Nom :	Prénom :	Classe :

NSI 1re — Type de donnée - entier, flottant, booléen, chaîne de caractères

Élément du programme : Langages et programmation → Constructions élémentaires.

Pourquoi avoir plusieurs types de données ?

Les **types de données** viennent de la nécessité, en informatique, de **différencier** les différentes formes d'information à traiter : un nombre, du texte, une valeur vraie ou fausse. Ces catégories existent dans **tous les langages de programmation** et déterminent comment la machine représente et manipule l'information.

Qu'est-ce que ça change ?

Le type de donnée influence :

- Les opérations possibles (par exemple, on *additionne* des entiers, tandis que l'on *concatène* des chaînes de caractères).
- L'espace mémoire utilisé (un *flottant* consomme en général plus de mémoire qu'un *entier*, et l'impact d'une *chaîne de caractères* sur la mémoire dépend de sa longueur).
- La rapidité d'exécution (certains calculs sont plus rapides avec certains types).
- Le résultat des calculs ou manipulations (par exemple, une addition donne un nombre, une concaténation donne une chaîne).

Chaque type a donc une origine, une utilité et des restrictions spécifiques qui impactent la manière dont le programmeur et la machine travaillent ensemble. Dans ce cours, nous allons nous pencher sur les types suivants :

- Les entiers, *integer* en anglais, `int` en Python.
- Les flottants, *float* en anglais, et `float` également en Python.
- Les booléens, *boolean* en anglais, `bool` en Python.
- Les chaînes de caractères, *string* en anglais, `str` en Python.

Les entiers

En programmation, les **entiers** (int) sont utilisés lorsqu'il s'agit de manipuler des **nombres sans partie décimale**, et qui peuvent être positifs (ex : 12) ou négatifs (ex: -59).

Prenons un exemple, en Python :

```
1  >>> a = 5
2  >>> print(a)
3  5
```

Dans le code ci-dessus, nous assignons l'entier `5` à la variable `a` (ligne 1),
puis nous affichons ce que la variable `a` contient (ligne 2).

Python nous affiche alors `5` (ligne 3).

Affichage du type d'une variable

Il est également possible de demander à Python **d'afficher** *non pas le contenu* mais le **type** d'une variable.
Cette opération passe par la commande `type()`.

Pour comprendre comment cela fonctionne, reprenons notre exemple précédent. Mais au lieu d'afficher le *contenu* de `a`, nous allons en afficher le *type* :

```
1  >>> a = 5
2  >>> print(type(a))
3  <class 'int'>
```

Dans le code ci-dessus, nous assignons l'entier `5` à la variable `a` (ligne 1),
puis nous affichons le type de la variable `a` (ligne 2).

Python nous affiche alors `<class 'int'>` (ligne 3), ce qui signifie que cette variable est de type `int` (entier).

Savoir afficher le type de *telle* ou *telle* variable est très important en programmation.

Et oui, vous le verrez bientôt : *nombreuses* sont les situations où l'on ne comprend pas *ce qui ne fonctionne pas* dans son code 😡 avant de voir, grâce à un `print( type( nom_de_la_variable ) )` que la variable n'est pas du type escompté !

Et comme chaque type de données a ses subtilités, et bien ça change tout …
Alors n'oubliez pas : en cas de bug, pensez à afficher le type d'une variable, ça peut aider à y voir plus clair !

➤ Exercice 1 (à vous de jouer)

Lisez l'explication de code (colonne de gauche), puis écrivez le code Python correspondant (colonne de droite) :

Explication de code	Code Python (à compléter)
Nous assignons l'entier 23 à une variable nommée d. Puis nous affichons le contenu de cette variable. Puis nous affichons le type de cette variable.	

Explication de code	Code Python (à compléter)
Nous assignons l'entier 10 à une variable a. Nous assignons l'entier 3 à une variable b. Nous assignons à c le résultat de la division entière de a par b. Nous affichons le type de c.	

Les flottants

Les flottants sont faits pour celles et ceux qui manipulent des nombres non entiers, c'est-à-dire des nombres à virgules (en mathématiques, on parle de « nombres réels »). En Python, un flottant est nommé `float`.

