Les flottants

En Python, les nombres réels sont représentés de façon approchée au format double précision 64 bits de la norme IEEE 754 dans le type `float`. On désigne ces représentations approchées sous le terme de **flottants**.

Le type `float` de Python reprend l'idée d'une représentation en virgule flottante de la notation scientifique. Un réel x est représenté en base deux par un flottant :

$$f(x) = \text{signe} \times 2^{\text{exposant}} \times mantisse$$

où la mantisse est un nombre de la forme $1, d_1 ... d_{52}$. L'exposant positif ou négatif s'écrit sur 12 bits, la mantisse (nombre de chiffres après la virgule) sur 52 bits et il y a un bit pour stocker le signe. Ce format permet de représenter des nombres très petits ou très grands (ce que ne permettrait pas un format à virgule fixe) mais avec certaines contraintes :

- Les nombres de bits pour stocker l'exposant et la mantisse étant limités, il existe donc un plus petit et un plus grand nombre flottants représentables, la possibilité d'un dépassement de capacité par `underflow` ou `overflow` existe donc alors que pour les entiers de type `#!python int` la seule limite est la taille de la mémoire :

```
# dépassement de la capacité lors de la conversion  d'un entier en flottant {#dépassemer
>>> float(10**311)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
OverflowError: int too large to convert to float
# l'opération entre deux flottants dépasse le flottant maximum, résultat infini {#lopéra
>>> 1e308 * 1000
inf
```

- Le nombre de chiffres est limité après la virgule à 52 bits donc la plupart des réels sont représentés de façon approchée. **Il ne faut pas attendre une réponse exacte d'un test d'égalité entre deux flottants**. Par exemple, des réels comme `0.3` ont un développement fini en base dix mais infini en base deux, et ne sont donc pas représentés de façon exacte.

```
(0.1 + 0.1 + 0.1) == 0.3   # False
(0.1 * 3) == 0.3           # False
(0.1 * 4) == 0.4           # True
```

- **Pour tester l'égalité de deux flottants, utiliser une inégalité avec un seuil d'erreur.** Par exemple, la fonction `isclose(a, b, rel_tol=1e-09)` du module `math` renvoie `True` si l'erreur relative `abs(a - b)/max(abs(a), abs(b))` est inférieure à `rel_tol` .

```
from math import sqrt, isclose

a, b, c = sqrt(2), sqrt(3), sqrt(5)
(a**2 + b**2) == c**2              # False
isclose(a**2 + b**2, c**2)         # True
```

- **Il ne faut jamais baser un test sur une comparaison de flottants.** Par exemple, la boucle ci-dessous ne se termine pas :

```
k = 0
while k != 1.0:
    k = k + 0.1
```

Il faudrait plutôt l'écrire ainsi :

```
k = 0
while k < 1.0:
    k = k + 0.1
```