MEMENTO PYTHON

TYPE et CONVERSIONS DE VARIABLES

Types courants :

Entier (*int*)

Réel (*float*)

Chaîne de caractères - Texte (*str*)

Booléen (*bool*)

Déclaration de variables :

```
T = "Bonjour"
```

assigne à la variable texte (string) *T* le texte « *Bonjour* »

```
N = 2022
```

assigne à la variable entière (int) *N* la valeur 2022

pas d'accent - pas d'espace - pas d'instruction python - différence majuscule/minuscule (cas) importante.

Conversion de variables :

```
T = str(N)
```

converti un entier (int) ou réel (float) *N* en chaîne de caractères (string) (avec assignation)

```
N = int(T)
```

converti une chaîne de caractères *T* (string) en entier (int) (avec assignation)

```
N = float(T)
```

converti une chaîne de caractères *T* (string) en réel (float) (avec assignation)

ENTRÉES et SORTIES (demander une valeur à l'utilisateur et afficher à l'écran)

Entrée au clavier :

```
Un_Texte = input("question")
```

pose « *question* » à l'utilisateur

réponse affectée dans la variable de type chaîne de caractères : *Un_Texte*

```
Un_Entier = int(input("question"))
```

pose « *question* » à l'utilisateur

réponse affectée dans la variable de type entier : *Un_Entier*

Affichage écran :

```
print("Texte", Variable)
```

affiche *Texte* suivi du contenu de *Variable* séparés par un espace

```
print("Bonjour"+Texte.)
```

affiche *Bonjour* suivi du contenu de *Texte*. (concaténation de caractères)

```
print("Bonjour"+str(Nombre))
```

affiche *Bonjour* suivi de *Nombre* converti en chaîne de caractères

```
print("Bonjour", end=' ')
```

affiche *Bonjour* suivi d'un espace sans retour à la ligne (le prochain print sera collé)

```
print("Bonjour", end=',')
```

affiche *Bonjour* suivi d'une virgule sans retour à la ligne (le prochain print sera collé)

```
print("Bonjour\nle\nmonde")
```

affiche *Bonjour* suivi d'un saut de ligne (*\n*) puis le saut de ligne puis *monde*

```
print("%.2e"%N)
```

affiche le réel (ou entier) *N* en écriture scientifique avec 2 décimales

TESTS et CONDITIONS

Test simple:

```
if Condition:
```

```
    |Instructions si « Condition » est vraie
```

attention au respect de l'indentation

Test avec SINON (else):

```
if Condition:
```

```
    |Instructions si « Condition » est vraie
```

attention au respect de l'indentation

```
else:
```

```
    |Instructions si « Condition » est fausse
```

attention au respect de l'indentation

Test avec SINON SI (else if):

```
if Condition1:
```

```
    |Instructions si « Condition1 » est vraie
```

attention au respect de l'indentation

```
elif Condition2:
```

```
    |Instructions si « Condition2 » est vraie
```

attention au respect de l'indentation

```
else:
```

```
    |Instructions si « Condition1 et 2 » sont fausses
```

attention au respect de l'indentation

Test avec conditions multiples:

and: *Condition1 ET Condition2* respectées **or:** *Condition1 OU Condition2* respectées

```
if Condition1 and/or Condition2:
```

```
    |Instructions
```

attention au respect de l'indentation

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 **<=**):supérieur (ou inférieur) ou égal

On peut utiliser un intervalle : exemple :

```
if 2<x<3:
```

```
    |Instructions
```

attention au respect de l'indentation

Variable Booléenne

```
Mon_Booleen = (A==B)
```

la variable *Mon_Booleen* prendra la valeur *True* si *A* égal *B*, *False* sinon.

INCRÉMENTATION d'une variable (Changer la valeur d'une variable en fonction de sa valeur initiale)

```
Var += 1 # incrémente la variable de 1 ,(Var = Var +1) .
Var+=Valeur # incrémente la variable de 1 ,(Var = Var + Valeur) .
Cela fonctionne aussi avec une chaîne de caractères (Chaîne += Texte revient à écrire Chaîne = Chaîne + Texte)
Il existe aussi : Var -= Valeur (pour soustraire Valeur à Var), Var *= Valeur (pour multiplier Var par Valeur)
```

LISTES

Déclaration :

```
Liste = ["a", "b", "c"] # crée une liste de chaînes de caractères (type string)
Liste = [2, 4, 8] # crée une liste d'entiers (type int)
Liste = [] # crée une liste vide
Liste2D = [[1, 2, 3], [4, 5, 6]] # crée une liste d'entiers à 2 dimensions (ici 2 lignes et 3 colonnes)
```

Accès aux éléments de la liste :

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

Manipulation de listes :

```
len(Liste) # renvoie la longueur (= nombre d'éléments) de la liste Liste. C'est un entier.
Liste.append(Elément) # ajoute Elément à la fin de la liste Liste
Liste.remove(Elément) # enlève la 1ère occurrence de Elément
Liste[pos] = "a" # réaffecte 'a' à la position pos (début=0) (en écrasant l'élément qui s'y trouve)
Liste.insert(pos, Elément) # insère Elément à la position pos (sans écraser et en décalant les autres)
Liste.index(Elément) # renvoie l'emplacement de Elément dans la liste Liste (1ère position = index 0).
Liste.count(Elément) # renvoie le nombre de fois qu'Elément est présent dans la liste Liste
Elément in Liste # renvoie si Elément est présent dans la liste Liste (True/False). Utile dans un if :
```

Parcourir les éléments une liste :

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

Parcourir les indices d'une liste :

```
for pos in range (len(Liste)): # pos prend successivement les indices de chaque élément de la liste Liste
```

CHAÎNES DE CARACTÈRES (ou TEXTES)

Les instructions précédentes des listes sont utilisables aussi par les chaînes de caractères. Ces dernières peuvent recourir à d'autres fonctions supplémentaires.

