02-variables_input_print

April 1, 2019

1 Affectation des variables et impression des résultats

1.1 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.

1.2 la fonction input

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 entier12


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 entier44


Out[6]: int

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

entrer un nombre :12.5
12.5
```

Ici n est un nombre réel.

1.3 Print et l'écriture scientifique

Quand on souhaite imprimer un résultat à l'écran, on utilise la fonction print. Il est souvent utile en sciences de présenter les résultats avec un nombre correct de chiffre significatif.

```
In [8]: n = 10
        m = 10/7
        print(m)

1.4285714285714286


In [9]: 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 [10]: 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 [11]: 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
```

1.3.1 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 [20]: 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 [21]: n = 10
         m = n/7
```

```
        print("le résultat avec deux chiffre significatif est: \t",
              "{0:.1e}".format(m))
        print("le résultat avec trois chiffre significatif est: \t",
              "{0:.2e}".format(m))
        print("le résultat avec trois chiffre significatif est: \t\t",
              "{0:.2e}".format(m))
        print("le résultat avec trois chiffre significatif est: \t\t\t",
              "{0:.2e}".format(m))
```

```
le résultat avec deux chiffre significatif est:          1.4e+00
le résultat avec trois chiffre significatif est:          1.43e+00
le résultat avec trois chiffre significatif est:                  1.43e+00
le résultat avec trois chiffre significatif est:                          1.43e+00
```

1.3.2 le passage à la ligne

il peut être intéressant dans la présentation des résultats d'empécher le passage à la ligne. On utilise pour cela la commande end=''

```
In [14]: print("premier résultat", end =" ")
         print("second résultat")
```

```
premier résultat second résultat
```

La commande end = '' permet également de pouvoir positionner des séparateurs.

```
In [15]: for i in range (10):
             print(i, end=" - ")
```

```
0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 -
```

Il peut également être intéressant de passer volontairement à la ligne. On utilise pour cela la commande \n

```
In [22]: print("ceci est le début d'une phrase \nceci est la suite à la ligne")
```

```
ceci est le début d'une phrase
ceci est la suite à la ligne
```

1.3.3 affichage de caractères spéciaux

Il peut s'avérer nécessaire d'utiliser des caractères spéciaux. On utilise pour cela la commande

```
In [17]: print("le mot \"Pierre\" est entre guillements")
```

```
le mot "Pierre" est entre guillements
```

1.3.4 séparateurs

Enfin, il peut être utlise d'utiliser des séparateurs. On utilise pour cela la commande sep:

```
In [18]: print("premier résultat",
               "deuxième résultat",
               "troisième résultat",
               ".",
               sep = " -- ")
```

```
premier résultat -- deuxième résultat -- troisième résultat -- .
```

1.4 conclusion

Nous avons vu l'essentiel des fonctions imput et print. IL existe d'autres méthodes que l'on trouvera facilement quand l'usage s'en fera sentir.

```
In [ ]:
```