OPERATEURS, incrémentation et Fonctions Mathématiques :

Simples: `+` `-` `*`

Pour la division: `/`

Nbre1 `//` *Nbre2*	Renvoie la partie entière de la division de *Nbre1* par *Nbre2* *(plutôt pour les entiers)*
Nbre1 `%` *Nbre2*	Renvoie le reste de la division de *Nbre1* par *Nbre2* *(plutôt pour les entiers)*
`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
`cmp(`*x*`,` *y*`)`	renvoie *-1* si *x* < *y*, *+1* si *x* > *y* et *0* si *x* == *y*
`abs()`	renvoie la valeur absolue d'un nombre (sans le signe)

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

Déclaration :

`from math import *` ne nécessite pas de rappeler la classe math → `pi` renvoie 3.14159…

ou

`import math` nécessite de rappeler la classe math → `math. pi` renvoie 3.14159…

Fonctions :

`pi` *ou* `math.pi`	retourne une approximation de la constante pi: 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 néprérien
`math.log10()`	logarithme décimal
`math.pow(`*x*`,`*y*`)`	renvoie *x* à la puissance *y*
`math.sqrt()`	renvoie la racine carrée
`math.hypot(`*x*`,`*y*`)`	renvoie l'hypothénuse pour un triangle rectangle de côtés *x* et *y*
`math.fabs()`	renvoie la valeur absolue (notation décimale)
`math.fmod(`*x*`,`*y*`)`	renvoie le reste (notation décimale) de la division de *x* par *y* *(plutôt pour les flottants)*
`math.modf()`	renvoie un tuple (partie décimale, partie entière) d'un réel (*flottant*)
`math.ceil()`	arrondissement par excès d'un nombre "réel"
`math.floor()`	arrondissement par défaut d'un nombre "réel"

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 général : `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*)

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

Entrée :

`Var = input('`*question*`')` Pose « *question* » à l'utilisateur. La réponse est mise dans Var , c'est une chaîne de caractère par défaut. Pour l'utiliser dans un calcul après, il faut ajouter int ou float : `Var = int(input('`*question*`'))`

Sortie écran :

`Print('`*Texte*`',`*variable*`)` écrit sur l'écran *Texte* suivi du contenu de *Variable* .

LISTES : Ce sont des tableaux.

Déclaration :

```
Ma_Liste = ['a','b','c']      (ou bien [nbre1, nbre2, nbre3…])
Ma_Liste = []                 Crée une liste vide
Ma_Liste[n] = 'a'             met 'a' à la position n (attention, la position commence à 0)
```

Accès aux éléments de la liste :

```
Ma_Liste[index]          Renvoie l'élément situé à l'emplacement index (qui commence à 0)
Ma_Liste[index1:index2]  Renvoie les éléments entre index1 et index2
Ma_Liste[index1:]        Renvoie les éléments à partir d'index1 jusqu'à la fin
Ma_Liste[:index2]        Renvoie les éléments du début jusqu'à index2
Ma_Liste[-1]             Renvoie le dernier élément (si -2 : l'avant dernier, etc…)
```

Manipulation de listes :

```
len(Ma_Liste)              Renvoie la longueur de ma liste
Ma_Liste.append(Elément)   Ajoute Elément à la fin de la liste
Ma_Liste.index(Elément)    Renvoie l'emplacement de Elément dans la liste.
Ma_Liste.count(Elément)    Renvoie le nombre de fois qu'Elément est présent dans la liste
Elément in Ma_Liste        Renvoie si Elément est présent dans la liste (True or False)
```

Parcourir une liste :

```
for Compteur in Ma_Liste:   Compteur prend successivement le contenu de chaque élément de la liste
                            (Si Ma_Liste contient des caractères, Compteur sera une chaine)
```

CHAINES : Ce sont des listes de caractères donc ont les mêmes fonctions + des fonctions supplémentaires :