Mise en forme :

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

Manipulations fréquentes de chaînes :

```
Texte.lower() # met le texte en minuscule
Texte.upper() # met le texte en majuscule
Texte.split() # renvoie une liste contenant les mots du texte (s'ils st séparés par « espace »)
Texte.split(Caractère) # renvoie une liste contenant les mots du texte (s'ils st séparés par Caractère)
Texte.find(Mot) # renvoie le plus petit index de Mot dans Texte (ou -1 si pas trouvé)
Texte.replace(MotOld, MotNew) # remplace MotOld par MotNew dans Texte
Texte[:n] # renvoie les n premiers caractères en partant de la gauche
Texte [Len(Mon_Texte)- n:] # renvoie les n premiers caractères en partant de la droite
Texte [n1:n2] # renvoie les caractères entre les rangs n1 et n2
Texte.startswith(Mot) # renvoie True si Texte commence par Mot
Texte.endswith(Mot) # renvoie True si Texte finit par Mot
```

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

```
Caractere = chr(Code_Ascii) # renvoie le caractère (type str) dont le code ascii est Code_Ascii
Code_ASCII = ord(Caractère) # 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 de la boucle
`for Variable in Ensemble de valeurs :` # *Variable* va prendre toutes les valeurs de « ensemble de valeurs »
 |Instructions # attention au respect de 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/chaînes:
`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 à l'avance le nombre de répétitions de la boucle
`while Condition :`
 |Instructions tant que « Condition » est vraie # attention au respect de l'indentation
 |Modifier la variable intervenant Condition # sinon la boucle serait infinie !
`break` # arrête la boucle avec une sortie « artificielle » de boucle :

OPÉRATEURS 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: ** marque l'exposant et a la priorité sur les opérateurs +, -, *, /

`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.)
`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(x)` # renvoie la valeur absolue d'un nombre de *x*

IMPORTATION DES MODULES

Déclaration :
`import math` # importe tout le module *math* et ses fonctions
`import random` # importe tout le module *random* et ses fonctions
`import numpy as np` # importe tout le module *numpy* sous l'alias *np* et ses fonctions (tableaux et fonctions)
`import matplotlib as plt` # importe tout le module *matplotlib* sous l'alias *plt* et ses fonctions (courbes)

MODULE MATH

Déclaration :
`import math` # importe tout le module *math* et ses fonctions
`from math import *` # importe tout le module *math* et ses fonctions

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

MODULE RANDOM

Déclaration :
`import random`

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

MODULE NUMPY

Déclaration :

```
import numpy as np # np est un « Alias » de numpy
```

Certaines fonctions sont présentes dans plusieurs modules (comme sinus ou cosinus qui existent dans les modules math et numpy). Pour préciser quel module on utilise, on met son nom (ou son alias) devant la fonction.

Il est préférable d'utiliser les tableaux Numpy si l'on souhaite ensuite tracer des courbes avec Matplotlib et créer des tableaux à partir d'autres tableaux. Par exemple : si « t » est un tableau, l'instruction : $x=2*t$ va créer un tableau « x » dont chaque élément vaut le double de ceux de t.

Fonctions :

```
T = np.array([]) # créé un tableau vide
T = np.array([valeur1, valeur2...]) # créé un tableau contenant valeur1, valeur2...
T = np.append(T, Elément) # ajoute Elément à la fin du tableau.
T = np.delete(T, index) # supprime l'élément situé à la position index (1er: 0 ; dernier : len(T)-1)
T = np.linspace(début, fin, n) # créé un tableau de n valeurs entre début et fin (inclus)
T = np.arange(début, borne2, pas) # créé un tableau d'entiers début et fin-1 par pas de pas
```

FONCTIONS

Définition d'une fonction SANS return (procédure)

```
def Ma_fonction():
    |Liste d'instructions # attention au respect de l'indentation
```

Appel de la fonction :

```
Ma_fonction() # à placer dans le programme principal pour exécuter le code associé
```

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() # affecte 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...) # affecte dans Variable le résultat des instructions de Ma_fonction
                                (ayant utilisé les paramètres d'entrée - arguments 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_Programme_Principal
```

FICHIERS : LECTURE/ÉCRITURE 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 :

```
Mon_fichier.close() # OBLIGATOIRE s'il a été ouvert !!
```

Écriture de données dans « Mon_Fichier » :

```
Mon_fichier.write(Texte) # il faut que Mon_fichier ait été ouvert (en écriture ou modification)
                          # écrit Texte à la suite du fichier ; pour ajouter un saut de ligne : Texte doit être :
                          # '\n' et 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() # affecte dans Variable tout le contenu du fichier
Variable = Mon_fichier.read(N) # affecte dans Variable les N caractères du fichier
Variable = Mon_fichier.readline() # affecte dans Variable une ligne du fichier
Liste = Mon_fichier.readlines() # affecte toutes les lignes du fichier dans une liste Liste.
```

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

```
import os
os.chdir(Dossier_Voulu) # Dossier_Voulu absolu: "C:/Mon/Dossier" ou relatif: "/Mon/Dossier"
```