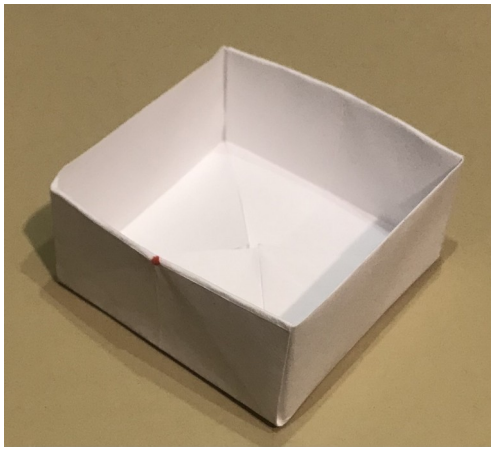
Objectif :



Finalité : étudier le principe de passage d'informations depuis la partie principale d'un script vers une fonction suite au découpage d'un algorithme en fonctions.

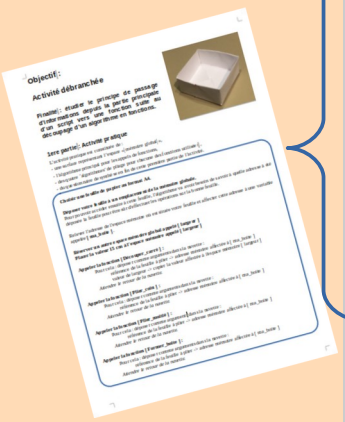
1ere partie : Activité pratique

L'activité pratique est constituée de :
- une surface au centre représentant l'espace mémoire alloué au processus correspondant au script Python,
- l'algorithme principal pour les appels de fonctions,
- des fiches navettes pour expliciter les passages d'arguments lors des appels de fonctions,
- des quatre 'algorithmes' de pliage pour chacune des fonctions utilisées ,
- du questionnaire de synthèse en fin de cette première partie de la séance.

Disposition de l'activité :

Espaces Mémoire où chaque information est stockée à une adresse **référencée par une variable**.

Espace pour le **script principal**



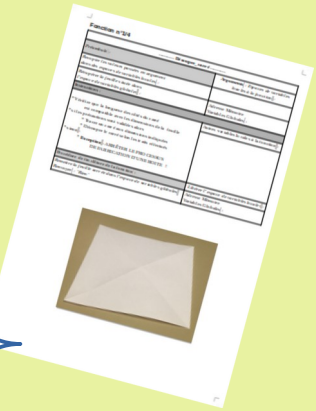
Espace mémoire du script principal (variable globale)

Adresse	*Référence*	*Information*
0000	**ma_boite**	
0001	**largeur**	15
0002		
0003		

Espace mémoire mis à disposition d'une fonction pendant son temps d'exécution (variable locale)

Adresse	*Référence*	*Information*
0010	***largeur_carre***	**15**
0011	***Feuille***	*Adresse 0000*

Espace réservé **pour les scripts des fonctions**

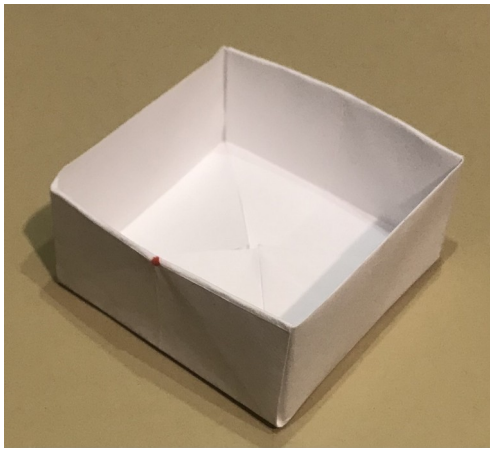


Fiche de **questions de synthèse** et compléments d'informations.

2eme partie : Analyse de scripts Python sur les fonctions

Scripts Python où la partie principale du script effectue des appels à des fonctions avec :
- des informations passées **par valeurs** *(int, float, tuple, str, ...)*
- des informations passées **par référence** *(list, dict, ...)*

1ere partie :



<u>Objectif</u> : étudier le principe de passage d'informations depuis la partie principale d'un script vers une fonction et réciproquement.

