

CALCULATRICE INTERACTIVE EN PYTHON

De zéro à une calculatrice qui gère tout



C Premiers Pas - vCODE #3

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



Table des matières

- Introduction : Pourquoi une calculatrice ?
- Les bases Python nécessaires
- Construction étape par étape
 - Étape 1 : Addition simple
 - Étape 2 : Les quatre opérations
 - Étape 3 : Choisir l'opération (IF/ELIF/ELSE)
 - Étape 4 : Gérer les erreurs comme un pro
 - Étape 5 : Répéter avec WHILE
 - Étape 6 : Organiser avec une fonction
- Version finale complète
- Aller plus loin
- Récapitulatif : Ce que tu as appris
- Conclusion

Introduction : Pourquoi une calculatrice ?

Le problème concret

Tu utilises ton téléphone pour faire des calculs ? Ton ordinateur a une calculatrice que tu ouvres 10 fois par jour ? **Et si tu pouvais créer LA TIENNE**, mais en mieux ?

Une calculatrice qui :

- ⇒ Ne plante jamais
- ⇒ Gère toutes les erreurs
- ⇒ Peut faire autant de calculs que tu veux sans redémarrer
- ⇒ Te dit exactement ce qui ne va pas si tu te trompes
- ⇒ **Et surtout : que TU as créée de A à Z**

La solution : Ce qu'on va construire






Aujourd'hui, tu vas créer une **calculatrice intelligente** qui :

CALCULATRICE INTELLIGENTE
✓ Addition (+)
✓ Soustraction (-)
✓ Multiplication (×)
✓ Division (÷)
✓ Gère la division par zéro
✓ Détecte les opérations invalides
✓ Répète autant que tu veux
✓ Code propre et professionnel



Ce qu'on va construire ensemble - Une calculatrice qui gère tout !

Ce que tu vas apprendre

Compétence	Ce que ça t'apporte
 Conditions (if/elif/else)	Faire des choix intelligents dans ton code
 Boucle while	Répéter des actions jusqu'à une condition
 Gestion d'erreurs	Anticiper et gérer les problèmes
 Opérateurs logiques	Combiner plusieurs conditions
 Fonctions	Organiser ton code comme un pro

Résultat final attendu

À la fin de cette session, tu auras :

Une calculatrice complète qui fonctionne

Compris comment prendre des décisions dans ton code

Maîtrisé la gestion d'erreurs (division par zéro, etc.)

Appris à répéter des actions intelligemment

Écrit du code propre et professionnel

Et surtout : tu sauras POURQUOI ça marche, pas juste comment copier-coller !



Les bases Python nécessaires

Avant de commencer, révisons rapidement les concepts qu'on a déjà vus (si tu es nouveau, pas de panique, on explique tout !) :

Variables : Des boîtes pour stocker des valeurs

```
nom = "Pascal"  
age = 25
```

Analogie : Une variable, c'est comme une boîte avec une étiquette. L'étiquette = le nom de la variable. Le contenu = la valeur.

Input : Demander à l'utilisateur

```
prenom = input("Quel est ton prénom ? ")
```

Ce qui se passe : Python pose la question, attend que tu tapes quelque chose, puis stocke ta réponse dans la variable `prenom`.

Print : Afficher du texte

```
print("Bienvenue dans ma calculatrice !")
```

Simple : Affiche le texte entre guillemets dans la console.

Types de données : int, float, string

Type	Exemple	Utilisation
int	5 , 42 , -10	Nombres entiers
float	3.14 , 9.99 , -2.5	Nombres décimaux
string	"Bonjour" , "+"	Texte

Point clé : Quand tu utilises `input()` , Python te donne TOUJOURS un string. Pour faire des calculs, il faut convertir en nombre !

```
age = input("Ton âge : ") # age = "25" (string)
age = int(age)            # age = 25 (int)
```

⊗ PIÈGE CLASSIQUE

```
a = "5"
b = "3"
resultat = a + b # Résultat : "53" (concaténation, pas addition !)
```

✓ Solution :

```
a = int("5")
b = int("3")
resultat = a + b # Résultat : 8 (addition mathématique !)
```

Construction étape par étape

On va construire notre calculatrice **étape par étape**. Chaque étape ajoute UNE nouveauté. Tu vas voir, c'est plus simple qu'il n'y paraît !

ÉTAPE 1 : Addition simple

Objectif

Créer une calculatrice qui additionne deux nombres. C'est notre point de départ !

Le code (Version débutant)

```
# Calculatrice v1.0 - Addition simple

# Demander le premier nombre
nombre1 = input("Entre le premier nombre : ")
nombre1 = float(nombre1)

# Demander le deuxième nombre
nombre2 = input("Entre le deuxième nombre : ")
nombre2 = float(nombre2)

# Faire l'addition
resultat = nombre1 + nombre2

# Afficher le résultat
print("Le résultat est :", resultat)
```

Explication ligne par ligne

Ligne 4 :

```
nombre1 = input("Entre le premier nombre : ")
```

→ Python affiche "Entre le premier nombre : " et attend que tu tapes quelque chose. Ce que tu tapes est stocké dans `nombre1` **sous forme de texte** (string).

Ligne 5 :

```
nombre1 = float(nombre1)
```

→ On convertit le texte en nombre décimal (float). Pourquoi ? Parce qu'on ne peut pas additionner du texte !

🤔 Pourquoi float() et pas int() ?

Fonction	Accepte	Exemple
int()	Nombres entiers uniquement	int("5") ✓ / int("5.5") ✗
float()	Nombres entiers ET décimaux	float("5") ✓ / float("5.5") ✓

Conclusion : float() est plus flexible, donc on l'utilise pour une calculatrice !

Lignes 8-9 :

```
nombre2 = input("Entre le deuxième nombre : ")  
nombre2 = float(nombre2)
```

→ Même chose pour le deuxième nombre.

Ligne 12 :

```
resultat = nombre1 + nombre2
```

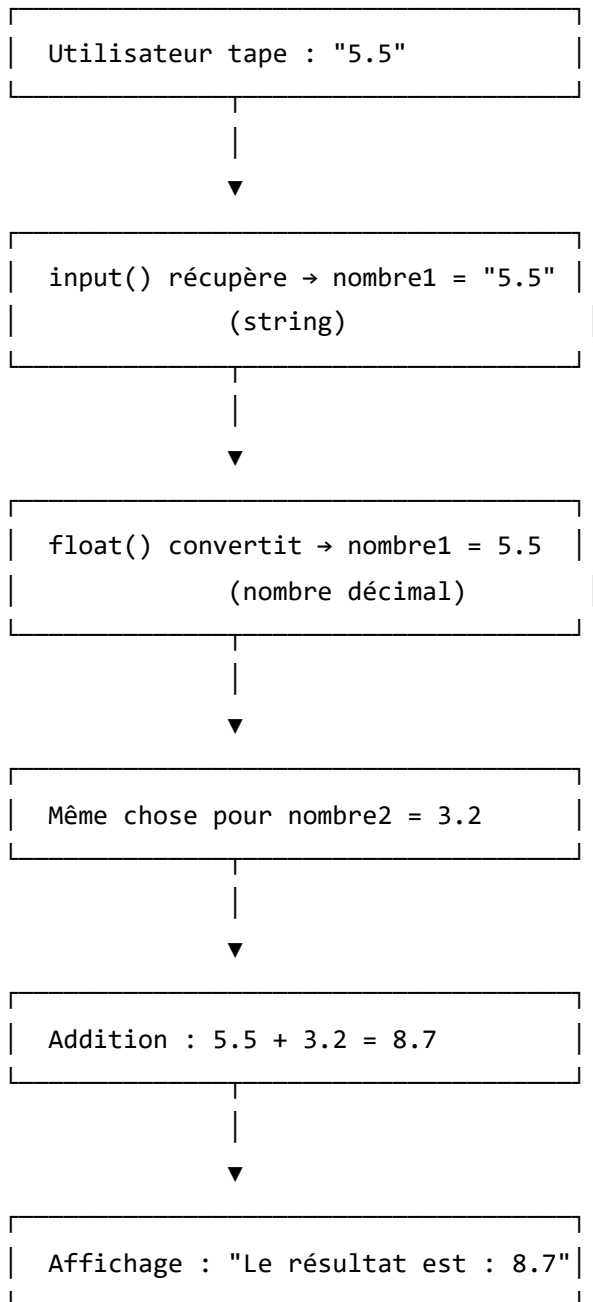
→ Python additionne les deux nombres et stocke le résultat dans une nouvelle variable resultat .

