MEMENTO PYTHON 3

Version 2.00 - O. Chaumette
Lycée JP SARTRE – 69500 BRON

TYPE et CONVERSIONS DE VARIABLES

Types courants : *Entier* (***int***) *Réel* (***float***) *Texte* (***str***) *Booléen* (***bool***)

Déclaration de variables :

```
T = "Bonjour"
N = 2
```
Assigne à la variable texte (string) *T* le texte « *Bonjour* »
Assigne à la variable entière (int) *N* la valeur *2*

Conversion de variables :

```
T = str(N)
N = int(T)   ou   N = float(T)
```
transforme **un entier** (int) ou réel (float) N **en texte** (string)
transforme **un texte** T (string) **en entier** (int) ou réel (float)

ENTREES et SORTIES : demander une valeur à l'utilisateur et afficher à l'écran.

Entrée au clavier :

```
Un_Texte = input("question")
Un_Entier = int(input("question"))
```
Pose « *question* » à l'utilisateur. Réponse dans la **variable « texte »** : Un_Texte
Réponse dans la **variable « entière »** : un_entier

Sortie écran :

```
print("Texte",variable)
print("Bonjour"+texte)
print("Bonjour"+str(Nombre))
print("Bonjour", end=' ')
print("Bonjour", end=',')
print("Bonjour\nle\nmonde")
print("%.2e"%N)
```
écrit sur l'écran *Texte* suivi du contenu de *Variable* séparés par un espace
écrit sur l'écran *Bonjour* suivi du contenu de *texte.* ***Réservé aux textes***
écrit sur l'écran *Bonjour* suivi de *Nombre* transformé en texte
écrit *Bonjour* suivi d'un espace sans retour à la ligne (le prochain print sera collé)
écrit *Bonjour* suivi d'une virgule sans retour à la ligne (le prochain print sera collé)
écrit *Bonjour* suivi d'un saut de ligne (**\n**) puis *le* puis saut de ligne puis *monde*
Ecrit le réel (ou entier) *N* en écriture scientifique avec 2 décimales (donc 3 chiffres significatifs)

TESTS et CONDITIONS :

Test simple:

```
if Condition:
    Instructions si « Condition » est vraie
```
bien penser aux « : »!
bien penser à l'indentation !!!

Test avec SINON (else):

```
if Condition:
    Instructions si « Condition » est vraie
else:
    Instructions si « Condition » est fausse
```
bien penser à l'indentation et aux « : »

Test avec SINON SI (else if):

```
If Condition1:
    Instructions si « Condition1 » est vraie
elif Condition2:
    Instructions si « Condition2 » est vraie
else:
    Instructions si « Condition1 et 2 » sont fausses
```
bien penser à l'indentation et aux « : »

Test avec conditions multiples:

```
if Condition1 and/or Condition2:
    Instructions
```
and: condition 1 **ET** 2 respectées
or: condition 1 **OU** 2 respectées

Opérateurs dans les conditions: **ATTENTION le signe = est réservé à l'affectation de variables**

== :égal **!=** :différent **not** : contraire de la condition
> (ou **<**):supérieur (ou inférieur) **>=** (ou **<=**):sup (ou inf) ou égal

On peut utiliser un intervalle : exple : `if 2<x<3:` *Instructions*

Variable Booléenne

```
Mon_Booleen = (A==B)
```
la variable *Mon_Booleen* prendra la valeur *True* si A égal B, *False* sinon.

INCREMENTATION d'une variable : Changer sa valeur en fonction de sa valeur initiale.

Var += 1 incrémente la variable de 1 : cela remplace **Var = Var +1.** (Cas g[ral] : **Var+=***Valeur*)

Cela marche aussi avec une chaine (**Chaine +=** *Texte* revient à écrire **Chaine = Chaine +** *Texte*)

Il existe aussi : **Var -=** *Valeur* (pour soustraire *Valeur* à Var), **Var *=** *Valeur* (pour multiplier Var par *Valeur*)

LISTES : Ce sont des tableaux.

Déclaration :

```
Liste = ["a","b","c"]
Liste = [2,4,8]
Liste = []
Liste2D = [[1,2,3],[4,5,6]]
```
Crée une liste de textes (type ***string*** car entre guillemets)
Crée une liste d'entiers (type ***int*** car pas de guillemets)
Crée une liste vide
Crée une liste d'entiers à 2 dimensions (ici 2 lignes et 3 colonnes)

Accès aux éléments de la liste :

```
Liste[index]
Liste[index1:index2]
Liste[index1:]
Liste[:index2]
Liste[-1]
Liste2D[n°_ligne][n°_col]
```
Renvoie l'élément situé à l'emplacement *index* (qui commence à 0)
Renvoie les éléments entre *index1* et *index2*
Renvoie les éléments à partir d'*index1* jusqu'à la fin
Renvoie les éléments du début jusqu'à *index2*
Renvoie le dernier élément (si -2 : l'avant dernier etc…)
Renvoie l'élément situé à *n°_ligne* (en commençant à 0) et *n°_col* (début 0)