Déclaration :

```
Mon_Texte = "Bonjour"
Mon_Texte = input('Entrer un texte') demande à l'utilisateur un texte qui sera mis dans Mon_Texte
Mon_Texte = "Bonjour \nSalut"                    Saute une ligne après le \n
Mon_Texte = "Bonjour \"Salut\""                  affiche des guillemets à la place de \"
Mon_Texte = "Bonjour {}, Salut {}".format(var1)  Met Var1 à la place des {}
Mon_Texte = "Bonjour {1}, Salut {0}".format(var1, var2)
                        Met Var1 à la place de {0} et Var2 à la place de {1}
Mon_Texte = "Bonjour {nom}, Salut {prenom}".format(nom=var1, prenom= var2)
Mon_Texte = str(Entier)                          transforme l'Entier en chaine
Mon_Texte = chr(Code_Ascii)   Renvoie le caractère (dont le code ascii est Code_Ascii)
Un_Entier = ord(Caractère)    Renvoie le code ascii de Caractère.
```

Manipulations fréquentes de chaines :

```
Mon_Texte.lower()          Met le texte en minuscule
Mon_Texte.upper()          Met le texte en majuscule
Mon_Texte.capitalize()     Met la 1ère lettre du texte en majuscule

Mon_Texte.split()          Renvoie une n^velle liste contenant les mots du texte (s'ils st séparés par « espace »)
Mon_Texte.split(Caractère) Renvoie une n^velle liste contenant les mots du texte (s'ils st séparés par Caractère)
Mon_Texte.find(Texte)      Renvoie le + petit index de Texte dans Mon_Texte (ou -1 si pas trouvé)
Mon_Texte.replace(TexteOld,TexteNew)  Remplace TexteOld par TexteNew dans Mon_Texte

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

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

TESTS et CONDITIONS :

Test simple:

```
if Condition:
        Instructions si « Condition » est vraie
```

bien penser au **:** !
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:

==	égal	**!=**	différent
> (ou **<**)	supérieur (ou inférieur)	**>=** (ou **<=**)	supérieur (ou inférieur) ou égal

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

BOUCLES :

Boucle FOR générale:

```
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 »**

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) :

```
while Condition:
        Instructions tant que « Condition » est vraie
        Modifier la variable intervenant dans la condition
```

bien penser aux « : » et à l'indentation
sinon la boucle serait infinie !

Sortie « artificielle » de boucle :

break arrête la boucle

MODULE RANDOM : hazard

Déclaration :

```
from random import * ou import random
```

Fonctions :

`choice(Ma_Liste)` ou `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)

STRUCTURE D'UN PROGRAMME

```
# partie commentaire qui décrit le programme
```

Partie `"Import"` pour importer des modules

Partie `"fonction":` on définit les fonctions que l'on va utilise avec "def" etc...

Déclaration des variables et leur initialisation

Programme principal: liste des instructions du programme

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

Définition d'une fonction SANS paramètres SANS retour de résultat (c'est une procédure) :

```
def Ma_fonction():
    Liste d'instructions
```

Attention à l'indentation

Appel de la fonction SANS paramètres :

```
Ma_fonction()
```

à placer dans le prg principal : Exécute le code de « Ma_Fonction »

Définition d'une fonction AVEC paramètres ET retour de résultat :

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

Rq : il y a 4 possibilités, on peut aussi faire une fonction sans paramètres qui renvoie un résultat et une procédure avec paramètres.

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 simples 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.chdir(Dossier_Voulu)
```
Dossier_Voulu peut être absolu: **"C:/Mon/Dossier"** ou relatif: **"/Mon/Dossier"**

```
Rq : os.getcwd()
```
renvoie le répertoire courant.

INTERFACE GRAPHIQUE avec TKINTER : une fenêtre et des WIDGETS : boutons, zone de texte etc…

Déclaration compléments : http://www.tutorialspoint.com/python/python_gui_programming.htm

```
from tkinter import *
```

Créer une fenêtre principale: (la fenêtre parent qui contiendra des widgets)

```
Fenetre = Tk()
```
déclare la fenêtre principale sous le nom "Fenetre"

```
Fenetre.title('Titre de la fenêtre')
```
donne un titre à la fenêtre

```
Fenetre.geometry('LargeurxHauteur+abscisse+ordonnée')
```
dimensions et position de la fenêtre

```
Fenetre.wm_attributes("-fullscreen","1")
```
affiche la fenêtre plein écran

```
Fenetre.wm_resizable(0,0)
```
empêche le redimensionnement de la fenêtre

```
w,h = Fenetre.winfo_screenwidth(), Fenetre.winfo_screenheight()
Fenetre.geometry("{}x{}+0+0".format(w, h))
```
pour maximiser une fenêtre

Afficher la fenêtre principale: **à placer après la déclaration des widgets ! (à la fin du programme)**

