Les types de base en Python

Résumé

En Python, chaque donnée a un **type** qui détermine ce qu'on peut en faire : on additionne des nombres, on concatène des chaînes de caractères, on teste des conditions avec des booléens. Confondre les types est l'une des sources d'erreur les plus fréquentes chez les débutants (par exemple, écrire `"3" + 4` au lieu de `int("3") + 4`). Comprendre les types, c'est comprendre le langage que parle Python pour représenter les informations. Cette fiche présente les quatre types fondamentaux et la valeur spéciale `None`, puis montre comment passer de l'un à l'autre.

Prérequis : Variables et affectation.

① Mémo

Définition 1 (Les quatre types fondamentaux)

`int` Entier relatif, précision illimitée en Python : `42`, `-7`, `0`.
`float` Nombre à virgule flottante (norme IEEE 754, précision finie) : `3.14`, `-0.5`, `1e-3`.
`str` Chaîne de caractères, entre guillemets simples ou doubles : `'hello'`, `"bonjour"`.
`bool` Booléen, deux valeurs : `True` et `False`.

Définition 2 (La valeur spéciale `None`)

`None` représente l'**absence de valeur**. Ce n'est ni un nombre, ni une chaîne, ni un booléen : c'est un type à part (`NoneType`).

- Une fonction sans `return` renvoie `None`.
- `print()` affiche à l'écran et renvoie `None`.
- On teste avec `is` : `x is None` (et non `x == None`).

Conversions de type (transtypage)

On convertit un type en un autre avec les fonctions éponymes :

- `int("42")` renvoie l'entier `42`;
- `float(3)` renvoie `3.0`;
- `str(42)` renvoie la chaîne `"42"`;
- `bool(0)` renvoie `False` (tout nombre non nul donne `True`).

Opérateurs arithmétiques

`+`, `-`, `*` Addition, soustraction, multiplication.
`/` Division réelle : `7 / 2` donne `3.5`.
`//` Division entière (quotient) : `7 // 2` donne `3`.
`%` Modulo (reste) : `7 % 2` donne `1`.
`**` Puissance : `2 ** 10` donne `1024`.

Pièges fréquents

- `0.1 + 0.2` ne vaut pas exactement `0.3` (erreur d'arrondi des flottants).
- `"3" + "4"` donne `"34"` (concaténation, pas addition).
- `type(7 / 1)` renvoie `float` : la division `/` renvoie toujours un flottant.
- `int("3.5")` provoque une erreur : il faut d'abord passer par `float`.

Erreurs classiques

Code erroné	Code correct	Explication
`"3" + 4` ⟶ `TypeError`	`int("3") + 4` ⟶ `7`	On ne peut pas additionner une chaîne et un entier : il faut convertir explicitement.
`7 - 2 // 2` ⟶ `6`	`(7 - 2) // 2` ⟶ `2`	La division entière `//` est prioritaire sur la soustraction : sans parenthèses, seul `2 // 2` est calculé d'abord.
`int("3.5")` ⟶ `ValueError`	`int(float("3.5"))` ⟶ `3`	`int()` ne convertit pas une chaîne contenant un point : il faut d'abord passer par `float()`.
`0.1 + 0.2 == 0.3` ⟶ `False`	`abs(0.1 + 0.2 - 0.3) < 1e-9`	Les flottants ont des erreurs d'arrondi : on compare avec une tolérance, jamais avec `==`.

② Exemples

Programme 1 · Vérifier le type d'une variable

```
x = 42
print(type(x))        # <class 'int'>

y = 42.0
print(type(y))        # <class 'float'>
```

Programme 2 · Division entière et modulo

```
heures = 137 // 60    # 2 heures
minutes = 137 % 60    # 17 minutes
print(f"{heures}h{minutes}min")  # 2h17min
```

Programme 3 · Piège des flottants

```
print(0.1 + 0.2 == 0.3)          # False !
print(abs(0.1 + 0.2 - 0.3) < 1e-9)  # True (comparaison correcte)
```

Programme 4 · Transtypage

```
age = input("Âge : ")    # input renvoie toujours un str
age = int(age)           # conversion nécessaire pour calculer
print(f"Dans 10 ans : {age + 10}")
```

Programme 5 · Booléens et valeurs "falsy"

```
print(bool(0))       # False
print(bool(""))      # False  (chaîne vide)
print(bool([]))      # False  (liste vide)
print(bool("0"))     # True   (chaîne non vide !)
```

③ Exercices

Exercice 1 *Types et transtypage* Donner le type et la valeur de chaque expression :

1. `17 // 5`
2. `17 / 5`
3. `"17" + "5"`
4. `int("17") + int("5")`
5. `bool("")`
6. `float(True)`

Exercice 2 *Extraction de chiffres* Écrire un programme qui demande un entier positif à trois chiffres et affiche séparément le chiffre des centaines, des dizaines et des unités.

Exemple. Pour `427`, afficher `4`, `2`, `7`.

Exercice 3 *Conversion de durée* On dispose d'une durée exprimée en secondes. Écrire un programme qui affiche cette durée en heures, minutes et secondes.

Exemple. Pour `3725` secondes, afficher `1h 2min 5s`.

Exercice 4 *Erreurs d'arrondi*

1. Prédire le résultat de chaque expression puis vérifier en Python :

```
print(0.1 + 0.2 == 0.3)
print(0.1 + 0.2)
print(type(1 + 2.0))
```

2. Expliquer pourquoi la première ligne affiche un résultat surprenant.
3. Proposer une façon de comparer correctement deux flottants (sans écrire de fonction).

Solutions des exercices

Corrigé de l'exercice 1.

1. `3` de type `int` (quotient de la division entière).
2. `3.4` de type `float` (la division `/` renvoie toujours un flottant).
3. `"175"` de type `str` (concaténation de deux chaînes).
4. `22` de type `int` (on additionne deux entiers après conversion).
5. `False` de type `bool` (une chaîne vide est considérée comme fausse).
6. `1.0` de type `float` (`True` est converti en `1` puis en flottant).

Corrigé de l'exercice 2.

```
n = int(input("Entier à 3 chiffres : "))
centaines = n // 100
dizaines = (n % 100) // 10
unites = n % 10
print(f"Centaines : {centaines}")
print(f"Dizaines : {dizaines}")
print(f"Unités : {unites}")
```

Principe : la division entière par 100 isole les centaines. Le modulo 100 donne les deux derniers chiffres, puis la division entière par 10 isole les dizaines. Le modulo 10 donne les unités.

Corrigé de l'exercice 3.

```
total = 3725
h = total // 3600
m = (total % 3600) // 60
s = total % 60
print(f"{h}h {m}min {s}s")
```

Principe : la division entière par 3600 donne les heures. Le reste (modulo 3600) donne les secondes restantes, que l'on divise par 60 pour obtenir les minutes.

Vérification : $1 \times 3600 + 2 \times 60 + 5 = 3725$. ✓

Corrigé de l'exercice 4.

1. Résultats : `False`, `0.30000000000000004`, `<class 'float'>`.
2. Les nombres `0.1` et `0.2` ne sont pas représentables exactement en binaire (norme IEEE 754). Leur somme vaut `0.30000000000000004`, qui diffère de la représentation interne de `0.3`.
3. On compare la valeur absolue de la différence à un seuil très petit :

```
a = 0.1 + 0.2
b = 0.3
print(abs(a - b) < 1e-9)   # True
```