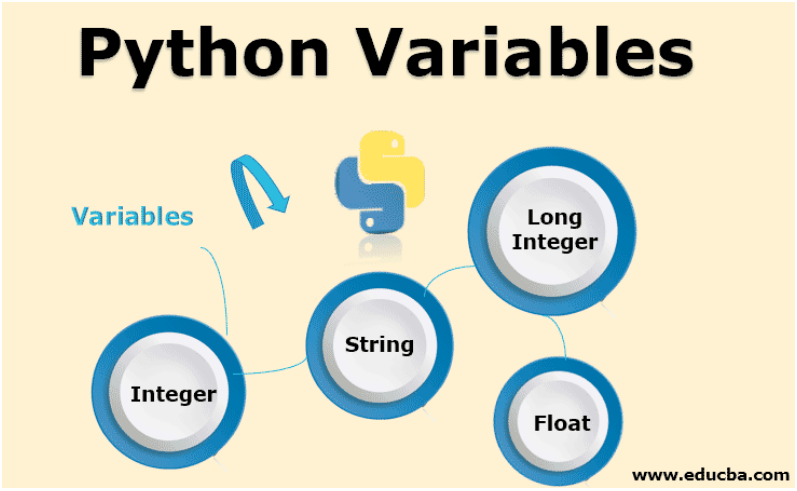
Expressions – Typage – Variables

```
>>> 2 > 8
False
>>> 2 <= 8 < 15
True
```


```
>>> 0b10110011 # binaire
179
>>> bin(179)
'0b10110011'
>>> 0o263  # octal
179
>>> oct(179)
'0o263'
>>> 0xB3   # hexadécimal
179
>>> hex(179)
'0xb3'
```



Expression, types et variables

1. Les expressions

Une expression est une suite de caractères définissant une valeur. Les expressions peuvent être simples ou complexes. Elles sont formées d'une combinaison de littéraux représentant directement des valeurs, des identificateurs et des opérateurs.

2. Typage des données

2.1. Les données entiers : INT

Le type `int` n'est limité en taille que par la mémoire de la machine. Il n'y a pas de distinction entre un entier court ou long (64 bits). Les entiers littéraux sont décimaux par défaut, mais on peut aussi utiliser les bases 2 et 16.

```
>>> 0b10110011 # binaire
179
>>> bin(179)
'0b10110011'
>>> 0o263  # octal
179
>>> oct(179)
'0o263'
>>> 0xB3   # hexadécimal
179
>>> hex(179)
'0xb3'
```

Un entier écrit en base 10 (par exemple 179) peut se représenter en binaire, octal et hexadécimal en utilisant les syntaxes ci contre :

```
>>> 2013       # décimal
2013
>>> 0b11111011101 # binaire
2013
>>> 0o3735     # octal
2013
>>> 0x7dd      # hexadecimal
2013
```

2.2. Les booléens : BOOL

✔ Deux valeurs possibles : `False`, `True`.
✔ Opérateurs de comparaison entre deux valeurs comparables, produisant un résultat de type bool :==, !=, >, >=, < ,<=
✔ Opérateurs logiques : `not`, `or` et `and`.

```
>>> (3 == 3) or (9 > 24)
True
>>> (9 > 24) and (3 == 3)
False
```

```
>>> 2 > 8
False
>>> 2 <= 8 < 15
True
```

2.3. Les données flottants : FLOAT

Un `float` est nombre à virgule, noté avec un point décimal (jamais avec une virgule) ou en notation exponentielle avec un « e » symbolisant le « 10 puissance » suivi des chiffres de l'exposant.
✔ Les flottants supportent les mêmes opérations que les entiers.
✔ Ils ont une précision finie limité.

```
2.718
.02
-1.6e-19
6.023e23
```

L'import du ***module math*** (avec ***import math*** en début de code) autorise toutes les opérations mathématiques usuelles. N'oubliez pas de taper math.(opération). Mais mieux vaut placer en début de programme : `from math import*`

Expression, types et variables

```
>>> import math
>>> print(math.sin(math.pi/4))
0.707106781187
>>>> print(math.degrees(math.pi))
180.0
>>>> print(math.factorial(9))
362880
>>> print(math.log(1024, 2))
10.0
```

```
>>> from math import*
>>> print(sin(pi/4))
0.7071067811865475
>>> print(degrees(pi))
180.0
>>> factorial(9)
362880
>>> log(1024,2)
10.0
>>>
```

2.4. Les données complexes

✔ Les complexes sont écrits en notation cartésienne formée de deux flottants.
✔ La partie imaginaire est suffixée par `j`.
✔ Un module mathématique spécifique (cmath) leur est réservé : `from cmath import*`.