```
Fenetre.mainloop()
```

Fermer la fenêtre principale: **cela termine le programme**

```
Fenetre.destroy()
```
on peut aussi « destroy() » n'importe quel widget.

Méthodes applicable à tous les widget *: une fois le widget déclaré…*

```
Mon_widget.pack()
```
place le widget en haut à gauche de la fenêtre

```
Mon_widget.place(x=abscisse,y=ordonnée,width=largeur,height=hauteur)
```
Place le Widget à des coordonnées précises et fixe ses dimensions. (with, height optionnels).
coordonnées en pixels par rapport au haut-gauche de la fenêtre.

```
Mon_widget.config(Propriétés séparées par les virgules)
```
modifie les propriétés du widget (après sa création)

Widget LABEL: affiche un texte dans la fenêtre principale. Cette dernière doit exister !!

```
Mon_Label = Label(Fenetre,Propriétés séparées par les virgules)
```
déclare le Label

Propriétés utiles d'un Label :

```
text ="Texte du Label"
fg='couleur'
```
couleur du texte : red, blue, black, white,….

Autres propriétés: `font, cursor, underline, justify, height, width`…

Widget BUTTON: crée un bouton dans la fenêtre principale.

```
Mon_bouton = Button(Fenetre,Propriétés séparées par les virgules)
```
déclare le bouton

Propriétés utiles d'un « Button »:

```
text ="Texte du bouton"
command= Ma_fonction
```
fonction à exécuter quand on clique sur le bouton.

Autres propriétés: `font, fg, highlightcolor, underline, justify, height, width`…

Widget ENTRY: crée une zone de texte dans laquelle l'utilisateur peut écrire (à 1 seule ligne)

```
Mon_entry = Entry(Propriétés séparées par les virgules)
```
déclare la zone (les propriétés sont optionnelles)

Propriétés: `font, fg, highlightcolor, underline, justify, height, width`…

Méthodes utiles d'un « Entry »:

```
Mon_entry.get()
```
renvoie la chaine contenue dans l'Entry

```
Mon_entry.insert(index,Texte)
```
insert *Text* dans l'Entry à la position *index*.

Widget messagebox: permet d'afficher une boite de dialogue avec un message (et boutons oui/non/OK/cancel)

```
messagebox.showinfo(Titre,Message)
```
Affiche une boite avec OK, un *titre* et un *message*

```
messagebox.showwarning(Titre,Message)
```
Boite avec OK, un symbole attention, *titre* et *Message*

```
messagebox.showerror(Titre,Message)
```
Boite avec OK, un symbole erreur, *titre* et *Message*

```
messagebox.askyesno(Titre,Message)
```
Boite avec boutons Oui/Non. Renvoie une chaine avec «yes» ou «no»