Ligne 15 :

```
print("Le résultat est :", resultat)
```

→ On affiche le résultat. La virgule dans print() permet de mettre plusieurs choses bout à bout.

Ce qui se passe vraiment



Résultat attendu

Entre le premier nombre : 5.5
Entre le deuxième nombre : 3.2
Le résultat est : 8.7

Le problème

✓ Ça marche pour l'addition

✗ Mais on ne peut faire QUE des additions

✗ Et si je veux soustraire ? Multiplier ? Diviser ?

Solution : On passe à l'étape 2 !

Vocabulaire technique

Terme	Définition simple
float()	Fonction qui convertit du texte en nombre décimal
variable	Boîte qui stocke une valeur (nombre, texte, etc.)
conversion	Changer le type d'une donnée (texte → nombre)

ÉTAPE 2 : Les quatre opérations

Objectif

Maintenant, on veut pouvoir faire les 4 opérations de base : +, -, ×, ÷

Le code (Version débutant)

```
# Calculatrice v2.0 - Quatre opérations

# Demander les nombres
nombre1 = float(input("Premier nombre : "))
nombre2 = float(input("Deuxième nombre : "))

# Faire les quatre opérations
addition = nombre1 + nombre2
soustraction = nombre1 - nombre2
multiplication = nombre1 * nombre2
division = nombre1 / nombre2

# Afficher tous les résultats
print("Addition :", addition)
print("Soustraction :", soustraction)
print("Multiplication :", multiplication)
print("Division :", division)
```

Explication ligne par ligne

Ligne 4 :

```
nombre1 = float(input("Premier nombre : "))
```

→ **NOUVEAU !** On fait la conversion en une seule ligne. C'est équivalent à :

```
nombre1 = input("Premier nombre : ")
nombre1 = float(nombre1)
```

Mais en plus court !

Lignes 8-11 :

```
addition = nombre1 + nombre2
soustraction = nombre1 - nombre2
multiplication = nombre1 * nombre2
division = nombre1 / nombre2
```

→ On fait les 4 opérations et on stocke chaque résultat dans une variable différente.



Les 4 opérateurs mathématiques en Python

Les opérateurs mathématiques en Python

Opération	Symbole Python	Exemple	Résultat
Addition	+	5 + 3	8
Soustraction	-	5 - 3	2
Multiplication	*	5 * 3	15
Division	/	5 / 2	2.5

⚠ **Attention** : En Python, la division **donne toujours un float** (nombre décimal), même si le résultat est un nombre entier !

```
10 / 2 # Résultat : 5.0 (pas 5)
```

Ce qui se passe vraiment

Analogie : Imagine que tu as deux piles de pièces de monnaie.

- **Addition (+)** : Tu rassembles les deux piles
- **Soustraction (-)** : Tu enlèves la deuxième pile de la première
- **Multiplication (*)** : Tu dupliques ta première pile autant de fois que le deuxième nombre
- **Division (/)** : Tu divises ta première pile en parts égales

Résultat attendu

Premier nombre : 10

Deuxième nombre : 3

Addition : 13.0

Soustraction : 7.0

Multiplication : 30.0

Division : 3.3333333333333335

Le problème

✓ On peut faire les 4 opérations

✗ Mais on fait TOUTES les opérations à chaque fois

✗ Je veux juste une addition ? Dommage, tu auras aussi les 3 autres !

Solution : Il faut pouvoir **choisir** quelle opération faire. On passe à l'étape 3 !

Vocabulaire technique

Terme	Définition simple
opérateur	Symbole qui effectue une opération (+, -, *, /)
expression	Calcul qui produit une valeur (ex: 5 + 3)

ÉTAPE 3 : Choisir l'opération (IF/ELIF/ELSE)

Objectif

Laisser l'utilisateur **choisir** quelle opération il veut faire. C'est là qu'interviennent les **conditions** !

Le code (Version débutant)

```
# Calculatrice v3.0 - Avec choix d'opération

# Demander les nombres
nombre1 = float(input("Premier nombre : "))
nombre2 = float(input("Deuxième nombre : "))

# Demander l'opération souhaitée
operation = input("Quelle opération ? (+, -, *, /) : ")

# Faire le calcul selon l'opération choisie
if operation == "+":
    resultat = nombre1 + nombre2
    print("Résultat :", resultat)
elif operation == "-":
    resultat = nombre1 - nombre2
    print("Résultat :", resultat)
elif operation == "*":
    resultat = nombre1 * nombre2
    print("Résultat :", resultat)
elif operation == "/":
    resultat = nombre1 / nombre2
    print("Résultat :", resultat)
else:
    print("Opération invalide !")
```

Explication ligne par ligne

Ligne 8 :

```
operation = input("Quelle opération ? (+, -, *, /) : ")
```

→ On demande à l'utilisateur de choisir une opération. Il va taper + , - , * ou / .

Ligne 11 :

```
if operation == "+":
```

→ **NOUVEAU CONCEPT : LA CONDITION !**

⊗ PIÈGE CLASSIQUE : = vs ==

Symbole	Utilisation	Exemple
=	Assignment (donner une valeur)	age = 25
==	Comparaison (vérifier si égal)	if age == 25:

✗ ERREUR

```
if operation = "+": # Erreur de syntaxe !
```

✓ CORRECT

```
if operation == "+": # Compare si operation égale "+"
```

Ce qui se passe :

- Python vérifie si `operation` est égal à `"+"`
- Si OUI → il exécute le bloc indenté (lignes 12-13)
- Si NON → il passe à la condition suivante

Ligne 14 :

