



Premiers pas

DÉCLARER DES VARIABLES

Pour chaque commande, prévoir ce qu'elle fait puis la taper en console et commenter ce document.

```
>>> a = 0.5
>>> print(a)
>>> (a, b) = (1, 2)
>>> (a, b) = (b, a)
>>> (a, b) = (a, a+b)
>>> c = a*b
>>> c = a**b
>>> 25//4
>>> 25%4
```

Principales opérations sur les nombres	
somme	
différence	
produit	
division	
quotient de la division euclidienne	
reste de la division euclidienne	

Continuons avec quelques commandes

```
>>> sin(5)
>>> import math
>>> math.sin(5)
>>> math.sin(math.pi)
```

Pour les commandes moins « grand public », il existe des bibliothèques que l'on appelle selon les besoins afin de ne pas saturer inutilement la mémoire de l'ordinateur. Ici, `import math` charge en mémoire de nouvelles fonctions mathématiques.

Pour se souvenir qu'elles proviennent de la bibliothèque `math`, on les utilise avec le préfixe `math`.

BOOLÉENS ET OPÉRATEURS LOGIQUES

Tapez les commandes une par une après avoir anticipé mentalement la réponse. Vous êtes invités à faire le lien avec le cours de logique.

```
>>> (3>2) and (5>3)
>>> (3>2) and (a>3)
>>> a>3>2
>>> a<3>2
>>> (3>2) or (a>3)
>>> (3>2) and (2<1 or 4==2+2)
>>> (True == False) == False
```

Ceci servira pour les strictures conditionnelles et les boucles `while`.

CHAÎNES DE CARACTÈRES

On peut définir une chaîne de caractères avec des guillemets simples ou doubles. Par exemple, tapez les commandes suivantes et commentez :

```
>>> a = 'Bonjour'
>>> b = "les BCPST"
>>> c = a+b
>>> print(c)
>>> print(3*a)
```

Quand on ajoute des chaînes de caractères on effectue en fait une **concaténation**. Il faut donc penser à adapter les espaces si besoin.

Noter que ces commandes ont « écrasé » les anciennes valeurs de `a` et de `b`.

Enregistrer un programme

Dans la fenêtre de gauche, recopier le code ci-dessous et enregistrer sous TP01-01.py (menu Fichier) puis exécuter (menu Exécuter)

```
# Déclarer des variables numériques
H2O_nb_O = 1
H2O_nb_H = 2
H2O_densite = 1000
masse_O = 15.9994
masse_H = 1.00794
avogadro = 6.023e23
masse_H2O = H2O_nb_O * masse_O + H2O_nb_H * masse_H
print('Masse moléculaire de H2O = ', masse_H2O)
```

Enregistrer son travail permet de retrouver des codes afin de les réutiliser ultérieurement. Quand vous aurez des travaux importants à sauvegarder, faites-en plusieurs copies.

Les variables ainsi définies permettent de réutiliser des valeurs prédéfinies plus tard. Pour la lisibilité de votre code, veillez à **choisir des noms de variable évocateurs**. Les noms ci-dessus sont peut-être un peu longs, mais très explicites.

Dans un projet s'étalant sur plusieurs mois, vous serez agacés de rechercher quelle variable représente quoi donc, **évités les noms à une lettre** (sauf pour des calculs rapides à usage unique).

Tout ce qui suit un symbole # sera considéré comme un commentaire et non exécuté.

Pour des commentaires plus longs, on peut utiliser dans un fichier

```
"""
-----
Voici un commentaire
plus long qui s'étale sur plusieurs lignes
-----
"""
```

Il ne sera pas exécuté et permet d'expliquer ce que vous faites à un éventuel collaborateur... ou membre de jury.

→ **Prenez l'habitude de commenter votre code !**

CRÉER SES FONCTIONS

On peut avoir besoin de créer ses propres fonctions pour :

- avoir un code réutilisable (quand on a plusieurs fois le même calcul à faire)
- séparer les fonctionnalités d'un programme et avoir un code plus lisible.
- gérer la complexité en encapsulant les fonctions
- décomposer le code en plusieurs fichiers, l'un pouvant contenir les fonctions.
- ...