Manipulation de listes :

```
len(Liste)
Liste.append(Elément)
Liste.remove(Elément)
Liste [pos] = "a"
Liste.insert(pos,Elément)
Liste.index(Elément)
Liste.count(Elément)
Elément in Liste
```
Renvoie la longueur (= **nbre d'éléments**) de la liste. C'est un **entier**.
Ajoute *Elément* **à la fin** de la liste
Enlève la 1ère occurrence de *Elément*
met 'a' à la position *pos* (début=0) (**en écrasant** l'él[t] qui s'y trouve)
Insère *Elément* à la position *pos* (**sans écraser.** Cela décale les autres)
Renvoie l'emplacement de *Elément* dans la liste (**1ère position = index 0**).
Renvoie le nombre de fois qu'*Elément* est présent dans la liste
Renvoie si *Elément* est présent dans la liste (True/False). Utile dans un **if** :
Exemple : `if Lettre in Mot` (si *Lettre* est présent dans *Mot*)

Parcourir une liste :

```
for Element in Liste:
```
Element prend successivement le contenu de chaque élément de la liste

ASTUCE : *Afficher les éléments d'une liste les uns à la suite des autres (séparés par un espace):*

```
T = ""
for Element in Liste:
    T = T +" "+ Element
print (T)
```
← remplacer ici `Element` par `str(Element)` si la liste contient des entiers

CHAINES (ou TEXTES): Ce sont des listes de caractères donc ont les mêmes fonctions + des fonctions suppl. :

Mise en forme :

```
Texte = "Bonjour \nSalut"
Texte = "Bonjour \"Salut\""
```
Saute une ligne après le \n
Saute des guillemets à la place de \"

Manipulations fréquentes de chaines :

```
Texte.lower()
Texte.upper()
Texte.split()
Texte.split(Caractère)
Texte.find(Mot)
Texte.replace(MotOld,MotNew)
```
Met le texte en minuscule
Met le texte en majuscule
Renvoie **une liste** contenant les mots du texte (s'ils st séparés par « **espace** »)
Renvoie **une liste** contenant les mots du texte (s'ils st séparés par ***Caractère***)
Renvoie le + petit index de *Mot* dans Mon_Texte (ou **-1 si pas trouvé**)
Remplace *MotOld* par *MotNew* dans Mon_Texte

```
Texte [:n]
Texte [Len(Mon_Texte)- n:]
Texte [n1:n2]
```
Renvoie les *n* premiers caractères en partant de la gauche
Renvoie les *n* premiers caractères en partant de la droite
Renvoie les caractères entre les rangs *n1* et *n2*

```
Texte.startswith(Mot)
Texte.endswith(Mot)
```
Renvoie True si Mon_Texte commence par *Mot*
Renvoie True si Mon_Texte finit par *Mot*

Utilisation des codes ASCII (des caractères d'un texte) :

```
Caractere = chr(Code_Ascii)
Code_ASCII = ord(Caractère)
```
Renvoie le caractère (type **str**) dont le code ascii est *Code_Ascii*
Renvoie le code ascii de *Caractère*.

Codes ASCII utiles : **"A"**:65 **"Z"**:90 **"a"**:97 **"z"**:122 **"0"**:48 **"9"**:57 espace :32

BOUCLES :

Boucle FOR générale: Dans le cas où on connaît le nombre de répétitions

```
for Variable in Ensemble de valeurs:
        Instructions
```

Variable va prendre toutes les valeurs de « ensemble de valeurs »
bien penser à l'indentation !!!

Boucle FOR avec des nombres:

```
for Compteur in range(Nombre):
```
Compteur varie de ***0*** à ***Nombre-1***

```
for Compteur in range(début,fin):
```
Compteur varie de ***début*** à ***fin-1***

```
for Compteur in range(début,fin,pas):
```
Compteur varie de ***début*** à ***fin-1*** par sauts de ***pas***

```
for Compteur in range(fin,début,-1):
```
Compteur varie de ***fin*** à ***début+1*** (donc **à l'envers**)

Boucle FOR avec des listes/chaines:

```
for Variable in Liste:
```
Variable prend la valeur de chaque élément de la liste *Liste*

Boucle WHILE (tant que) : Dans le cas où on ne connaît **pas** le nombre de répétitions

```
while Condition:
```
bien penser aux « : » et à l'indentation

Instructions tant que « Condition » est vraie

Modifier la variable intervenant dans la condition sinon la boucle serait infinie !