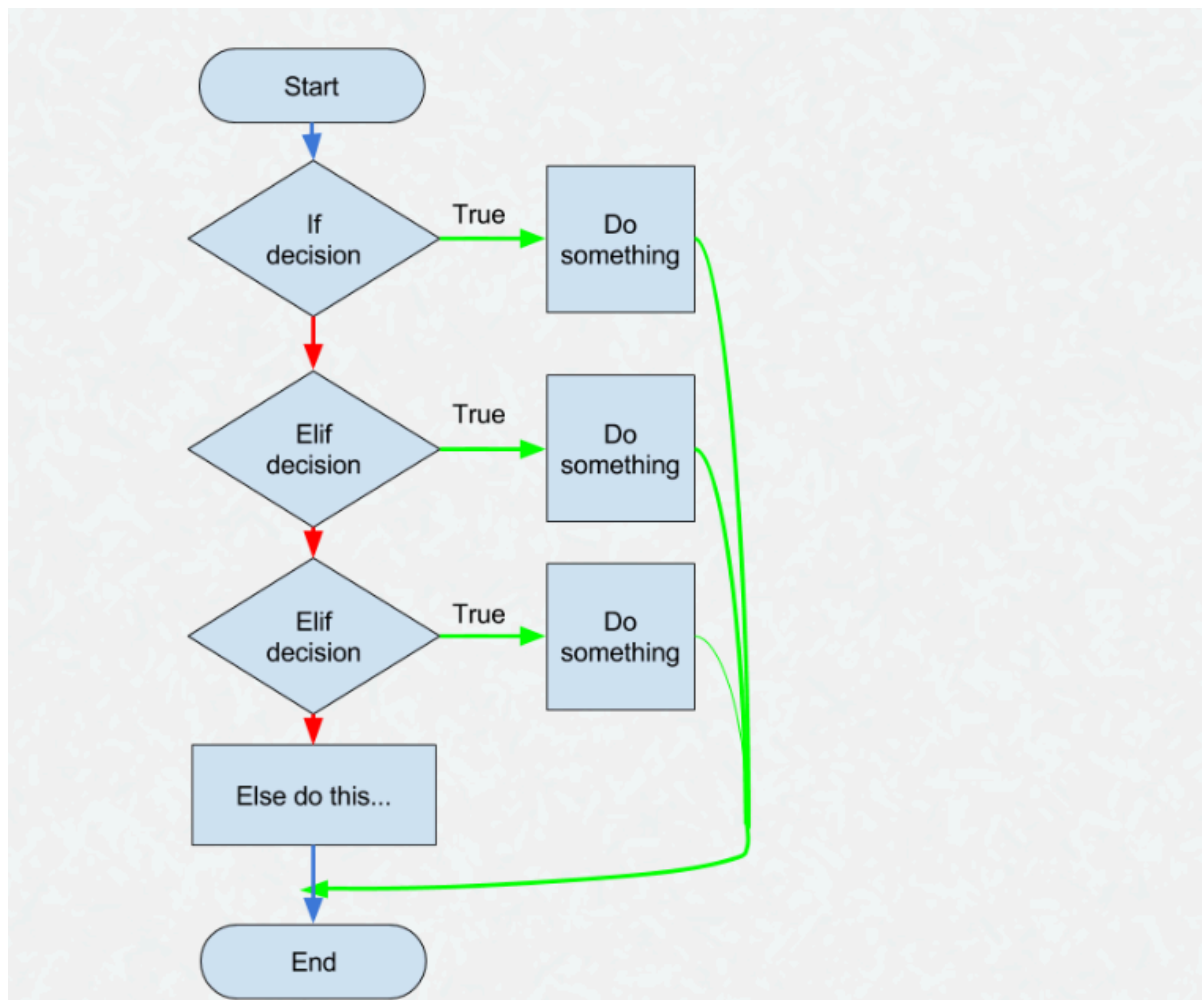
```
elif operation == "-":
```

→ **elif** = "sinon si" en français. Si la première condition est fausse, Python teste celle-ci.

Ligne 23 :

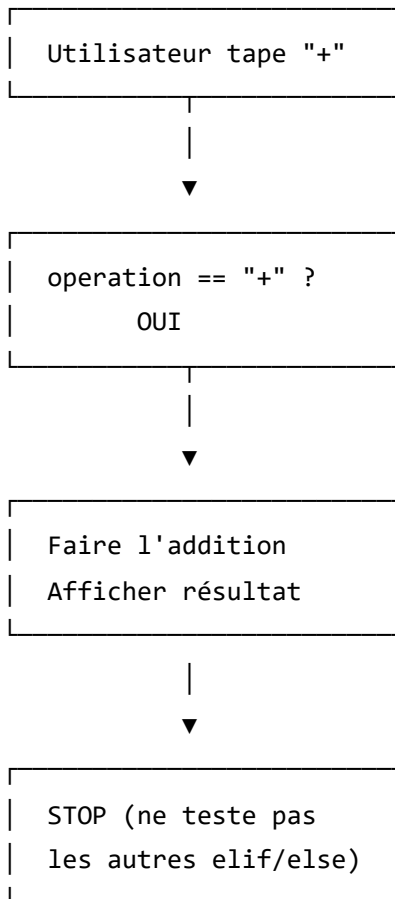
```
else:
```

→ **else** = "sinon". Si AUCUNE des conditions précédentes n'est vraie, Python exécute ce bloc.



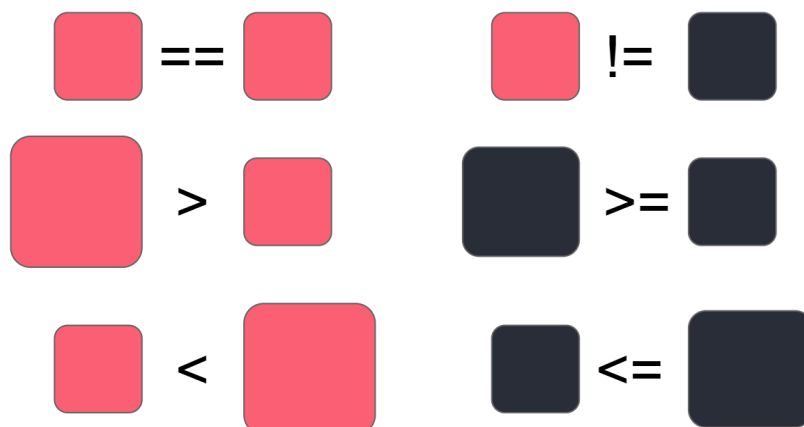
Comment Python prend des décisions

Ce qui se passe vraiment : L'organigramme de décision



Point crucial : Dès qu'une condition est vraie, Python exécute le bloc correspondant et **IGNORE TOUTES LES AUTRES CONDITIONS !**

Les opérateurs de comparaison



Tous les opérateurs de comparaison en Python

Opérateur	Signification	Exemple	Résultat
==	Égal à	5 == 5	True
!=	Différent de	5 != 3	True
<	Plus petit que	3 < 5	True
>	Plus grand que	5 > 3	True
<=	Plus petit ou égal	5 <= 5	True
>=	Plus grand ou égal	5 >= 3	True

Structure IF / ELIF / ELSE

```

if condition1:
    # Code si condition1 est vraie
elif condition2:
    # Code si condition1 est fausse ET condition2 est vraie
elif condition3:
    # Code si condition1 et condition2 sont fausses ET condition3 est vraie
else:
    # Code si TOUTES les conditions sont fausses

```

Règles importantes :

- if : obligatoire, toujours en premier
- elif : optionnel, autant que tu veux
- else : optionnel, toujours en dernier
- **L'indentation (décalage) est OBLIGATOIRE** en Python !

Résultat attendu

```

Premier nombre : 15
Deuxième nombre : 3
Quelle opération ? (+, -, *, /) : *
Résultat : 45.0

```

Ou si on tape une opération invalide :

Premier nombre : 10
Deuxième nombre : 5
Quelle opération ? (+, -, *, /) : %
Opération invalide !

Le problème

- ✓ On peut choisir l'opération
- ✓ On détecte les opérations invalides
- ✗ Mais si je divise par zéro ? **CRASH !**
- ✗ Python affiche une erreur moche et s'arrête

Solution : Il faut gérer les erreurs ! Étape 4 !

