
☒ C'est trop grand ! (Tentative n°2)

📌 Ton nombre : 62
☒ C'est trop petit ! (Tentative n°3)

📌 Ton nombre : 68
☒ C'est trop grand ! (Tentative n°4)

📌 Ton nombre : 65
☒ C'est trop petit ! (Tentative n°5)

📌 Ton nombre : 67

✓ BRAVO ! Tu as trouvé en 6 tentative(s) !

🎉 Partie terminée !
```

Le problème

✓ Le jeu fonctionne complètement !
☒ Mais si le joueur tape du texte au lieu d'un nombre → **CRASH**
☒ Pas de gestion d'erreurs

Solution : Étape 5 - Gérer les erreurs !

ÉTAPE 5 : Gérer les erreurs

Objectif

Rendre le jeu robuste : gérer les entrées invalides sans faire planter le programme.

Le code (Version débutant)

```
# Jeu de devinettes v5.0 - Avec gestion d'erreurs

import random

# L'ordinateur choisit un nombre secret
nombre_secret = random.randint(1, 100)

print("🎮 JEU DE DEVINETTES")
print("=" * 30)
print("J'ai choisi un nombre entre 1 et 100.")
print("À toi de le deviner !\n")

# Initialiser les variables
tentatives = 0
trouve = False

# Boucle de jeu
while not trouve:

    try:
        # Demander un nombre
        proposition = int(input("📌 Ton nombre : "))

        # Vérifier que le nombre est dans la plage
        if proposition < 1 or proposition > 100:
            print("⊗  Le nombre doit être entre 1 et 100 !\n")
            continue  # Retour au début de la boucle

        tentatives += 1

        # Comparer
        if proposition > nombre_secret:
            print(f"☒ C'est trop grand ! (Tentative n°{tentatives})\n")
        elif proposition < nombre_secret:
            print(f"☒ C'est trop petit ! (Tentative n°{tentatives})\n")
        else:
            print(f"\n✓ BRAVO ! Tu as trouvé en {tentatives} tentative(s) !")
            trouve = True

    except ValueError:
        print("☒ Erreur : Tu dois entrer un nombre entier !\n")
```

```
print("\n🎉 Partie terminée !")
```

Explication ligne par ligne

Ligne 20 :

```
try:
```

→ NOUVEAU : TRY/EXCEPT (Gestion d'erreurs) !

Ce qui se passe :

- `try:` = "essaie de faire ça"
- Si ça marche → continue normalement
- Si ça plante → va au bloc `except`

🛡️ Structure try/except

```
try:
    # Code qui pourrait planter
    nombre = int(input("Nombre : "))
except ValueError:
    # Code à exécuter si ça plante
    print("Erreur : ce n'est pas un nombre !")
```

Lignes 25-27 :

```
if proposition < 1 or proposition > 100:
    print("⊗  Le nombre doit être entre 1 et 100 !\n")
    continue
```

→ Validation de la plage.

Décomposition :

- `or` = "ou" - si l'une des deux conditions est vraie
- `continue` = **NOUVEAU !** Retourne au début de la boucle immédiatement

🔄 Le mot-clé continue

Sans continue	Avec continue
Exécute tout le code de la boucle	Saute directement au début de la boucle
Même si une condition est remplie	Sans exécuter le reste

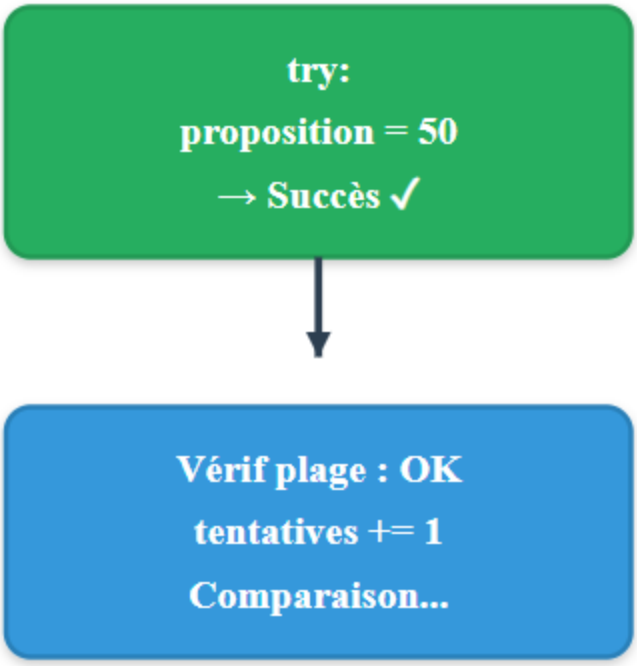
Ligne 40 :