L'activité pratique est constituée de :
- **une feuille A3** : espace mémoire alloué au processus pour le script principal *(espace mémoire global)*
- **une feuille A4 plastifiée** : espace mémoire alloué à une fonction en cours d'exécution *(feuille à compléter avec un feutre effaçable)*,
- l'**algorithme principal** pour les appels de fonctions,
- les **quatre algorithmes des fonctions de pliage** et **une fiche navette plastifiée par fonction**,
- le questionnaire de synthèse en fin de cette première partie de l'activité.

Réserver un espace de la mémoire globale référencé par le nom Continuer et lui attribuer l'info. True

Tant que Continuer "*contient*" True **faire :**

Choisir une feuille de papier au format A4.

Déposer votre feuille à un emplacement de la mémoire du script principal *(mémoire globale).*
*Pour pouvoir accéder à cette feuille, l'algorithme va ensuite utiliser l'**adresse mémoire** où a été déposée la feuille, ceci pour être sûr d'effectuer les opérations sur la bonne feuille.*

Référencer l'adresse mémoire où est située votre feuille par une variable appelée ma_boite .

Réserver un autre espace de la mémoire global référencé par le nom : largeur
Attribuer l'information 15 (cm) à l'espace mémoire référencé par la variable largeur

Appeler la fonction Découper_carré **(** *arguments* **) :**
Compléter la fiche navette avec les arguments :
adresse de la feuille à plier -> adresse mémoire référencé par [ma_boite]
largeur du carré -> 15
Attendre le retour de la fiche navette.

Appeler la fonction Plier_coin **(** *arguments* **) :**
Compléter la fiche navette avec les arguments :
adresse de la feuille à plier -> adresse mémoire référencé par [ma_boite]
Attendre le retour de la fiche navette.

Appeler la fonction Plier_moitié **(** *arguments* **) :**
Compléter la fiche navette avec les arguments :
adresse de la feuille à plier -> adresse mémoire référencé par [ma_boite]
Attendre le retour de la fiche navette.

Appeler la fonction Former_boite **(** *arguments* **) :**
Pour cela : déposer comme arguments dans la navette :
référence de la feuille à plier -> adresse mémoire référencé par [ma_boite]
texte à écrire : "NSI"
Attendre le retour de la navette.
Affecter la valeur de retour à l'espace mémoire référencé par la variable Continuer

Fonction n°1/4

Découper_carré	Variables et infos reçues	largeur_carre *int*	Feuille *object*	→ type de l'info. renvoyée	None

Information stockée dans l'espace local ou global ?	***informations passées en argument :***
Arguments reçus par la fonction : *- si l'argument de la fonction est de type* int, str ou tuple *alors* ***l'information est recopiée dans l'espace mémoire local** à la fonction :* ***passage par valeur** → pas de modif. de l'information présente dans l'espace global.* *- si l'argument de la fonction est de type* list, dict *ou object alors, c'est* ***l'adresse de l'espace mémoire global qui est recopié dans l'espace mémoire local** à la fonction → l'information stockée dans l'espace mémoire global peut être modifiée par les instructions de la fonction !*	largeur_carre → . . . Feuille → . . .

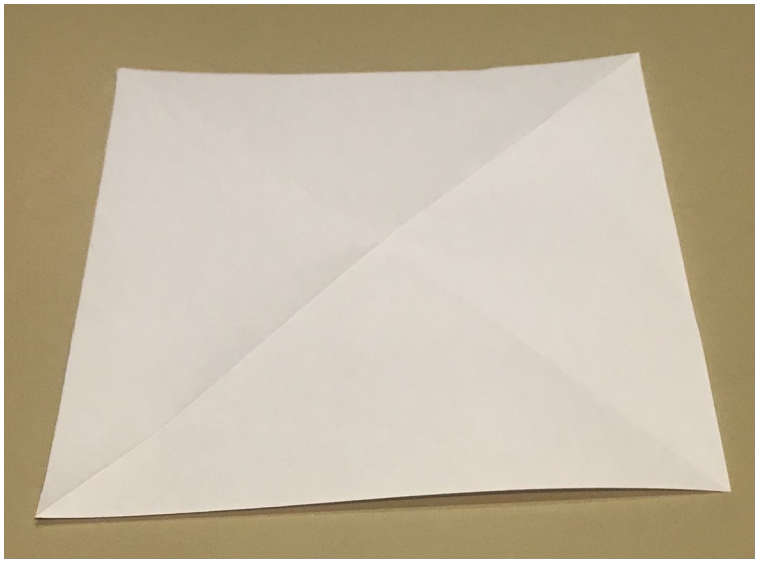
Instructions :