Vocabulaire technique

Terme	Définition simple
condition	Expression qui donne True (vrai) ou False (faux)
if	"Si" - exécute le code si la condition est vraie
elif	"Sinon si" - teste une autre condition
else	"Sinon" - exécute le code si toutes les conditions sont fausses
indentation	Décalage à gauche (avec Tab ou espaces) qui définit les blocs de code
True / False	Valeurs booléennes (vrai / faux)

ÉTAPE 4 : Gérer les erreurs comme un pro

Objectif

Anticiper et gérer les erreurs (division par zéro, opération invalide) pour que notre calculatrice ne plante JAMAIS !

Le code (Version débutant)

```
# Calculatrice v4.0 - Avec gestion d'erreurs

# Demander les nombres
nombre1 = float(input("Premier nombre : "))
nombre2 = float(input("Deuxième nombre : "))

# Demander l'opération
operation = input("Quelle opération ? (+, -, *, /) : ")

# Vérifier si l'opération est valide
if operation == "+" or operation == "-" or operation == "*" or operation == "/":
    # L'opération est valide

    # Cas spécial : division par zéro
    if operation == "/" and nombre2 == 0:
        print("✗ Erreur : Division par zéro impossible !")
    else:
        # Faire le calcul
        if operation == "+":
            resultat = nombre1 + nombre2
        elif operation == "-":
            resultat = nombre1 - nombre2
        elif operation == "*":
            resultat = nombre1 * nombre2
        elif operation == "/":
            resultat = nombre1 / nombre2

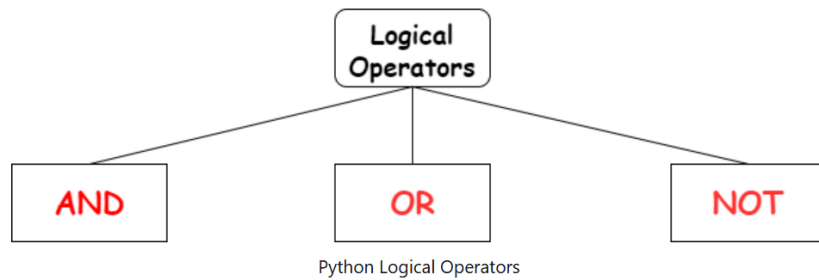
        print("✓ Résultat :", resultat)
else:
    print("✗ Opération invalide ! Utilise : +, -, * ou /")
```

Explication ligne par ligne

Ligne 11 :

```
if operation == "+" or operation == "-" or operation == "*" or operation == "/":
```

→ **NOUVEAU** : Opérateur logique OR (ou) !



Les opérateurs logiques : AND, OR, NOT

Ce qui se passe :

- or = "ou" en français
- La condition est vraie si **AU MOINS UNE** des sous-conditions est vraie
- Si operation vaut "+", "-", "*" OU "/" → la condition est vraie

Ligne 15 :

```
if operation == "/" and nombre2 == 0:
```

→ **NOUVEAU : Opérateur logique AND (et) !**

Ce qui se passe :

- and = "et" en français
- La condition est vraie si **TOUTES** les sous-conditions sont vraies
- On divise (operation == "/") **ET** le diviseur est zéro (nombre2 == 0) → DANGER !

Les opérateurs logiques

Opérateur	Signification	Condition vraie quand...	Exemple
and	ET	Toutes les conditions sont vraies	age > 18 and permis == True
or	OU	Au moins une condition est vraie	jour == "samedi" or jour == "dimanche"
not	NON	Inverse la condition	not est_mineur

Exemples concrets :

```
# AND - Il faut que les DEUX soient vrais
age = 20
permis = True
if age >= 18 and permis == True:
    print("Tu peux conduire !") # ✓ Affiché (les 2 conditions sont vraies)

# OR - Il suffit qu'UN SEUL soit vrai
jour = "samedi"
if jour == "samedi" or jour == "dimanche":
    print("C'est le weekend !") # ✓ Affiché (une condition est vraie)

# NOT - Inverse le résultat
pluie = True
if not pluie:
    print("Pas besoin de parapluie !") # ✓ Affiché (not False = True)
```

Ce qui se passe vraiment

Scénario 1 : Division normale

Utilisateur : 10 / 5

1. Opération valide ? OUI (/)
2. Division par zéro ? NON ($5 \neq 0$)
3. Calcul : $10 / 5 = 2.0$
4. Affichage : ✓ Résultat : 2.0

Scénario 2 : Division par zéro

Utilisateur : 10 / 0

1. Opération valide ? OUI (/)
2. Division par zéro ? OUI ($0 == 0$)
3. Message : ✗ Erreur : Division par zéro impossible !
4. Pas de calcul, pas de crash !

Scénario 3 : Opération invalide

Utilisateur : 10 % 5

1. Opération valide ? NON (% n'est pas dans la liste)
2. Message : **✗** Opération invalide !
3. On ne va même pas regarder les nombres

Résultat attendu

Cas 1 : Calcul normal

Premier nombre : 20
Deuxième nombre : 4
Quelle opération ? (+, -, *, /) : /
✓ Résultat : 5.0

Cas 2 : Division par zéro

Premier nombre : 10
Deuxième nombre : 0
Quelle opération ? (+, -, *, /) : /
✗ Erreur : Division par zéro impossible !

Cas 3 : Opération invalide

Premier nombre : 5
Deuxième nombre : 3
Quelle opération ? (+, -, *, /) : %
✗ Opération invalide ! Utilise : +, -, * ou /

Le problème

- ✓ On gère les erreurs
- ✓ Notre calculatrice ne plante plus
- ✗** Mais après un calcul, le programme s'arrête
- ✗** Je veux faire plusieurs calculs sans relancer le programme !

Solution : Il faut une boucle ! Étape 5 !

Vocabulaire technique

Terme	Définition simple
opérateur logique	Mot-clé qui combine des conditions (and, or, not)
and	"ET" - toutes les conditions doivent être vraies
or	"OU" - au moins une condition doit être vraie
not	"NON" - inverse une condition
gestion d'erreurs	Anticiper les problèmes et les gérer proprement

ÉTAPE 5 : Répéter avec WHILE

Objectif

Permettre de faire autant de calculs qu'on veut sans relancer le programme. On va utiliser une **boucle while** !

Le code (Version débutant)

```
# Calculatrice v5.0 - Avec boucle while

print("\n\n🧮 CALCULATRICE INTERACTIVE")
print("::=" * 26)

# Variable pour contrôler la boucle
continuer = "oui"

# Boucle tant que l'utilisateur veut continuer
while continuer == "oui":

    # Demander les nombres
    nombre1 = float(input("\nPremier nombre : "))
    nombre2 = float(input("Deuxième nombre : "))

    # Demander l'opération
    operation = input("Quelle opération ? (+, -, *, /) : ")

    # Vérifier si l'opération est valide
    if operation == "+" or operation == "-" or operation == "*" or operation == "/":

        # Cas spécial : division par zéro
        if operation == "/" and nombre2 == 0:
            print("❌ Erreur : Division par zéro impossible !")
        else:
            # Faire le calcul
            if operation == "+":
                resultat = nombre1 + nombre2
            elif operation == "-":
                resultat = nombre1 - nombre2
            elif operation == "*":
                resultat = nombre1 * nombre2
            elif operation == "/":
                resultat = nombre1 / nombre2

            print("✓ Résultat :", resultat)
        else:
            print("❌ Opération invalide ! Utilise : +, -, * ou /")

    # Demander si on veut continuer
    continuer = input("\nVeux-tu faire un autre calcul ? (oui/non) : ")
```

```
print("\n👋 Merci a très bientôt !")
```

Explication ligne par ligne

Ligne 7 :

```
continuer = "oui"
```

→ On crée une variable qui contrôle la boucle. Au départ, elle vaut "oui" .