```
except ValueError:
```

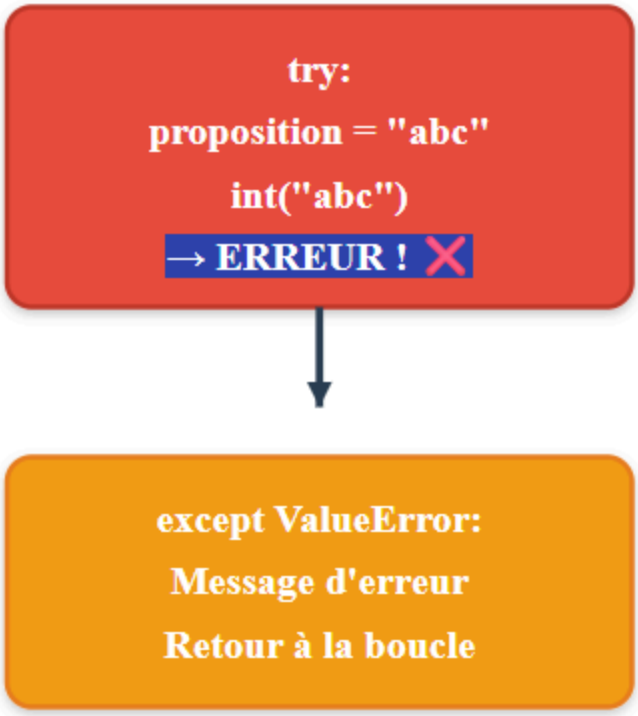
→ Si `int()` échoue (l'utilisateur tape "abc" par exemple), on attrape l'erreur.

Ce qui se passe vraiment

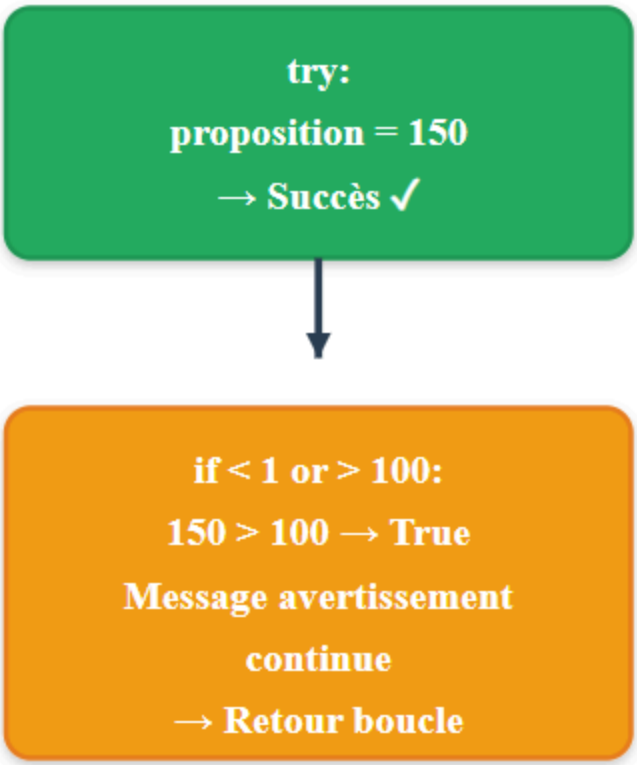
Scénario 1 : Nombre valide



Scénario 2 : Texte au lieu de nombre



Scénario 3 : Nombre hors plage



Résultat attendu

Cas 1 : Nombre hors plage

```
📌 Ton nombre : 150
⊗  Le nombre doit être entre 1 et 100 !

📌 Ton nombre : -5
⊗  Le nombre doit être entre 1 et 100 !
```

Cas 2 : Texte invalide

```
📌 Ton nombre : abc
☒ Erreur : Tu dois entrer un nombre entier !

