Exercice - Expressions

2 / 8 - 1 / 8 → 0.125

Les divisions passent avant la soustraction : 2/8 vaut 0.25 et 1/8 vaut 0.125.

Le calcul donne donc 0.25 − 0.125 = 0.125

2 ** 3 → 8

** est l'élévation à la puissance : $2^3 = 8$.

Le résultat reste un *int*, car la base et l'exposant sont des entiers (avec un exposant positif).

"K" + "L" → 'KL'

Entre deux chaînes, le + n'additionne rien : il les **colle bout à bout** (concaténation).

Le résultat est une chaîne (*str*).

1 / 3 → 0.3333333333333333

Division float classique. Python affiche autant de décimales que sa précision le permet (environ 17 chiffres). Le résultat est un *float*.

(1 + 3) / 4 → 1.0

Voici le cas le plus surprenant. (1 + 3) vaut 4, puis 4 / 4 vaut 1… mais l'affichage donne **1.0** et non 1. La raison : / produit toujours un float, **même quand la division tombe juste**. Ce .0 n'est pas une vraie décimale, c'est seulement la marque du type float.

350 // 100 → 3

// est la division **entière** : seul le quotient est conservé, le reste est ignoré. 350 ÷ 100 = 3 reste 50, donc le résultat est 3. Comme les deux nombres sont des entiers, le résultat est un *int*. À comparer avec /, qui aurait donné 3.5.

"K" + str(45) → 'K45'

str(45) transforme le nombre 45 en chaîne "45", ce qui permet de le coller à "K". Sans le str(), l'expression "K" + 45 provoquerait une erreur, car une chaîne et un entier ne peuvent pas s'additionner.

float(3) → 3.0

Conversion volontaire de l'entier 3 en *float*. La valeur reste identique, mais le type change, d'où l'apparition du .0.

print(round(1 / 3, 3)) → 0.333

1 / 3 vaut 0.3333..., puis round(..., 3) arrondit à **3 décimales**, ce qui donne 0.333. Détail à connaître : round ne conserve pas les zéros inutiles. Par exemple, round(0.5, 3) afficherait 0.5, et non 0.500.

print(round(2 / 3 + 1, 2)) → 1.67

En respectant l'ordre des opérations : 2 / 3 vaut 0.6666..., l'ajout de 1 donne 1.6666..., et round(..., 2) arrondit à 2 décimales → 1.67.