Sortie « artificielle » de boucle :

`break` arrête la boucle

OPERATEURS et FONCTIONS MATHÉMATIQUES:

Division:

`Nbre1 // Nbre2`	Renvoie la **partie entière** de la division de *Nbre1* par *Nbre2*
`Nbre1 % Nbre2`	Renvoie le **reste de la division** de *Nbre1* par *Nbre2*
`divmod(Nbre1, Nbre2)`	Renvoie un tuple (Résultat Division entière,Reste)

Fonctions:

`round(x)`	arrondit un "réel" *x* vers l'entier le plus proche
`round(x, n)`	arrondit un "réel" à la décimale *n*. *n* négatif permet un arrondi à la dizaine, centaine... près.
`**`	marque l'**exposant** et a la priorité sur +, -, *, /
`pow(x,y)`	renvoie *x* à la puissance *y*, équivaut à *x* ** *y*
`max(x, y)`	renvoie la plus grande des deux valeurs
`min(x, y)`	renvoie la plus petite des deux valeurs
`abs()`	renvoie la valeur absolue d'un nombre (sans le signe)

MODULE MATH : en complément des fonctions précédente. Pour les réels.

Déclaration :

```
import math
```

Fonctions :

`math.pi`	retourne une approximation de la constante pi: 3.1415926535897931
`math.degrees()` et `radians()`	transforment en degrés ou en radians
`math.cos(),.sin(),.tan()`	fonctions trigonométriques usuels
`math.acos(),.asin(),.atan()`	fonctions trigonométriques inverses
`math.exp()`	exponentielle
`math.log()` et `math.log10()`	logarithme népérien et décimal
`math.sqrt()`	renvoie la racine carrée

MODULE RANDOM : hazard

Déclaration :

```
import random
```

Fonctions :

`random.choice(Ma_Liste)` choisit un élément de la liste `Ma_Liste`
`random.sample(Ma_Liste,n)` renvoie une liste de *n* éléments choisis dans `Ma_Liste`
`random.shuffle(Ma_Liste)` mélange les éléments de `Ma_Liste`
`random.randrange(borne1,borne2)` renvoie un entier au hasard entre *borne1* (incluse) et *borne2* (exclue)

FONCTIONS : Ce sont des parties de programme que l'on peut appeler régulièrement selon les besoins.

Définition d'une fonction SANS return (=procédure) : c'est un morceau de programme qu'on peut appeler

```
def Ma_fonction():
    Liste d'instructions
```
Attention à l'indentation

Appel de la fonction:

```
Ma_fonction()
```
à placer dans le prg principal : Exécute le code de « Ma_Fonction »

Définition d'une fonction AVEC return:

```
def Ma_fonction():
    Liste d'instructions qui crée une «Variable_Résultat »
    return Variable_Résultat
```

Appel de la fonction :

```
Variable = Ma_fonction()
```
met dans *Variable* le résultat des instructions de *Ma_fonction*

Définition d'une fonction avec des ARGUMENTS (ou paramètres):

```
def Ma_fonction(paramètre1, paramètre2,…):
    Liste d'instructions qui crée une «Variable_Résultat »
    en utilisant paramètre1, paramètre2
    return Variable_Résultat
```

Appel de la fonction AVEC paramètres :

```
Variable = Ma_fonction(p1,p2…)
```
met dans *Variable* le résultat des instructions de *Ma_fonction*
(ayant utilisé les paramètres p1, p2 etc…)

Utilisation d'une variable du programme principal dans une fonction :

Les Variables d'une fonction et celles du programme principal sont indépendantes. Si on peut utiliser une variable du programme principal dans une fonction, il faut ajouter dans la fonction :

```
global Nom_de_la_Variable_du_prg_Principal
```

FICHIERS : LECTURE/ECRITURE simple de textes

Création/ouverture d'un fichier « Nom_de_Fichier.ext » :

`Mon_fichier = open('Nom_de_Fichier.ext','w')`	ouverture en écriture (efface l'existant si déjà présent)
`Mon_fichier = open('Nom_de_Fichier.ext','a')`	ouverture en modification (pour ajouter des données)
`Mon_fichier = open('Nom_de_Fichier.ext','r')`	ouverture en lecture

Fermeture d'un fichier : ***OBLIGATOIRE s'il a été ouvert !!***

```
Mon_fichier.close()
```

Ecriture de données dans « Mon_Fichier » : *il faut que « Mon_fichier » ait été ouvert (en écriture ou modif)*

```
Mon_fichier.write(Texte)
```
Ecrit « Texte » à la suite du fichier ; sans saut de ligne !

Pour ajouter une saut de ligne : *Texte* doit être : `'\n'`
Pour ajouter une tabulation : *Texte* doit être : `'\t'`

Lecture de données dans « Mon_Fichier » : *il faut que « Mon_fichier » ait été ouvert (en lecture)*

`Variable = Mon_fichier.read()`	Met dans « Variable » tout le contenu du fichier
`Variable = Mon_fichier.read(N)`	Met dans « Variable » les *N* caractères du fichier (à partir de la position atteinte dans le fichier)
`Variable = Mon_fichier.readline()`	Met dans « Variable » une ligne du fichier (à partir de la position atteinte dans le fichier)
`Liste = Mon_fichier.readlines()`	Met toutes les lignes du fichier dans une liste.

Changement du dossier dans lequel on lit/écrit le fichier :

```
import os
os.chrdir(Dossier_Voulu)
```
Dossier_Voulu peut être absolu: **"C:/Mon/Dossier"** ou relatif: **"/Mon/Dossier"**