📌 Ton nombre : vingt
☒ Erreur : Tu dois entrer un nombre entier !
```

Cas 3 : Jeu normal

```
📌 Ton nombre : 50
☒ C'est trop grand ! (Tentative n°1)

📌 Ton nombre : 25
☒ C'est trop petit ! (Tentative n°2)
```

Le problème

✓ Le jeu est robuste

✓ Il gère toutes les erreurs

☒ Mais on pourrait ajouter des statistiques et un meilleur affichage

Solution : Étape 6 - Version finale avec statistiques !

ÉTAPE 6 : Version finale avec statistiques

Objectif

Créer la version complète et professionnelle du jeu console avec statistiques de performance.

Le code (Version professionnelle)

```
# ═══════════════════════════════════════════
# JEU DE DEVINETTES - VERSION CONSOLE FINALE
# Auteur : C Premiers Pas
# Version : 1.0 (vCODE #4)
# ═══════════════════════════════════════════

import random

def jouer():
    """
    Fonction principale du jeu de devinettes.
    Le joueur doit deviner un nombre choisi par l'ordinateur.
    """

    # En-tête du jeu
    print("\n" + "="*50)
    print("🎮  JEU DE DEVINETTES - TROUVE LE NOMBRE MYSTÈRE")
    print("="*50)
    print("📋 Règles :")
    print("   • L'ordinateur choisit un nombre entre 1 et 100")
    print("   • Tu dois le deviner en un minimum de tentatives")
    print("   • L'ordinateur te donnera des indices")
    print("="*50 + "\n")

    # Configuration du jeu
    nombre_secret = random.randint(1, 100)
    tentatives = 0
    trouve = False
    historique = []  # Pour garder l'historique des propositions

    # Boucle de jeu
    while not trouve:

        try:
            # Demander une proposition
            proposition = int(input("📌 Ta proposition (1-100) : "))

            # Vérifier la plage
            if proposition < 1 or proposition > 100:
                print("⊗  Le nombre doit être entre 1 et 100 !\n")
                continue
```

```
# Vérifier si déjà proposé
if proposition in historique:
    print(f"⊗  Tu as déjà essayé {proposition} !\n")
    continue

# Enregistrer la tentative
tentatives += 1
historique.append(proposition)

# Comparer
if proposition > nombre_secret:
    diff = proposition - nombre_secret
    if diff > 20:
        indice = "BEAUCOUP trop grand"
    elif diff > 10:
        indice = "Trop grand"
    else:
        indice = "Un peu trop grand"
    print(f"⊠ {indice} ! (Tentative n°{tentatives})\n")

elif proposition < nombre_secret:
    diff = nombre_secret - proposition
    if diff > 20:
        indice = "BEAUCOUP trop petit"
    elif diff > 10:
        indice = "Trop petit"
    else:
        indice = "Un peu trop petit"
    print(f"⊠ {indice} ! (Tentative n°{tentatives})\n")

else:
    # Le joueur a trouvé !
    trouve = True
    print("\n" + "="*50)
    print("🎉 FÉLICITATIONS ! TU AS TROUVÉ !")
    print("="*50)
    print(f"✓ Le nombre mystère était : {nombre_secret}")
    print(f" Nombre de tentatives : {tentatives}")

    # Évaluation de la performance
    if tentatives <= 5:
        print("🏆 Performance : EXCELLENT !")
```

```
                print("   Tu es un(e) vrai(e) champion(ne) !")
            elif tentatives <= 8:
                print("🥇 Performance : TRÈS BIEN !")
                print("   Belle stratégie !")
            elif tentatives <= 12:
                print("🥈 Performance : BIEN !")
                print("   Tu t'améliores !")
            else:
                print("🥉 Performance : PAS MAL !")
                print("   Continue à t'entraîner !")

            print("="*50)

        except ValueError:
            print("☒ Erreur : Entre un nombre entier valide !\n")
        except KeyboardInterrupt:
            print("\n\n⊗  Partie abandonnée.")
            return

    # Afficher l'historique
    print(f"\n📜 Historique de tes essais : {historique}")

    # Proposer de rejouer
    print("\n" + "-"*50)
    rejouer = input("Veux-tu rejouer ? (oui/non) : ").strip().lower()
    if rejouer == "oui":
        jouer()  # Appel récursif
    else:
        print("\n👋 Merci d'avoir joué ! À bientôt sur C Premiers Pas !")


