Affectation des variables et impression des résultats

Les differents types de variables

Python est un langage objet, l'affectation du type de variable se fait par l'objet, il n'est donc pas nécessaire de déclarer le type d'une variable.

Il existe plusieurs types d'objets: - entier - flottant - chaine de caractère - booléen ...

voyons par l'exemple l'affectation des variables:

In [1]:
```
n = 1
type (n)
```

Out [1]:
```
int
```

La commande type renvoi le type de variable. Ici l'objet 1 étant un entier, la variable n prend le type de l'objet.

In [2]:
```
n = "1"
type(n)
```

Out [2]:
```
str
```

l'objet "1" étant une chaîne de caractère, la variable n prend le type chaine de caractère (str)

In [3]:
```
n = 1.13e-7
type (n)
```

Out [3]:
```
float
```

de même ici, 1.13e-7 est un flottant, la variable n prend le type flottant.

In [4]:
```
n = (1, 2, 3, 5, 7, 11, 13, 17, 19)
type (n)
```

Out [4]:
```
tuple
```

En conclusion, en python, c'est l'objet qui détermine le type de la variable.

La fonction input

Présentation

Attention, l'utilisation de `input` est fortement déconseillée, il vaut mieux utiliser des fonctions, voir un peu après.

Quand on demande à l'utilisateur d'entrer une grandeur, on utilise la commande input. Il faut cependant faire attention au type de grandeur :

In [5]:
```
n = input ("entrer un nombre entier : ")
type(n)
```

```
entrer un nombre entier : 42
```

Out [5]:
```
str
```

Par défaut, la fonction input entre une chaîne de caractère, avec laquelle nous ne pouvons pas faire de calculs. Il faut alors préciser que nous souhaitons une valeur entière :

In [6]:
```
n = int(input("entrer un nombre entier : "))
type(n)
```

```
entrer un nombre entier : 42
```

Out [6]:
```
int
```

In [7]:
```
n = float(input("entrer un nombre : "))
print(n)
```

```
entrer un nombre : 42.42
42.42
```

Ici n est un nombre réel.

Comment ne pas utiliser `input`

Imaginons un programme devant calculer le nombre de neutrons d'un atome; nous avons besoin de `A` et de `Z` pour faire le calcul.

Une (mauvaise) solution serait d'écrire le programme de la manière suivante :

```
A = int(input("A ?"))
Z = int(input("Z ?"))
print("Nombre de protons : ", A-Z)
```

Ce code nécessite une intéraction avec l'utilisateur pour être utilisé, on ne peut pas l'utiliser de manière automatique pour calculer le nombre de protons d'un ensemble d'atomes => pas bien :)

La "bonne" solution est d'écrire l'action que l'on souhaite faire sous la forme d'une fonction (voir un peu plus loin pour les détails sur les fonctions), en détaillant et documentant les arguments à utiliser, et de faire utiliser

cette fonction à l'utilisateur.

In [8]:
```
def calcul_nombre_protons(A, Z):
    """
    Calcul et affichage du nombre de neutrons d'un atome

    :param A: nombre de masse
    :param Z: numéro atomique
    """
    print("Nombre de neutrons : ", A-Z)


calcul_nombre_protons(A=12, Z=6)
```

```
Nombre de neutrons :  6
```

Print et l'écriture scientifique

Quand on souhaite imprimer un résultat à l'écran, on peut utiliser la fonction print.

Cependant, dans un notebook jupyter, il n'est pas toujours utile d'utiliser print : en effet, la valeur retournée par la dernière instruction est automatiquement affichée comme résultat de la cellule.

In [9]:
```
# Affichage sans "print" (notebook jupyter)
n = 10
m = n/7
m
```

Out [9]:
```
1.4285714285714286
```

In [10]:
```
# Affichage avec print
n = 10
m = n/7
print(m)
```

```
1.4285714285714286
```

L'instruction print a cependant quelques avantages : il est par exemple souvent utile en sciences de présenter les résultats avec un nombre correct de chiffres significatifs.

In [11]:
```
print("{0:.2e}".format(m))
```

```
1.43e+00
```

Ce formalisme dans le print permet de présenter les résultats avec le nombre de chiffres significatifs corrects. Nous avons ici l'écriture scientifique à trois chiffres significatifs.

La commande print permet d'afficher des caractères afin de présenter le résultat:

In [12]:

```
print("le résultats du calcul est: ", "{0:.2e}".format(m))
```

```
le résultats du calcul est:  1.43e+00
```

On remarque que les chaînes de caractères sont entre " ‘‘. Nous pouvons également utiliser ’ ‘. Il faut cependant faire attention, car si la chaîne de caractère contient un’, elle doit être entourée de” " :

In [13]:

```
print("ceci est une écriture correcte")
print('ceci est une écriture équivalente')
print("l'usage de l'apostrophe nécessite ce formalisme")
```

```
ceci est une écriture correcte
ceci est une écriture équivalente
l'usage de l'apostrophe nécessite ce formalisme
```

Alignement des résultats

il peut être utlise d’aligner les résultats pour une lecture facilitée.

L’instruction print(“{:60}’’.format()) calera ce qu’il y a dans la parenthèse du format sur 60 caractères, et permet donc cet alignement.

Exemple :

In [14]:

```
n = 10
m = n/7
print('{:60}'.format("le résultat avec un chiffre significatif est: "),
      "{0:.0e}".format(m))
print('{:60}'.format("le résultat avec deux chiffres significatif est: "),
      "{0:.1e}".format(m))
print('{:60}'.format("le résultat avec trois chiffres significatif est: "),
      "{0:.2e}".format(m))
print('{:60}'.format("le résultat avec quatre chiffres significatif est: "),
      "{0:.3e}".format(m))
```

```
le résultat avec un chiffre significatif est:                1e+00
le résultat avec deux chiffres significatif est:             1.4e+00
le résultat avec trois chiffres significatif est:            1.43e+00
le résultat avec quatre chiffres significatif est:           1.429e+00
```

Il est possible également d’utiliser la tabulation, grâce à la commande `\t`

In [1]:

```
n = 10
m = n/7
print("le résultat avec deux chiffres significatifs est: \t",
      "{0:.1e}".format(m))
print("le résultat avec trois chiffres significatifs est: \t",
      "{0:.2e}".format(m))
print("le résultat avec trois chiffres significatifs est: \t\t",
```