PRIVÉ : PYTHON AU LYCÉE (1) : LES VARIABLES ET OPÉRATEURS

1. TYPES DE VARIABLES

Dans le langage Python, chaque variable et chaque constante possède un **type**.

Les principaux types de base sont :

- **int** (integer) : nombre entier comme 5 ou -12345678901234567890
- **float** (floating-point number) : nombre décimal comme 0.0001 ou 3.141592654
- **str** (string) : chaîne de caractères comme « Bonjour toto123 ! » ou « A »

D'autres types (booléens, nombres complexes, listes, ...) seront vus dans des chapitres ultérieurs.

L'instruction « `type` » renvoie le type d'une expression. Par exemple :

```
>>> type(1) # affiche <class 'int'>
>>> type(1.5) # affiche <class 'float'>
>>> type('A') # affiche <class 'str'>
>>> type(True) # affiche <class 'bool'>
```

Remarque :
Dans l'exemple ci-dessus, les chevrons »> indiquent que les commandes ont été saisies en mode interactif - sous IDLE par exemple.
Les # indiquent le début d'un commentaire ; ici, les commentaires sont utilisés pour indiquer le résultat de l'instruction.

AFFECTATION D'UNE VALEUR À UNE VARIABLE

On affecte une valeur à une variable grâce au symbole « = ». Par exemple, l'instruction :

```
a=2
```

- crée la variable a (si elle n'existait pas déjà)
- affecte la valeur 2 à la variable a
- définit le type de a (ici : int)

En Python, il n'est pas nécessaire de déclarer préalablement les variables ou leur type.
Python détermine de façon dynamique le type d'une variable en fonction de sa valeur.
Il est ainsi possible de changer le type d'une variable à l'intérieur d'un programme.

Par exemple :

```
a=1 # a est de type 'int'
a='Bonjour' # ne provoque pas d'erreur. a est maintenant de type 'str'
```

Le nom d'une variable doit être composé uniquement de lettres, de chiffres et du symbole « _ » (underscore). Il ne doit pas commencer par un chiffre.

Exemple :

```
ma_variable=5 # correct
ma-variable=5 # incorrect
variable1=5 # correct
1variable=5 # incorrect
```

Enfin, signalons que la casse (distinction majuscule/minuscule) est également prise en compte : `MaVariable` et `mavariable` représentent deux variables différentes.

INSTRUCTIONS D'ENTRÉE/SORTIE

La fonction **print** affiche à l'écran la valeur d'une variable ou d'une constante. Il est possible d'afficher plusieurs valeurs en les séparant par une virgule.

Par exemple :

```
x=12
print('La valeur de x est',x) # affiche : La valeur de x est 12
print('Le type de x est',type(x)) # affiche : Le type de x est <class 'int'>
```

La fonction **input** est utilisée pour permettre à l'utilisateur d'entrer des données. Elle suspend le programme jusqu'à ce que l'utilisateur saisisse un texte. Lorsque l'utilisateur valide sa saisie à l'aide de la touche <ENTER> la fonction renvoie le texte saisi et le programme reprend son exécution.

Il est possible d'indiquer en paramètre de la fonction *input* une chaine de caractères qui sera affichée pour guider l'utilisateur (« prompt »).

Par exemple :

```
a=input('Saisir une valeur') # a recevra la valeur saisie
print('La valeur de a est',a) # affiche : La valeur de a est <valeur saisie par
    l'utilisateur>
```

La valeur saisie par l'utilisateur est toujours considérée par Python comme étant de type *str*. Pour obtenir un autre type de données, il faut *transtyper* la valeur saisie.

Exemple :

```
a=input('Saisir une valeur : ') # a sera du type string
a=int(input('Saisir une valeur entiere : ')) # a sera du type int
a=float(input('Saisir une valeur decimale : ')) # a sera du type float
```

2. TYPE « INT »

À partir de la version 3 de Python, il n'y a pas de limite (autre que la mémoire de l'ordinateur) au nombre de chiffres que peut avoir un entier :

```
print('2 puissance 100 =', 2**100)
    # affiche : 2 puissance 100 = 1267650600228229401496703205376
```

Le tableau ci-dessous recense les différentes opérations que l'on peut effectuer sur des entiers :

Op.	Description	Exemple
+	Calcule la somme de deux entiers	b = a + 2
-	Calcule la différence de deux entiers	b = 4 - a
*	Calcule le produit de deux entiers	b = 5 * a
/	Calcule le quotient décimal de deux entiers (le résultat est de type *float*)	b = 8/5 (b vaut alors 1.6)
//	Calcule le quotient entier de deux entiers (division « euclidienne »)	b = 8/5 (b vaut alors 1)
%	Calcule le reste de la division « euclidienne » de deux entiers	b = 8%5 (b vaut alors 3)

Python respecte l'ordre mathématique des calculs (parenthèses, puis puissances, puis multiplications et divisons, puis additions et soustractions). Par exemple :

```
a=12+5*(3-1)**3
print(a) # affiche 52
```

3. TYPE « FLOAT »

En Python, les nombres décimaux (*float*) s'écrivent en utilisant un point comme séparateur décimal (par exemple 1.2 au lieu de 1,2). Lorsqu'on veut définir un nombre entier comme nombre décimal, on peut le transtyper, mais il est plus simple d'ajouter « .0 » à la fin de ce nombre :

```
>>> type(5) # affiche : <class 'int'>
>>> type(float(5)) # affiche : <class 'float'>
>>> type(5.0) # affiche : <class 'float'>
```

Les opérations que l'on peut effectuer sur les nombres décimaux sont les mêmes que pour les nombres entiers à la différence près qu'elles renvoient un nombre décimal.

Par exemple :

```
>>> 2.5*1.2 # affiche 3.0
```

4. TYPE « STR »

Pour définir une chaîne de caractères (type : *str* pour *string*) en Python, il faut placer cette chaîne entre apostrophes (') ou entre guillemets ("). Par exemple :

```
a="Bonjour les amis" # a est de type str
b='Comment allez-vous?' # b est de type str
```

Si la chaîne de caractères comporte déjà une apostrophe ou un guillemet, cela peut être problématique : par exemple, l'instruction :

```
a='J'ai froid'
```

est incorrecte à cause de la présence d'une apostrophe dans le texte (Python croit alors que cette apostrophe indique la fin de la chaîne de caractères et il ne comprend pas la suite...) On peut résoudre ce problème de 3 manières différentes :

- alterner guillemets et apostrophes : `a="J'ai froid"` est correct
- *échapper* l'apostrophe en la faisant précéder d'un antislash (\) : *a='Jái froid'* est correct
- placer le texte entre **trois** apostrophes : `a='''J'ai froid'''` est correct

Une autre caractéristique de la notation avec les triples apostrophes est d'accepter plusieurs lignes de texte dans la chaîne.
Par exemple :

```
a= '''Ligne 1
Ligne 2'''
# a contient deux lignes de texte
```

Pour effectuer un saut de ligne, il est également possible d'utiliser le caractère spécial « \n ».
Par exemple :

```
a='Ligne 1\nLigne 2'
# a contient aussi deux lignes de texte
```

L'opérateur + permet de concaténer (mettre bout à bout) deux chaînes de caractère ; l'opérateur * (suivi ou précédé d'un nombre entier) permet de répéter plusieurs fois une chaîne :

```
a='Bonjour '
b='Vincent'
print(a+b) # affiche : Bonjour Vincent
print(3*a) # affiche : Bonjour Bonjour Bonjour
```