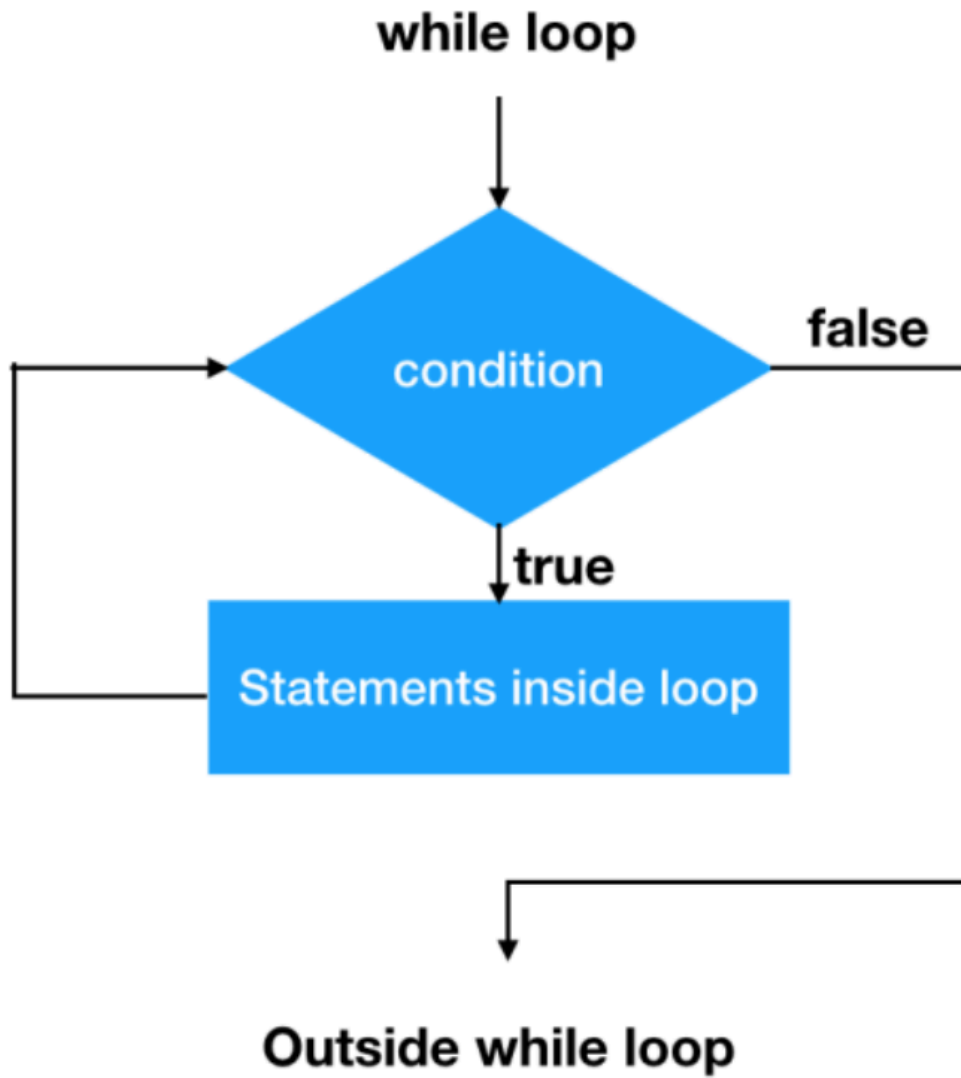
Ligne 10 :

```
while continuer == "oui":
```

→ **NOUVEAU : LA BOUCLE WHILE !*

Ce qui se passe :

1. Python vérifie si `continuer` est égal à "oui"
2. Si OUI → il exécute tout le code indenté sous le `while`
3. À la fin du bloc, il retourne à la ligne 10 et **revérifie la condition**
4. Si la condition est toujours vraie → il recommence
5. Si la condition devient fausse → il sort de la boucle et continue après



Comment fonctionne une boucle while

Ligne 40 :

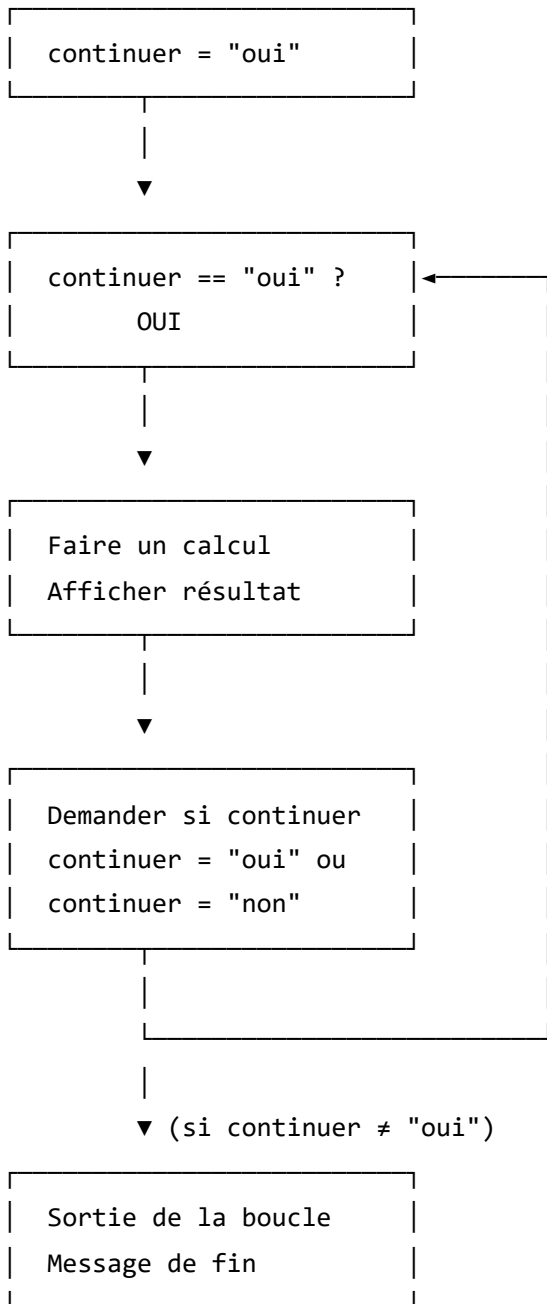
```
continuer = input("\nVeux-tu faire un autre calcul ? (oui/non) : ")
```

→ On demande à l'utilisateur s'il veut continuer. Sa réponse **modifie la variable** `continuer` .

