Nombres et calcul

1) Opérations de base

Calcul à la main	Calcul en Python	Résultat affiché
2 + 3,5	2 + 3.5	5.5
4 x 3	4 * 3	12
2^3	2**3	8
$1{,}8 \times 10^{-2}$	1.8e-2	0.018
$\frac{3}{8}$	3/8	0.375

2) La division entière à reste et la division décimale :

Si on effectue la division euclidienne de 154 par 6 on obtient : 154 = 25 x 6 + 4.

25 est le quotient et correspond au nombre maximal de paquets de 6 unités pouvant être fait à partir de 154 unités. 4 représente le reste. Les commandes Pythons pour obtenir ces deux grandeurs sont :

- Pour le quotient : **154//6**
- Pour le reste : **154%6**

En revanche, si on tape la commande **print(154/6)** on voit s'afficher 25.666666666666668

Cela s'explique par le fait que Python ne peut représenter un nombre qu'avec un nombre fini de décimales, environ une quinzaine. D'ailleurs on peut facilement trouver la plus petite puissance de 10 pouvant être prise en compte en étant ajoutée à 1 en notant que :

- **print((1+1e-16)-1)** donne à l'exécution 0.0, la valeur exacte étant 10^{-16}
- **print((1+1e-15)-1)** donne à l'exécution 1,1102230246251565e-15, ce qui est très proche de la valeur exacte

on qualifie ce nombre 10^{-15} **d'epsilon machine**. Pour résumer ce qui est au-dessus, on peut considérer que pour l'ordinateur :

1 + 0,0000000000000001 = 1 (il y adonc perte d'information, la seizième décimale n'étant pas codable)

1 + 0,000000000000001 = 1,0000000000000011… (il n'y a donc pratiquement pas de perte d'information)

Cette perte d'information peut avoir de lourdes conséquences si on doit diviser. En effet, l'instruction

print(1/(1+1e-16)-1) renvoie un message d'erreur. Pour que ce ne soit pas le cas, il faudrait augmenter l'epsilon machine, ce qui peut se faire dans d'autres programmes que Python en déclarant des flottants en double précision (on peut même aller au-delà et cela se pratique couramment dans les gros calculs scientifiques)

Mais, quand on manipule des nombres dans le monde réel, il est rare qu'on ait besoin d'un résultat avec plus de trois chiffres, qu'on dit significatifs. Par exemple une distance de 1,24 km ou bien de 1,00 km, ce qui dans ce dernier cas signifie par l'ajout des deux 0 que la distance est donnée avec une marge d'incertitude de 0,01 donc comprise entre 0,99 et 1,01.

Python comporte la commande **round(nombre, precision)** afin d'arrondir un nombre à un nombre de décimales après la virgule défini par la variable entière precision. Ainsi **round(print(154/6),1)** affiche à l'exécution 25.7

3) Calcul avec des variables de type flottant et affichage du résultat

Nombreux calculs se font dans les applications les plus courantes avec des nombres décimaux donc de type float en Python. Cependant, la saisie génère toujours une chaine de caractère et pas un nombre. Pour pouvoir l'exploiter en tant que tel, il faut procéder à une conversion adaptée, dans notre exemple, en type flottant, avec la command **float()**, mais aussi déclarer une variable qui recevra le contenu de la saisie. Cela se fait ainsi, en donnant par exemple le nom distance à la variable :

```
distance= float(input("saisir une distance en km : "))
```

On peut alors adjoindre une autre saisie pour demander par exemple la durée pour effectuer la distance précédente.

```
duree= float(input("saisir une durée en h : "))
```

et ainsi, on pourra calculer la vitesse moyenne avec une précision d'un chiffre après la virgule, mettre le résultat dans une variable nommée vitesse puis l'afficher avec une phrase réponse :

```
vitesse = round(distance/duree,1)

print("la vitesse est de : "+str(vitesse)+" km")
```

A noter, la conversion nécessaire du contenu de la variable vitesse en string.

Il existe une variante pour afficher la vitesse avec un nombre de décimales fixé, c'est la commande **format**() qui s'utilise ainsi, sans avoir besoin d'utiliser la commande round() au préalable :

```
vitesse = distance/duree

print("la vitesse est de : {:.1f} km".format(vitesse))
```