UN PREMIER EXEMPLE

```
def calculVolumeMolaire(masse, densite):
    volume = masse/densite
    return volume
```

L'indentation (le fait de décaler certains débuts de lignes) ne sert pas qu'à faire joli mais **fait partie de la syntaxe**. Il est très important de la respecter.
L'unité d'indentation est 4 espaces.

Attention à l'indentation

```
def ImportanceIndentation():
    print('Ce code indenté fait partie de la fonction')
    print('Ce code aussi...')
print('Cette ligne non indentée ne fait pas partie de la fonction !')
```

On peut compléter le code précédent en

```
def calculVolumeMolaire(masse, densite):
    volume = masse/densite
    return volume
```

```
VolumeMolaireH2O = calculVolumeMolaire(masse_H2O, H2O_densite)
print('Volume de 1 mole of H2O = ',VolumeMolaireH2O,'L')
```

ce code doit être exécuté à la suite des codes précédents, sinon, certaines variables ne seront pas définies.

UNE FONCTION PEUT APPELER UNE AUTRE FONCTION

```
# une fonction calculant le nombre de molécule par litre
def moleculesParLitre(masse,densite):
    VolumeMolaire = calculVolumeMolaire(masse, densite)
    nombreDeMoles = 1.0 / VolumeMolaire
    nombreDeMolecules = avogadro * nombreDeMoles
    return nombreDeMolecules
```

ENTRÉES

```
masse = int(input("entrez une masse d'eau (en grammes) : "))

print("Dans ", masse, " grammes d'eau, il y a ",
      moleculesParLitre(masse, H2O_densite), "molécules d'eau.")
```

input

Pour que le programme demande d'entrer une valeur, on utilise input. Par défaut la réponse est de type « chaîne de caractères » et il faut parfois (comme ici) convertir. La commande int permet de transformer la réponse en nombre entier.

pas de input dans une fonction

En général, une fonction ne comportera pas de « input ».

Les valeurs des paramètres d'une fonction sont précisées lors de l'appel de celle-ci.

À retenir : syntaxe d'une fonction

```
def nomfonction(var1, var2):
    ... commandes
    ... commandes
    ... commandes
    (return sortie)
```

où les entrées var1, var2 (il peut y en avoir plus, ou aucune) peuvent être utilisées dans les commandes. Pour renvoyer quelque-chose, la fonction doit exécuter « return » suivi d'une valeur (booléen, float, integer, string...)

Exercices

exercice 1 Écrire un programme demandant l'année de naissance et affichant une phrase donnant l'âge d'une personne née cette année là.

Le programme ne fonctionnera plus l'année prochaine. On ne testera qu'avec des années passées.

exercice 2 1) Écrire une fonction `distance(x,y)` calculant la distance à l'origine du point $M(x,y)$.

2) Écrire une fonction `distance2(xA,yA,xB,yB)` calculant la distance entre les points $A(x_A, y_A)$ et $B(x_B, y_B)$.

exercice 3 Écrire une fonction `trinome(a,b,c)` renvoyant les valeurs des racines distinctes de l'équation $ax^2 + bx + c = 0$ (on supposera que le discriminant Δ est positif)

Pour approfondir...

taper les commandes suivantes, et en tirer les conséquences qui s'imposent sur les différents types gérés par Python.

```
a = 2
type(a)
type(2*3)
type(2**3)
type(3/2)
a = 4/2
type(a)
a = 4//2
type(a)
a = int(4/2)
type(a)
from math import *
sin(pi)
2*sin(pi/4) == sqrt(2)
```

VARIANTES D'APPEL DE BIBLIOTHÈQUES

	<code>import math as m</code>	<code>from math import *</code>
ex	<code>m.sin(m.pi)</code> <code>m.exp(-2)</code>	<code>sin(pi)</code> <code>exp(-2)</code>
+	• plus explicite • pas de risque d'effacer une fonction	• plus simple
–	• plus lourd	• quelle bibliothèque ? • risques d'incompatibilités