Prenons un exemple :

```
1  >>> a = 3.14
2  >>> print(a)
3  3.14
4  >>> print(type(a))
5  <class 'float'>
```

Dans le code ci-dessus, nous assignons le flottant `3.14` à la variable `a` (ligne 1),
puis nous affichons la valeur stockée dans `a` (ligne 2), Python affiche `3.14` (ligne 3).

Et nous affichons le type de `a` (ligne 4), Python affiche `<class 'float'>` (ligne 5).
Donc `a` est bien un flottant.

Notons également qu'il est possible de déclarer en Python un flottant en notation scientifique.
Par exemple 2.5×10^{-3} s'écrirait `2.5e-3`.

Changement de type

En Python, le typage des variables est **dynamique**, c'est à dire que le **type** d'une variable est déterminé **au moment de son affectation**, mais **qu'il peut changer pendant l'exécution du programme**.

Premier exemple :

```
1  >>> a = 4
2  >>> print(type(a))
3  <class 'int'>
4  >>> a = a / 2
5  >>> print(type(a))
6  <class 'float'>
```

← Dans le code suivant, on stocke dans la variable `a` un entier. La variable `a` est donc de type entier.

Puis on stocke dans cette même variable le résultat d'une division « simple ». Elle devient donc de type flottant.[1]

Deuxième exemple :

```
1  >>> a = 3.14
2  >>> print(type(a))
3  <class 'float'>
4  >>> a = 10
5  >>> print(type(a))
6  <class 'int'>
7  >>> a = 10 / 5
8  >>> print(type(a))
9  <class 'float'>
```

← Dans ce nouveau code, on stocke dans la variable `a` un flottant. La variable `a` est donc de type flottant.

Puis on stocke dans cette même variable un entier. Elle devient donc de type entier.

Enfin, on stocke dans cette variable le résultat d'une division « simple ». Elle devient alors de type flottant.

1 Relire à ce propos le cours sur les calculs arithmétiques en Python.

➤ Exercice 2

Déterminez le type de la variable aux lignes indiquées :

```
1  a = 18.0
2  a = 10
3  a = a // 2
4  a = 10 / 2
5  a = a ** 2
6  a = 10 % 3
```

Ligne 1, `a` est de type : .....

Ligne 2, `a` est de type : .....

Ligne 3, `a` est de type : .....

Ligne 4, `a` est de type : .....

Ligne 5, `a` est de type : .....

Ligne 6, `a` est de type : .....

Les booléens

Les booléens sont comme de petits interrupteurs qui peuvent être allumés ou éteints.
En d'autres termes, un booléen ne peut avoir que deux états : vrai (True) ou faux (False).

En prenant un exemple de la vie courante, cela donnerait :

- L'interrupteur de la console de jeu est sur ON.
  Est-elle allumée ? Oui, c'est vrai (True).
- L'interrupteur de la console de jeu est sur OFF.
  Est-elle allumée ? Non, c'est faux (False).

En Python, un booléen est nommé `bool` (de l'anglais « boolean »),
et il ne peut avoir que deux valeurs : `True` ou `False`.

En Python, les valeurs `True` et `False` des booléens possèdent une **première lettre en majuscule**.

Ce qui n'est pas le cas dans d'autres langages de programmation...

Voyons comment assigner un booléen à une variable, en Python :

```
1  >>> a = True
2  >>> print(a)
3  True
4  >>> print(type(a))
5  <class 'bool'>
6  >>> b = False
7  >>> print(b)
8  False
9  >>> print(type(b))
10 <class 'bool'>
```

← Dans le code suivant, on stocke dans la variable `a` un booléen. La variable `a` est donc de type booléen.

Puis on change la valeur contenue dans `a` (ligne 6), mais cette valeur est encore un booléen.

Donc la variable `a` reste de type booléen.

Les chaînes de caractères

En programmation, les **chaînes de caractères** (« strings », en anglais) sont des séquences (suites) de caractères utilisables dans tous les domaines où le texte, les mots ou les symboles doivent être manipulés.

Les développeuses et développeurs travaillent avec des chaînes de caractères pour gérer du texte, des identifiants, des messages ou des informations tapées par l'utilisateur.

> On parle de façon équivalente de « chaînes de caractères » ou simplement de « chaînes ».
>
> Cela désigne la même chose !