# Point d'entrée du programme
if __name__ == "__main__":
    try:
        jouer()
    except Exception as e:
        print(f"\n☒ Erreur inattendue : {e}")
```

Nouvelles fonctionnalités

1. 📜 Historique des tentatives

```
historique = []  # Liste vide au départ

# À chaque tentative :
historique.append(proposition)

# Vérifier si déjà proposé :
if proposition in historique:
    print("Tu as déjà essayé ce nombre !")
```

Résultat : Le joueur ne peut pas proposer deux fois le même nombre !

2. Indices de précision

```
if proposition > nombre_secret:
    diff = proposition - nombre_secret
    if diff > 20:
        indice = "BEAUCOUP trop grand"
    elif diff > 10:
        indice = "Trop grand"
    else:
        indice = "Un peu trop grand"
```

Résultat : Le joueur sait s'il est proche ou loin !

3. 🏆 Système d'évaluation

```
if tentatives <= 5:
    print("🏆 Performance : EXCELLENT !")
elif tentatives <= 8:
    print("🥇 Performance : TRÈS BIEN !")
# ...
```

Résultat : Le joueur reçoit une évaluation de sa performance !

4. 🔁 Rejouer sans relancer

```
rejouer = input("Veux-tu rejouer ? (oui/non) : ")
if rejouer == "oui":
    jouer()  # Rappel de la fonction (récursivité)
```

Résultat attendu (Partie complète)

```
==========================================
🎮  JEU DE DEVINETTES - TROUVE LE NOMBRE MYSTÈRE
==========================================
📋 Règles :
   • L'ordinateur choisit un nombre entre 1 et 100
   • Tu dois le deviner en un minimum de tentatives
   • L'ordinateur te donnera des indices
==========================================

📌 Ta proposition (1-100) : 50
☒ Trop grand ! (Tentative n°1)

📌 Ta proposition (1-100) : 25
☒ Un peu trop petit ! (Tentative n°2)

📌 Ta proposition (1-100) : 37
☒ Un peu trop grand ! (Tentative n°3)

📌 Ta proposition (1-100) : 31
☒ Un peu trop petit ! (Tentative n°4)

📌 Ta proposition (1-100) : 34

==========================================
🎉 FÉLICITATIONS ! TU AS TROUVÉ !
==========================================
✓ Le nombre mystère était : 34
 Nombre de tentatives : 5
🏆 Performance : EXCELLENT !
   Tu es un(e) vrai(e) champion(ne) !
==========================================

📜 Historique de tes essais : [50, 25, 37, 31, 34]

--------------------------------------------------
Veux-tu rejouer ? (oui/non) : oui
```

Vocabulaire technique (Partie 1)

Terme	Définition
random.randint()	Génère un nombre aléatoire entier dans une plage
booléen	Type de donnée qui ne peut être que True ou False
try/except	Structure pour gérer les erreurs sans planter
continue	Retourne immédiatement au début de la boucle
liste	Collection ordonnée de valeurs (ex: `[1, 2, 3]`)
récursivité	Fonction qui s'appelle elle-même

🎓 Conclusion

Le chemin parcouru

Il y a quelques heures, tu ne savais peut-être pas comment générer de l'aléatoire en Python. Maintenant, tu as créé :

✓ Un jeu de devinettes complet en console
✓ Un système de comptage et de statistiques
✓ Une gestion d'erreurs robuste
✓ Une boucle de jeu intelligente
✓ Un code propre et professionnel

Ce que tu as vraiment appris

Au-delà du code, tu as compris :

Penser en termes de jeu : Comment créer des règles et de la logique
Gérer l'aléatoire : Créer de l'imprévisibilité avec random
Structurer une boucle : Répéter jusqu'à une condition de victoire
Anticiper les erreurs : Valider les entrées utilisateur

Prochaine étape : Mercredi 20h

Présentation en binômes : Chaque équipe présente sa version améliorée du jeu (10 min par binôme).

Puis vCODE #5 (vendredi) : On va transformer ce jeu console en une belle application graphique avec Tkinter !

Challenge pour mercredi

Avant la présentation, améliore ton jeu avec **AU MOINS 1 fonctionnalité** :

Idées d'amélioration :

- Niveaux de difficulté (facile/moyen/difficile)
- Chronomètre pour mesurer le temps
- Sauvegarde des meilleurs scores
- ASCII art pour les messages
- Statistiques détaillées (moyenne, record)
- Messages personnalisés selon la performance

Tu as toutes les compétences pour le faire !

> *"Un jeu que tu as créé toi-même, même simple, vaut mieux que 100 jeux que tu as seulement téléchargés. Tu n'es plus un joueur, tu es un créateur."*
>
> **— C Premiers Pas**

Document réalisé par C Premiers Pas

Contact : NOUS CONTACTER

Rejoignez notre communauté

LINKEDIN
WHATSAPP
DISCORD