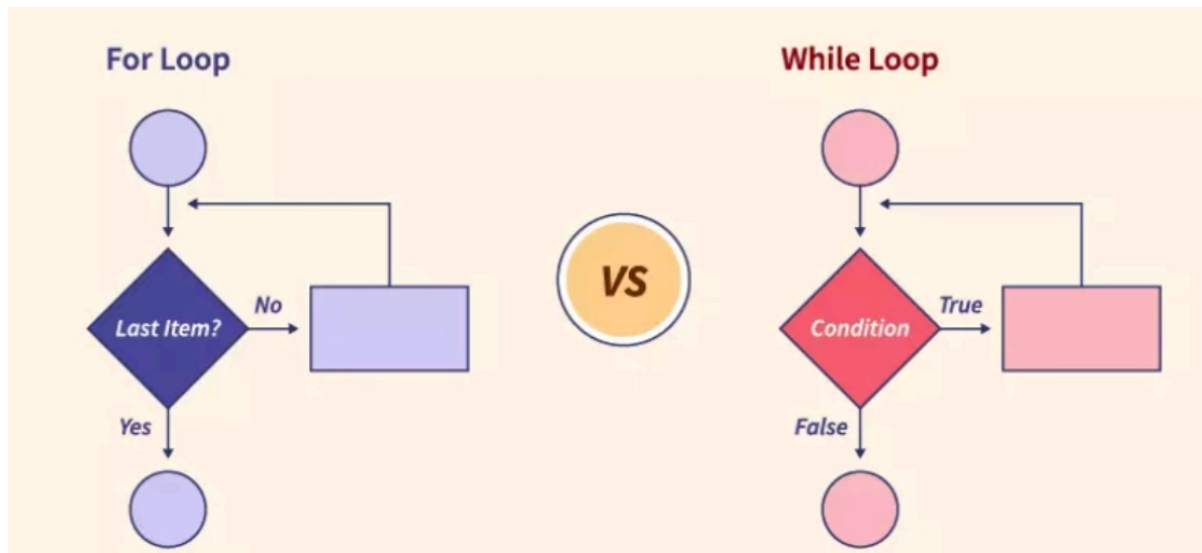
Point crucial :

- Si l'utilisateur tape "oui" → la boucle recommence
- Si l'utilisateur tape "non" (ou n'importe quoi d'autre) → la boucle s'arrête

Ce qui se passe vraiment



FOR vs WHILE : Quelle différence ?



Quand utiliser for et quand utiliser while ?

Boucle	Utilisation	Exemple d'usage
for	Quand tu sais combien de fois répéter	"Répète 10 fois", "Pour chaque élément de la liste"
while	Quand tu répètes jusqu'à une condition	"Répète tant que l'utilisateur dit oui", "Répète tant qu'il y a des erreurs"

Exemples concrets :

```
# FOR - Nombre de répétitions connu
for i in range(5):
    print("Tour", i) # On sait qu'on va faire 5 tours

# WHILE - On répète jusqu'à une condition
essais = 0
mot_de_passe = ""
while mot_de_passe != "1234": # On répète jusqu'à trouver le bon mot de passe
    mot_de_passe = input("Mot de passe : ")
    essais += 1
```

Pour notre calculatrice : On utilise `while` car on ne sait pas combien de calculs l'utilisateur voudra faire !

Résultat attendu

△ CALCULATRICE INTERACTIVE

=====

Premier nombre : 10

Deuxième nombre : 5

Quelle opération ? (+, -, *, /) : +

✓ Résultat : 15.0

Veux-tu faire un autre calcul ? (oui/non) : oui

Premier nombre : 20

Deuxième nombre : 4

Quelle opération ? (+, -, *, /) : /

✓ Résultat : 5.0

Veux-tu faire un autre calcul ? (oui/non) : non

👋 Merci a très bientôt !

Le problème

✓ On peut faire autant de calculs qu'on veut

✓ Tout fonctionne bien

✗ Mais le code commence à être long et répétitif

✗ Difficile de le réutiliser ou de le modifier

Solution : Organisons tout ça dans une fonction ! Étape 6 !

Vocabulaire technique

Terme	Définition simple
boucle while	Répète un bloc de code tant qu'une condition est vraie
condition de sortie	Ce qui fait arrêter la boucle
itération	Un passage dans la boucle (un "tour")

ÉTAPE 6 : Organiser avec une fonction

Objectif

Créer une fonction propre et réutilisable pour notre calculatrice. C'est la touche professionnelle finale !

Le code (Version professionnelle)

```
# Calculatrice v6.0 - Version finale avec fonction

def calculatrice():
    """
    Fonction principale de la calculatrice interactive.
    Permet de faire plusieurs calculs à la suite.
    """

    print("△ CALCULATRICE INTERACTIVE")
    print("::=" * 26)

    continuer = "oui"

    while continuer == "oui":

        # Demander les nombres
        nombre1 = float(input("\nPremier nombre : "))
        nombre2 = float(input("Deuxième nombre : "))

        # Demander l'opération
        operation = input("Quelle opération ? (+, -, *, /) : ")

        # Vérifier si l'opération est valide
        if operation in ["+", "-", "*", "/"]:

            # Cas spécial : division par zéro
            if operation == "/" and nombre2 == 0:
                print("✗ Erreur : Division par zéro impossible !")
            else:
                # Faire le calcul
                if operation == "+":
                    resultat = nombre1 + nombre2
                elif operation == "-":
                    resultat = nombre1 - nombre2
                elif operation == "*":
                    resultat = nombre1 * nombre2
                elif operation == "/":
                    resultat = nombre1 / nombre2

                print(f"✓ Résultat : {nombre1} {operation} {nombre2} = {resultat}")
        else:
```

```
print("❌ Opération invalide ! Utilise : +, -, * ou /")
```

```
# Demander si on veut continuer
```

```
continuer = input("\nVeux-tu faire un autre calcul ? (oui/non) : ").lower()
```

```
print("\n👋 Merci a très bientôt!")
```

```
# Lancer la calculatrice
```

```
if __name__ == "__main__":
```

```
    calculatrice()
```

Explication ligne par ligne

Ligne 3 :

```
def calculatrice():
```

→ **NOUVEAU : DÉFINITION D'UNE FONCTION !**

Ce qui se passe :

- `def` = "définis une fonction"
- `calculatrice` = nom de la fonction
- `()` = parenthèses pour les paramètres (ici, pas de paramètres)
- `:` = début du bloc de code de la fonction



Anatomie d'une fonction Python

Lignes 4-7 :

```
"""
Fonction principale de la calculatrice interactive.
Permet de faire plusieurs calculs à la suite.
"""
```

→ **Docstring** : Documentation de la fonction. Explique ce qu'elle fait.

Ligne 24 :

```
if operation in ["+", "-", "*", "/"]:
```

→ **NOUVEAU : Opérateur IN !**

Ce qui se passe :

- `in` vérifie si une valeur est présente dans une liste
- Plus court et plus lisible que `operation == "+"` or `operation == "-"` or ...

Ligne 39 :

```
print(f"✓ Résultat : {nombre1} {operation} {nombre2} = {resultat}")
```

→ **NOUVEAU : f-string (formatted string) !**

Ce qui se passe :

- Le `f` avant les guillemets active le formatage
- Les `{}` sont remplacées par les valeurs des variables
- Résultat : ✓ Résultat : 10 + 5 = 15.0

Ligne 44 :

```
continuer = input("...").lower()
```

→ `.lower()` convertit la réponse en minuscules

Pourquoi ?

- Si l'utilisateur tape "OUI", "Oui", "oUi" → tout devient "oui"
- Ça rend notre code plus flexible !