Une chaîne sert à **stocker** et **manipuler** des **textes** (ex : nom, prénom, phrase, mot de passe, etc).

Exemple de manipulations de chaîne :

- Effectuer une recherche, par exemple rechercher un mot dans un texte complet.
- Comparer deux chaînes différentes.
- Fusionner deux chaînes différentes, c'est à dire assembler (concaténer) deux chaînes ensemble.

Quelques règles concernant les chaînes en Python

En Python, une chaîne peut être entourée ...

... de guillemets simples	... ou de guillemets doubles (ça revient au même)
`a = 'salut je suis une chaîne !'`	`a = "salut je suis une chaîne !"`

Il faut choisir soit l'un (guillemets simples), soit l'autre (guillemets doubles).
Mais pas les deux en même temps ! Sinon on aurait une erreur :

```
"Hésitation totale'
^
SyntaxError: unterminated string literal
```

L'exemple ci-dessus provoque une erreur de syntaxe.

En Python le type « chaîne de caractères » se nomme `str`.

```
1  >>> a = "I love Python"
2  >>> print(type(a))
3  <class 'str'>
```

← Dans le code suivant, on stocke dans la variable `a` une chaîne. La variable `a` est donc de type chaîne.

Que pouvons-nous mettre dans une chaîne ?

Les chaînes de caractères peuvent contenir **tout type de caractère** : lettres, majuscules, chiffres, symboles, lettres issues de différents alphabets (arabe, coréen, cyrillique, devanagari, grec, etc).

Donc rien ne nous empêche de créer par exemple les chaînes suivantes :

```
a = "الخوارزمي"
a = "नमस्ते"
a = "(✿◠‿◠)"
a = "Mille millions de mille sabords ! ... Ectoplasme !"
a = "https://www.youtube.com/watch?v=wf9Hh6pu78I"
a = "C'est la saison des 🍄"
a = "✅ Commande expédiée"
```

Les émojis sont des symboles comme les autres ! Un nouvel espace de créativité s'offre à nous !

Et si une chaîne contient des guillemets ?

Parfois nous devons créer des chaînes de caractères comprenant des guillemets.

Par exemple, nous souhaitons stocker la phrase `J'ai un petit "faible" pour Python` dans une variable.

- Cette phrase contient un guillemet **simple** : `J'ai`
- Et deux guillemets **doubles** : `"faible"`

Si nous entourons cette phrase de guillemets simples, nous avons une erreur Python :

```
phrase = 'J'ai un petit "faible" pour Python'
                                           ^
SyntaxError: unterminated string literal
```

Dans ce code, Python a cru que la chaîne s'arrêtait à la lettre J.

Et si nous entourons cette phrase de guillemets doubles, nous avons toujours une erreur Python :

```
phrase = "J'ai un petit "faible" pour Python"
                         ^
SyntaxError: invalid syntax
```

Dans ce code, Python a cru que la chaîne s'arrêtait après l'espace suivant le mot « petit ».

Heureusement, les langages de programmation comme Python permettent d'utiliser un **caractère d'échappement**. Lorsqu'il est utilisé, le caractère d'échappement permet, comme son nom l'indique, d'échapper, (ignorer) le caractère qui suit. En Python, le caractère d'échappement est `\` (antislash).

Ce qui nous permet de stocker la phrase `J'ai un petit "faible" pour Python` dans une variable, sans soucis, regardez :

```
phrase = "J'ai un petit \"faible\" pour Python"     # super, ça fonctionne !

phrase = 'J\'ai un petit "faible" pour Python'      # super, ça fonctionne aussi !
```

➤ Exercice 3

Déterminez le type des variables suivantes :

Code	Type (à compléter)
`a = 14`	
`a = 3.15`	
`a = "14"`	
`a = 10 / 2`	
`a = False`	
`a = -1298377291243`	
`a = 10 + 94`	
`a = 9e-4`	
`a = 234981 // 12382`	
`a = 234981 % 12382`	
`a = 0`	
`a = 123 - 3`	
`a = 10 ** 3`	
`a = 'J\'aime grave'`	
`a = "\"Hello\""`	
`a = 3.1415926535897`	
`a = '10400123 // 3'`	
`a = 1239912385823`	
`a = 62e-20`	