Pour afficher le type, les parties réelles et imaginaires du complexe z=1.5 + 0.5i :

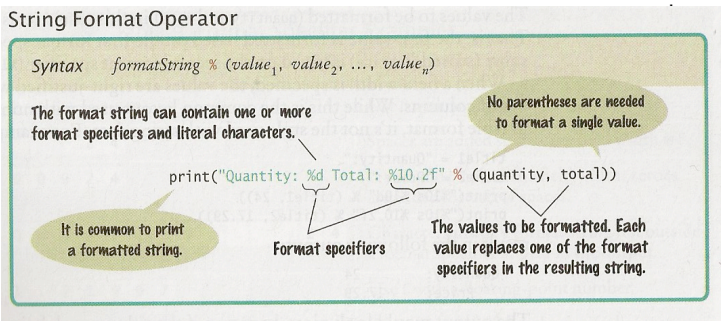
```
1  z=1.5+0.5j
2  print type(z) # affichage : <type 'complex'>
3  print z.real # affichage : 1.5
4  print z.imag # affichage : 0.5
5  print z.conjugate() # affichage : 1.5-0.5j
```

```
>>> print(1j)
1j
>>> print((2+3j) + (4-7j))
(6-4j)
>>> print((9+5j).real)
9.0
>>> print((9+5j).imag)
5.0
>>> print((abs(3+4j))) # module
5.0
```

2.5. Les chaines de caractères : STR

Les chaînes de caractères dans Python sont typés : `str` ***(string)***

```
>>> a="bonjour"
>>> type(a)
<class 'str'>
```



Expression, types et variables

Table 9 Format Specifier Examples

Format String	Sample Output	Comments
"%d"	24	Use d with an integer.
"%5d"	␣␣␣24	Spaces are added so that the field width is 5.
"%05d"	00024	If you add 0 before the field width, zeroes are added instead of spaces.
"Quantity:%5d"	Quantity:␣␣24	Characters inside a format string but outside a format specifier appear in the output.
"%f"	1.219997	Use f with a floating-point number.
"%.2f"	1.22	Prints two digits after the decimal point.
"%7.2f"	␣␣␣1.22	Spaces are added so that the field width is 7.
"%s"	Hello	Use s with a string.
"%d %.2f"	24 1.22	You can format multiple values at once.
"%9s"	␣␣␣␣Hello	Strings are right-justified by default.
"%-9s"	Hello␣␣␣␣	Use a negative field width to left-justify.
"%d%%"	24%	To add a percent sign to the output, use %%.

3. Les variables en algorithmique

Dès que l'on possède des types de données, on a besoin des ***variables*** pour stocker les données. Une variable est une représentation idéale d'une zone mémoire (adresse) de l'ordinateur. Il s'agit d'un endroit (emplacement précis dans mémoire vive) où l'on peut stocker une valeur, y accéder et changer cette valeur.

4. Déclaration des variables – début de l'algorithme

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette. Ceci se fait tout au début de l'algorithme, avant même les instructions proprement dites. C'est ce qu'on appelle la déclaration des variables.

Les noms de variables sont des noms que vous choisissez vous-même assez librement. Efforcez-vous cependant de bien les choisir : de préférence assez courts, mais aussi explicites que possible, de manière a exprimer clairement ce que la variable est censée contenir. Par exemple, des noms de variables tels que `altitude`, `altit` ou `alt` conviennent mieux que `x` pour exprimer une altitude.

Un bon programmeur doit veiller à ce que ses lignes d'instructions soient faciles à lire.

Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

✔ Un nom de variable est une séquence de lettres (a → z, A → Z) et de chiffres (0 → 9), qui doit toujours commencer par une lettre.

✔ Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que `$`, `#`, `@`, etc. sont interdits, a l'exception du caractère _ (souligne).

✔ La casse est significative (les caractères majuscules et minuscules sont distingués). Attention : `Joseph`, `joseph`, `JOSEPH` sont donc des variables différentes. Soyez attentifs !

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans `tableDesMatieres`.

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme nom de variables les 33 « mots réservés » ci-après (ils sont utilisés par le langage lui-même) :

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

5. *L'affectation (ou assignation)*

Nous savons désormais comment choisir judicieusement un nom de variable. Voyons à présent comment nous pouvons ***définir une variable et lui affecter une valeur***. Les termes ≪ affecter une valeur ≫ ou ≪ assigner une valeur ≫ à une variable sont équivalents. Ils désignent l'opération par laquelle ***on établit un lien entre le nom de la variable et sa valeur (son contenu).***
En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe =

```
>>> n = 7                       # définir la variable n et lui donner la valeur 7
>>> msg = "Quoi de neuf ?"      # affecter la valeur "Quoi de neuf ?" à msg
>>> pi = 3.14159                # assigner sa valeur à la variable pi
```

Les trois instructions d'affectation ci-dessus ont eu pour effet chacune de réaliser plusieurs opérations dans la mémoire de l'ordinateur :
✔ ***Créer et mémoriser*** un nom de variable ;
✔ Lui attribuer ***un type*** bien déterminé (ce point sera explicite a la page suivante) ;
✔ Créer et mémoriser une ***valeur particulière*** ;
✔ Etablir ***un lien*** (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante.