```
messagebox.askokcancel(Titre,Message)
```
Boite OK/Cancel. Renvoie l'entier '1' pour OK ou '0' pour Cancel

Widget CANVAS: crée une zone de dessin

`Mon_canvas = Canvas(`*Propriétés séparées par les virgules*`)` déclare la zone de dessin

Propriétés: `bg,font, fg, cursor, height, width`…

Méthodes d'un « Canvas » créant des objets « dessin »: les items

`Mon_canvas.create_rectangle(`*x0,y0,x1,y1,propriétés*`)` Coord points haut gauche et bas droite

`Mon_canvas.create_line(`*x0,y0,x1,y1,…,xn,yn, propriétés*`)`

`Mon_canvas.create_polygon(`*x0,y0,x1,y1,…,xn,yn, propriétés*`)`

`Mon_canvas.create_oval(`*x0,y0,x1,y1, propriétés*`)` Coord du rectangle contenant l'oval

`Mon_canvas.create_arc(`*x0,y0,x1,y1,*`start=0, extend,` *propriétés*`)`

Coord du rectangle contenant l'oval, start=angle de départ, extend=angle d'arrivée

voir les autres sur http://effbot.org/tkinterbook/canvas.htm

Propriétés les plus fréquentes des items :

`fill="couleur"`	couleur du trait pour les line, de remplissage pour oval, rectangle,arc…
`dash=(x1,x2)`	trait en pointillé : trait de longueur x1, rien de longueur x2
`outline="couleur"`	couleur de trait extérieur pour rect, oval, arc, polygone…

voir les autres sur http://effbot.org/tkinterbook/canvas.htm

Modification d'un Canvas en cours de programme

Il faut que les objets « dessin » créés aient ***un nom*** *:*

`Rect=Mon_canvas.create_rectangle(`*x0,y0,x1,y1,*`fill="blue"…)`

On peut alors modifier le dessin :

`Mon_canvas.itemconfig(Rect,fill="red",`*autres ptés modifiées*`)`	Modifie les propriétés du dessin
`Mon_canvas.coords(Rect,`*newx0,newy0,newx1,newy1*`)`	Change ses coordonnées
`Mon_canvas.move(Rect,`*deplacementX,deplacementY*`)`	Déplace le dessin
`Mon_canvas.delete(Rect)`	Efface le dessin

Gestion précise d'un évènement sur un widget *(Clavier, clic droit ou gauche de la souris, double clic…)*
Cela concerne tous les widgets et la fenêtre principales. *A placer au moment de la création du Widget*

`Mon_widget.bind("<Button-1>",` *Fonction*`)`	Fonction est appelée si Clic gauche souris
`Mon_widget.bind("<Button-3>",` *Fonction*`)`	Fonction est appelée si Clic droit souris
`Mon_widget.bind("<B1-Motion>",` *Fonction*`)`	Fonction est appelée si clic gauche appuyé tout en déplaçant la souris
`Mon_widget.bind("<ButtonRelease-1>",` *Fonction*`)`	Fonction est appelée si Clic gauche lâché souris
`Mon_widget.bind("<Double-Button-1>",` *Fonction*`)`	Fonction est appelée si Double Clic gauche souris
`Mon_widget.bind("<Key>",` *Fonction*`)`	Fonction est appelée si On appuie sur une touche
`Mon_widget.bind("<Return>",` *Fonction*`)`	Fonction est appelée si On appuie sur ENTREE

Propriétés de l'évènement :

Dans la fonction appelée, on peut récupérer des informations sur l'évènement (coordonnées du clic, touche pressée etc…). Pour cela, mettre « `event` » en paramètre de la fonction quand on la déclare.

`def Fonction(event) :`

On peut alors utiliser : `event.x` et `event.y` qui renvoient les coordonnées du lieu cliqué
`event.char` qui renvoie le caractère appuyé
`event.num` qui renvoie le numéro d'évènement.

Voir complément sur http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

VERIFIER UN CODE AVANT DE L'EXECUTER :

try/except:

```
try :
        Bloc à essayer
except type_de_l_exception :
        Bloc qui sera exécuté en cas d'erreur
```

bien penser à l'indentation !!!

`type_de_l_exception` est le type d'erreur

le type d'erreur peut être *NameError, TypeError, ZeroDivisionError*, etc.

Voire la liste des exceptions sur fr.openclassrooms.com

Exemple :

```
try :
        resultat=numerateur / denominateur
except NameError:
        print("La variable numérateur ou dénominateur n'a pas été défini ")
except TypeError:
        print("La variable numérateur ou dénominateur possède un type incompatible avec la division ")
except ZeroDivisionError :
        print("La variable dénominateur est égale à 0. ")
```

Ceci permet en particulier de vérifier les entrées d'un input en combinant avec un while

```
while True:
        try:
                nb=int(input("Entrez un entier :"))
                break
        except ValueError:
                print("Entrez un nombre.")
```

FAIRE COMMUNIQUER DES ORDINATEURS :

try/except:

```
try :
        Bloc à essayer
except type_de_l_exception :
        Bloc qui sera exécuté en cas d'erreur
```

bien penser à l'indentation !!!

`type_de_l_exception` est le type d'erreur

le type d'erreur peut être *NameError, TypeError, ZeroDivisionError*, etc. Exemple :

```
try :
        resultat=numerateur / denominateur
except NameError:
        print("La variable numérateur ou dénominateur n'a pas été défini ")
except TypeError:
        print("La variable numérateur ou dénominateur possède un type incompatible avec la division ")
except ZeroDivisionError :
        print("La variable dénominateur est égale à 0. ")
```

Ceci permet en particulier de vérifier les entrées d'un input en combinant avec un while

```
while True:
        try:
                nb=int(input("Entrez un entier :"))
                break
        except ValueError:
                print("Entrez un nombre.")
```