Ligne 51 :





```
if __name__ == "__main__":
```

→ **BONNE PRATIQUE PROFESSIONNELLE !**

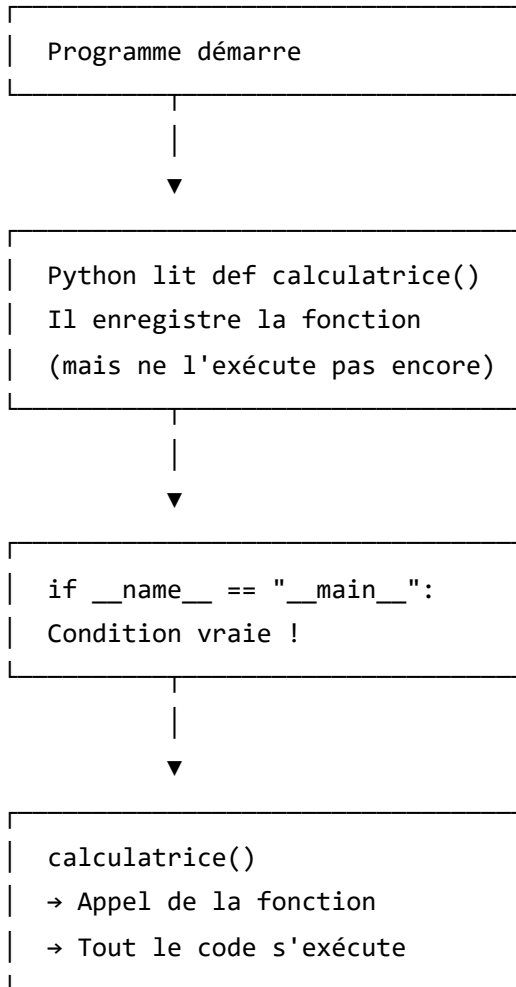
Ce qui se passe :

- Ce bloc ne s'exécute QUE si on lance ce fichier directement
- Si quelqu'un importe notre code dans un autre fichier, la calculatrice ne se lance pas automatiquement
- C'est comme ça que travaillent les vrais développeurs !

Pourquoi utiliser des fonctions ?

Avantage	Explication	Exemple
 Organisation	Code structuré et facile à lire	Tout le code de la calculatrice au même endroit
 Réutilisabilité	Appeler la fonction plusieurs fois	<code>calculatrice()</code> lance la calculatrice
 Maintenance	Plus facile de corriger ou améliorer	Tout est à un seul endroit
 Clarté	Nom de fonction = documentation	<code>calculatrice()</code> → on sait ce que ça fait

Ce qui se passe vraiment



Résultat attendu

▢ CALCULATRICE INTERACTIVE

== == == == == == == == == == == ==

Premier nombre : 15

Deuxième nombre : 3

Quelle opération ? (+, -, *, /) : *

✓ Résultat : 15.0 * 3.0 = 45.0

Veux-tu faire un autre calcul ? (oui/non) : OUI

Premier nombre : 100

Deuxième nombre : 4

Quelle opération ? (+, -, *, /) : /

✓ Résultat : 100.0 / 4.0 = 25.0

Veux-tu faire un autre calcul ? (oui/non) : non

👋 Merci a très bientôt !

Vocabulaire technique

Terme	Définition simple
fonction	Bloc de code réutilisable qu'on peut appeler par son nom
def	Mot-clé pour définir une fonction
docstring	Documentation d'une fonction (entre triple guillemets)
in	Opérateur qui vérifie si une valeur est dans une liste
f-string	Chaîne formatée qui permet d'insérer des variables
lower()	Méthode qui convertit un texte en minuscules



Version finale complète

Voici le code complet, commenté et prêt à l'emploi. Tu peux le copier et l'utiliser !

```

# =====
# CALCULATRICE INTERACTIVE EN PYTHON
# Auteur : C Premiers Pas
# Version : 1.0 (vCODE #3)
# =====

def calculatrice():
    """
    Calculatrice interactive complète avec gestion d'erreurs.

    Fonctionnalités :
    - Addition, soustraction, multiplication, division
    - Gestion de la division par zéro
    - Détection des opérations invalides
    - Boucle pour faire plusieurs calculs
    - Interface utilisateur claire
    """

    # Afficher l'en-tête
    print("\n" + "=="*26)
    print("△  CALCULATRICE INTERACTIVE")
    print("=="*26)
    print("Opérations disponibles : +, -, *, /")
    print("○"*36 + "\n")

    # Variable de contrôle de la boucle
    continuer = "oui"

    # Compteur de calculs effectués
    nombre_calculs = 0

    # Boucle principale
    while continuer.lower() == "oui":

        try:
            # Demander les nombres
            print(f"\n Calcul n°{nombre_calculs + 1}")
            print("-" * 25)
            nombre1 = float(input("Premier nombre : "))
            nombre2 = float(input("Deuxième nombre : "))

            # Demander l'opération
            operation = input("Opération (+, -, *, /) : ").strip()

```

```

# Vérifier si l'opération est valide
if operation in ["+", "-", "*", "/"]:

    # Cas spécial : division par zéro
    if operation == "/" and nombre2 == 0:
        print("\n✗ ERREUR : Division par zéro impossible !")
        print(" Conseil : Le diviseur doit être différent de zéro.")

    else:
        # Effectuer le calcul selon l'opération
        if operation == "+":
            resultat = nombre1 + nombre2
            nom_operation = "Addition"
        elif operation == "-":
            resultat = nombre1 - nombre2
            nom_operation = "Soustraction"
        elif operation == "*":
            resultat = nombre1 * nombre2
            nom_operation = "Multiplication"
        elif operation == "/":
            resultat = nombre1 / nombre2
            nom_operation = "Division"

        # Afficher le résultat
        print("\n" + "="*40)
        print(f"✓ {nom_operation} réussie !")
        print(f" {nombre1} {operation} {nombre2} = {resultat}")
        print("="*40)

        # Incrémenter le compteur
        nombre_calculs += 1

else:
    # Opération invalide
    print("\n✗ ERREUR : Opération non reconnue !")
    print(" Conseil : Utilise uniquement +, -, * ou /")

except ValueError:
    # Erreur si l'utilisateur ne tape pas un nombre
    print("\n✗ ERREUR : Entrée invalide !")
    print(" Conseil : Tu dois entrer un nombre (ex: 5 ou 3.14)")

```

```

except Exception as e:
    # Attraper toute autre erreur imprévue
    print(f"\n❌ ERREUR inattendue : {e}")

# Demander si on continue
print("\n" + "-"*40)
continuer = input("Veux-tu faire un autre calcul ? (oui/non) : ").strip()

# Message de fin
print("\n" + "=="*40)
print(f" Total de calculs effectués : {nombre_calculs}")
print(" Merci d'avoir utilisé la calculatrice !")
print("💖 À bientôt sur C Premiers Pas !")
print("=="*40 + "\n")

# Point d'entrée du programme
if __name__ == "__main__":
    calculatrice()

```

Améliorations de cette version finale

Nouvelles fonctionnalités :

1. **Compteur de calculs** : Affiche combien de calculs tu as faits
2. **Gestion d'erreurs avancée** (try/except) : Attrape les erreurs quand tu tapes du texte au lieu d'un nombre
3. **Interface améliorée** : Plus de séparateurs visuels (=, -)
4. **Messages d'aide** : Conseils quand il y a une erreur
5. **Nom des opérations** : Affiche "Addition réussie !" au lieu de juste le résultat
6. **strip()** : Enlève les espaces avant/après ce que tu tapes

Aller plus loin

Tu maîtrises maintenant les bases ! Voici 5 idées pour améliorer ta calculatrice :