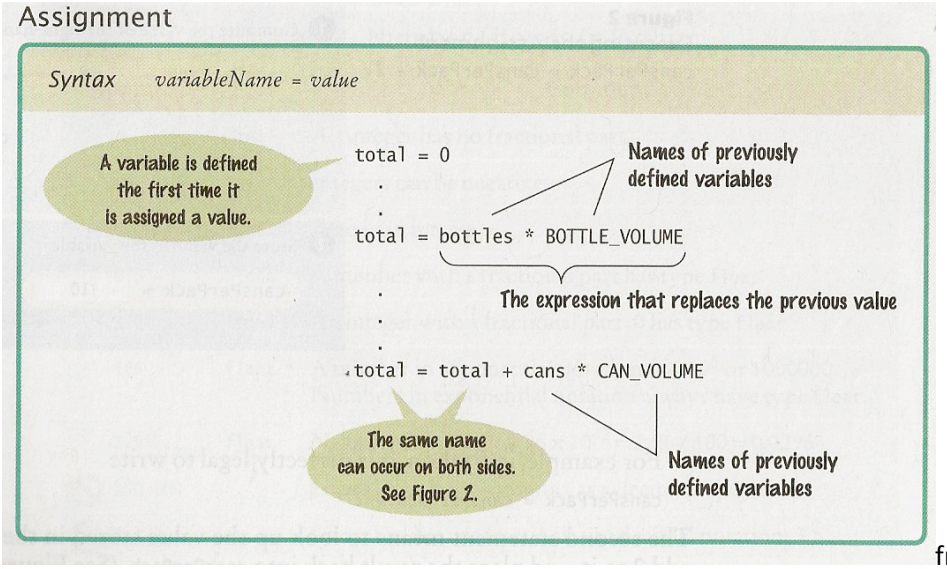
Dans Python, on peut affecter une valeur à plusieurs variables simultanément :

```
>>> x = y = 7
>>> x
7
>>> y
7
```

On peut aussi effectuer des affectations parallèles à l'aide d'un seul operateur :

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Expression, types et variables



fr'

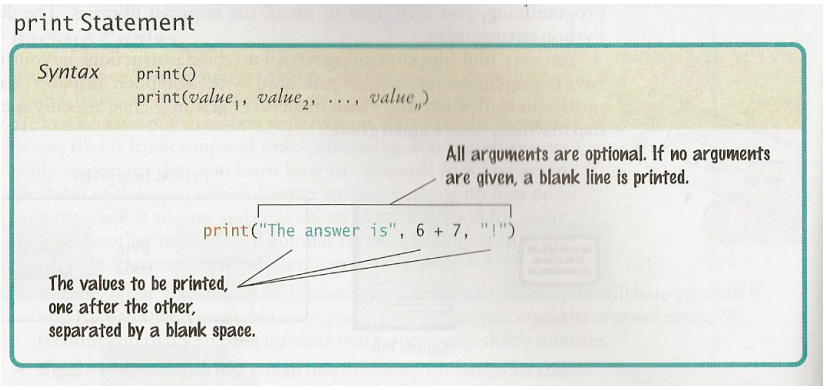
6. *L'appel de variable à l'utilisateur :* INPUT

L'appel de variable à l'utilisateur (entrée) est réalisée par la fonction « input »

```
# Saisie des donnees

H = float(input("Hauteur de l\'echantillon : en cm : "))
D = float(input("Diametre de l\'echantillon : en cm : "))
```

7. *L'affichage de résultats:* PRINT



L'affichage d'un résultat (sortie) est obtenu en tapant le nom de la variable directement dans l'IDLE ou en passant par l'opérateur « print » :

```
>>> msg="quoi de neuf"
>>> msg
'quoi de neuf'
>>> print(msg)
quoi de neuf
>>>
```

Ou encore mieux avec un texte introductif:

```
>>> print("le message est:",msg)
le message est: quoi de neuf
>>>
```

8. Ordre des instructions

Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final. Considérons les deux algorithmes suivants :

```
Variable A en Entier          Variable A en Entier
Début                           Début
A ← 34                           A ← 12
A ← 12                           A ← 34
Fin                             Fin
```

Il est clair que dans le premier cas la valeur finale de A est 12, dans l'autre elle est 34 .

Super intéressant pour visualiser les affectations : http://www.pythontutor.com/visualize.html#mode=display