Les types des variables

Durant l'affectation, le type des variables est choisi automatiquement en fonction de ce qu'on trouve à la droite du symbole =. Nous utilisons la fonction Python `type()` afin de savoir ce qui a été décidé. La fonction `print()` permet d'afficher le résultat à l'écran :

```
a = 1
print(type(a))
a = 1.0
print(type(a))
a = 'Bonjour'
print(type(a))

>> <class 'int'>
>> <class 'float'>
>> <class 'str'>
```

À la première ligne, nous associons le nombre 1 à la variable `a`. Comme ce nombre est sans virgule, la convention veut qu'il soit stocké comme un nombre entier : *integer* en anglais ou en abrégé *class int*. À la troisième ligne, la valeur 1.0 désigne cette fois un nombre à virgule. Automatiquement, l'interpréteur bascule vers ce type dénommé **class float**. Dans la dernière affectation, on trouve du texte entouré par des guillemets, c'est le code pour désigner une chaîne de caractères, **string** en anglais ou encore **class str**.

Type ou classe ? Vous entendrez l'une ou l'autre désignation. Ne vous inquiétez pas, ces deux termes désignent exactement le même concept. Le terme « type » est apparu en premier avec les variables. Le terme « classe » est plus récent car il est relié à la programmation objet. Dans d'autres langages, comme en Java/C++/C#, on distingue les types primitifs (les nombres essentiellement) des classes (tout ce qui est différent d'un nombre : une date, une couleur, la souris, le clavier…). Il s'agit ici d'une habitude. Python a regroupé ces deux tendances et ne fait pas de différence entre ces deux termes.

Dans la mémoire RAM de l'ordinateur, on stocke l'information dans des cases appelées octets. Chaque octet correspond à un paquet de 8 bits chacun pouvant valoir 0 ou 1. Ainsi, pour stocker chaque type d'information : du texte, des nombres entiers, des nombres à virgule, des dates… il va falloir encoder chaque donnée sous forme de 0 et de 1 suivant une norme donnée. Pour les nombres à virgule, on utilise la norme IEEE 754 par exemple. Pour le texte, on utilise le standard Unicode lié à la norme ISO/CEI 10646 afin de permettre des échanges de texte dans différentes langues au niveau mondial. Pour la représentation des couleurs à l'écran, on utilise le standard sRGB (*Red Green Blue*) défini par la norme CIE 61966. Vous l'aurez compris, il y a une norme pour tout !

1. Quand utiliser un type entier ?

C'est une très bonne question ! En effet, si l'on peut utiliser des nombres à virgule, on peut se passer des nombres entiers ! En fait, tout dépend...

- Si vous traitez des prix par exemple, vous pouvez stocker toutes vos données dans des nombres à virgule, même si un des prix est un chiffre rond. En effet, si vous avez les valeurs 12 € 99, 10 € 30 et 12 €, vous n'avez aucun intérêt à choisir des types de stockage différents en fonction de la valeur. Vous pouvez choisir le type nombre à virgule pour représenter l'ensemble des prix et c'est une sage décision.
- Lorsque vous voulez désigner un élément dans une liste, prendre un nombre à virgule est maladroit. Si vous voulez désigner le deuxième prénom dans la liste ['Bob', 'Alice', 'Edouard'], cette position doit être décrite par un nombre entier. Vous ne demanderiez jamais l'élément se trouvant à la position 2.34 !
- Pour les variables qui comptent des choses, cela peut poser problème de choisir un nombre à virgule. Si vous comptez

le nombre de filles dans une classe et que vous vouliez savoir s'il y en a autant que de garçons, vous allez comparer deux nombres à virgule en faisant par exemple : 18.00000 est-il égal à 18.00000 ? Or, un nombre à virgule en informatique a une précision limitée, et avec 1/1 000 000 de précision, la comparaison précédente peut être vraie ou fausse suivant les erreurs de précision accumulées durant l'exécution. Les tests d'égalité sur les nombres à virgule sont une source d'erreurs aléatoires fréquentes !

Préférez un nombre entier pour comparer des valeurs avec le test d'égalité.

2. Les opérations entre types différents

Un jour ou l'autre, vous allez additionner un nombre à virgule et un nombre entier ! De quel type sera le résultat ? Le principe suivant s'applique : le résultat correspondra au type float, c'est le nombre à virgule qui l'emporte sur le nombre entier. Vous n'avez rien à faire pour cela. Le mécanisme est automatique. Il s'agit du choix fait par défaut et il se révélera dans la pratique le plus adéquat.

```
a = 1
b = 3.14
c = a+b
print(type(c))
print(c)

>> <class 'float'>
>> 4.140000000000001
```

Si vous effectuez une opération +, - ou * entre un type entier et un type float, le résultat final sera de type float.

Le résultat d'une opération entre deux nombres entiers reste entier. Cependant, il existe un cas en Python où l'opération entre deux nombres entiers est un nombre à virgule ! Il s'agit de la division. Vous tomberez sûrement sur ce piège un jour ou l'autre. Si vous voulez éviter toute ambiguïté, vous pouvez utiliser le symbole double de la division // qui force le résultat obtenu à être un nombre entier :

```
a = 9
b = a / 2
print(b)
print(type(b))

b = a // 2
print(b)
print(type(b))

>> 4.5
>> <class 'float'>
>> 4
>> <class 'int'>
```

3. Convertir un type en un autre

Il est possible de transformer une chaîne de caractères en nombre. Cela arrive, par exemple, lorsque l'on demande à l'utilisateur d'entrer une valeur au clavier. L'information récupérée est une chaîne et il faut alors la convertir. La

fonction pour effectuer ce travail est **int()** :

```
a = ’123 ’
b = int(a)
print(b+1)
print(type(b))

>> 124
>> <class ’int’>
```

Si vous traitez une chaîne de caractères correspondant à un nombre à virgule, il faut utiliser la fonction équivalente **float()** :

```
a = ’123.5’
b = float(a)
print(b+1)
print(type(b))

>> 124.5
>> <class ’float’>
```

À noter qu’une fois converties en valeurs numériques, vous pouvez manipuler les quantités comme bon vous semble avec les opérations arithmétiques habituelles.

Vous pouvez avoir besoin de convertir un nombre à virgule en valeur entière. Pour cela, la fonction **int()** est utile :

```
a = 123.5
b = int(a)
print(b)
print(type(b))

>> 123
>> <class ’int’>
```

Pour convertir un nombre entier ou un nombre à virgule en chaîne de caractères, la fonction **str()** est disponible :

```
a = 7
b = 3.14

c = str(a)
d = str(b)

print(type(c))
print(type(d))
print(c,d)

>> <class ’str’>
>> <class ’str’>
```

```
>> 7 3.14
```

En résumé, les fonctions de conversion depuis un type vers un autre type sont présentées dans le tableau suivant :

De/Vers	Nombre entier	Nombre flottant	Chaîne de caractères
Nombre entier		`float()`	`str()`
Nombre flottant	`int()`		`str()`
Chaîne de caractères	`int()`	`float()`	