1. Calculatrice scientifique

Ajoute des opérations avancées :

```
import math

# Dans ta fonction calculatrice, ajoute :
elif operation == "**":
    resultat = nombre1 ** nombre2 # Puissance
    nom_operation = "Puissance"
elif operation == "sqrt":
    resultat = math.sqrt(nombre1) # Racine carrée
    nom_operation = "Racine carrée"
elif operation == "%":
    resultat = nombre1 % nombre2 # Modulo (reste de division)
    nom_operation = "Modulo"
```

Nouvelles opérations disponibles :

- `**` : Puissance ($5 ** 2 = 25$)
- `sqrt` : Racine carrée ($\sqrt{16} = 4$)
- `%` : Modulo ($10 \% 3 = 1$, le reste de $10 \div 3$)

2. Historique des calculs

Garde une trace de tous les calculs :

```
def calculatrice():
    historique = [] # Liste pour stocker les calculs

    # Dans ta boucle, après chaque calcul :
    historique.append(f"{nombre1} {operation} {nombre2} = {resultat}")

    # À la fin, affiche l'historique :
    print("\n HISTORIQUE DES CALCULS")
    for i, calcul in enumerate(historique, 1):
        print(f"{i}. {calcul}")
```

Résultat :

HISTORIQUE DES CALCULS

1. $10.0 + 5.0 = 15.0$
2. $20.0 * 3.0 = 60.0$
3. $100.0 / 4.0 = 25.0$

3. Sauvegarde dans un fichier

Enregistre les calculs dans un fichier texte :

```
def sauvegarder_calcul(calcul, resultat):
    """Sauvegarde un calcul dans un fichier"""
    with open("historique_calculatrice.txt", "a", encoding="utf-8") as fichier:
        from datetime import datetime
        date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        fichier.write(f"[{date}] {calcul} = {resultat}\n")
    print("💾 Calcul sauvegardé dans historique_calculatrice.txt")

# Dans ta boucle, après chaque calcul :
sauvegarder_calcul(f"{nombre1} {operation} {nombre2}", resultat)
```

4. Interface graphique (Tkinter)

Crée une vraie fenêtre avec boutons :

```
import tkinter as tk

def cliquer_bouton(chiffre):
    # Code pour gérer les clics sur les boutons
    pass

# Créer la fenêtre
fenetre = tk.Tk()
fenetre.title("Ma Calculatrice")

# Créer l'écran d'affichage
ecran = tk.Entry(fenetre, width=20, font=("Arial", 20))
ecran.grid(row=0, column=0, columnspan=4)

# Créer les boutons (0-9, +, -, *, /, =)
# ... code des boutons ...

fenetre.mainloop()
```

Résultat : Une vraie calculatrice avec interface comme sur ton ordinateur !

5. 🎤 Calculatrice vocale

Utilise la reconnaissance vocale :

```
import speech_recognition as sr

def ecouter_commande():
    """Écoute et reconnaît la voix de l'utilisateur"""
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("🎤 Parle maintenant...")
        audio = recognizer.listen(source)
        try:
            commande = recognizer.recognize_google(audio, language="fr-FR")
            print(f"Tu as dit : {commande}")
            return commande
        except:
            print("❌ Je n'ai pas compris")
            return None

# Dans ta calculatrice :
# nombre1 = ecouter_commande() # "cinq"
# operation = ecouter_commande() # "plus"
# nombre2 = ecouter_commande() # "trois"
```





Exemple d'utilisation :




```
🎤 Parle maintenant...
Tu as dit : dix plus cinq
✓ Résultat : 10 + 5 = 15
```

Récapitulatif : Ce que tu as appris



Toutes les compétences que tu maîtrises maintenant

Notion	Description	Utilité dans la vie réelle
 Conditions (if/elif/else)	Prendre des décisions dans ton code	Site web : "Si mot de passe correct → connecter, sinon → erreur"
 Boucle while	Répéter des actions jusqu'à une condition	Jeu vidéo : "Tant que le joueur a des vies, continuer le jeu"
 Gestion d'erreurs	Anticiper et gérer les problèmes	Application : "Si pas de connexion internet → afficher message clair"
 Opérateurs logiques	Combiner plusieurs conditions	E-commerce : "Si panier > 50€ ET code promo valide → réduction"

Notion	Description	Utilité dans la vie réelle
 Fonctions	Organiser et réutiliser du code	Application mobile : Une fonction "envoyer_notification" utilisée partout
 Conversion de types	Transformer les données (string → float)	Formulaire web : Convertir "25" (texte) en 25 (nombre) pour calculer l'âge
 Opérations mathématiques	Faire des calculs	Application de finance : Calculer les intérêts, les totaux, les moyennes

Ce que tu peux DÉJÀ créer avec ces compétences

Un jeu de devinettes : "Devine le nombre entre 1 et 100" (boucle while + conditions)

Un système de connexion : "3 essais pour le mot de passe" (boucle + conditions + compteur)

Un quiz : "Réponds aux questions et calcule ton score" (conditions + boucle + calculs)

Un convertisseur de devises : "Convertis euros en dollars" (calculs + fonctions)

Un jeu de dés : "Lance les dés jusqu'à obtenir un double 6" (boucle while + random)

****Tu n'es plus un débutant. Tu es un créateur ! ****



Conclusion

Le chemin parcouru

Il y a quelques heures, tu ne savais peut-être pas ce qu'était une condition. Maintenant, tu as créé une **calculatrice intelligente complète** qui :

- ✓ Gère 4 opérations mathématiques
- ✓ Détecte et gère les erreurs
- ✓ Peut faire autant de calculs que tu veux
- ✓ Est organisée proprement avec une fonction
- ✓ Affiche des messages clairs et professionnels

Ce que tu as vraiment appris

Au-delà du code, tu as compris des concepts **fondamentaux** de la programmation :

Penser logiquement : "Si ceci, alors cela"

Automatiser : "Répète cette action jusqu'à..."

Anticiper les problèmes : "Que se passe-t-il si l'utilisateur se trompe ?"

Structurer : "Comment organiser mon code proprement ?"

Ces compétences sont la BASE de TOUS les programmes que tu créeras !

Prochaine étape : vCODE #4

La semaine prochaine, on va utiliser ce que tu as appris aujourd'hui pour créer quelque chose d'encore plus impressionnant. Prépare-toi !

Citation motivante

"Le code que tu as écrit aujourd'hui n'est pas parfait. Mais c'est le tien. Tu l'as compris. Tu peux l'expliquer. C'est ça, être développeur."

— C Premiers Pas

Continue à pratiquer !

Ne t'arrête pas là. Prends le code de la calculatrice et :

1. **Modifie-le** : Change les messages, ajoute des émojis
2. **Améliore-le** : Intègre une des 5 idées du chapitre "Aller plus loin"
3. **Crée ta version** : Fais une calculatrice pour un usage spécifique (pourboires, notes d'examen, etc.)
4. **Partage-le** : Montre ton code à la communauté, demande des retours

L'apprentissage se fait en pratiquant, pas en regardant !

Challenge bonus

Essaie de créer une calculatrice de **pourboires** :

- Demander le montant de l'addition
- Demander le % de pourboire (10%, 15%, 20%)
- Calculer le pourboire
- Afficher le total à payer

Indice : Tu as TOUTES les compétences nécessaires maintenant !

Document réalisé par C Premiers Pas

 **Contact** : [NOUS CONTACTER](#)

 **Rejoignez notre communauté**

 LINKEDIN

 WHATSAPP

 DISCORD

© 2026 C Premiers Pas - Tous droits réservés