* Vérifier que la valeur affectée à la variable largeur_carre permet effectivement de tracer un carré de côté largeur_carre cm sur la feuille présente à l'adresse mémoire Feuille.

* si les précautions sont validées alors
    * Tracer un carré de largeur_carre cm sur la feuille présente à l'adresse mémoire Feuille
    * Découper la feuille à l'adresse mémoire Feuille selon les traits effectués
* sinon :
    * **Exception** : ARRÊTER LE PROCESSUS PRINCIPAL DE FABRICATION D'UNE BOITE !

Procédure de fin → clôture de la fonction :

La fonction ne renvoie pas d' info. donc inscrire None comme valeur de retour. Libérer l'espace mémoire local alloué à la fonction *(effacer l'espace mémoire local)* Renvoyer la fiche navette au script principal	Valeur de retour : → . . .



Fonction n°2/4

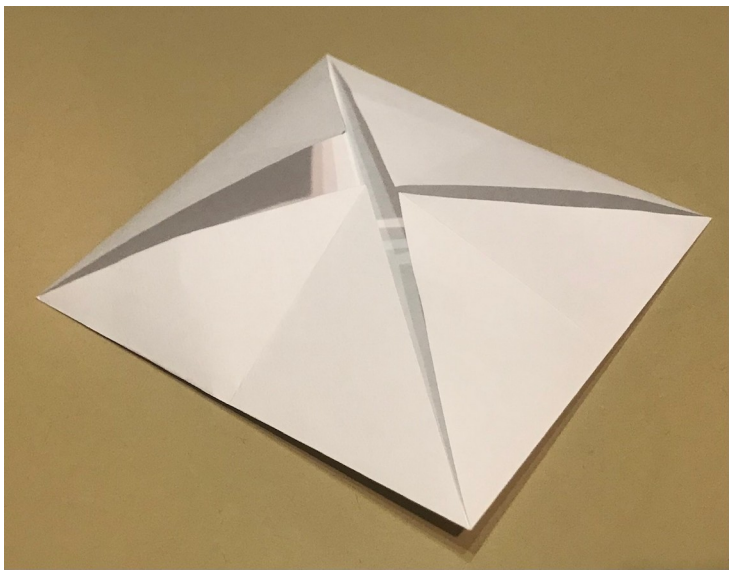
Plier_coins	*Variable et info reçue*	Feuille *object*	→ *type de l'info. renvoyée*	None

Information stockée dans l'espace local ou global ?	***informations passées en argument :***
Arguments reçus par la fonction : *- si l'argument de la fonction est de type* int, str ou tuple *alors* ***l'information est recopiée dans l'espace mémoire local** à la fonction :* ***passage par valeur*** *→ pas de modif. de l'information présente dans l'espace global.* *- si l'argument de la fonction est de type* list, dict *ou object alors, c'est* ***l'adresse de l'espace mémoire global qui est recopié dans l'espace mémoire local** à la fonction → l'information stockée dans l'espace mémoire global peut être modifiée par les instructions de la fonction !*	Feuille → . . .

Instructions :

* **Faire quatre fois** : => *Utiliser une variable* i *locale pour compter les itérations : for i in range (4) :*
- * Replier un coin de Feuille vers le centre du carré
- * Tourner Feuille d'un quart de tour

Procédure de fin → clôture de la fonction :	
La fonction ne renvoie pas d' info. donc inscrire None comme valeur de retour. Libérer l'espace mémoire local alloué à la fonction *(effacer l'espace mémoire local)* Renvoyer la fiche navette au script principal	Valeur de retour : → . . .



Fonction n°3/4

Plier_moitié	*Variable et info reçue*	*Feuille* — *object*	→ *type de l'info. renvoyée*	None

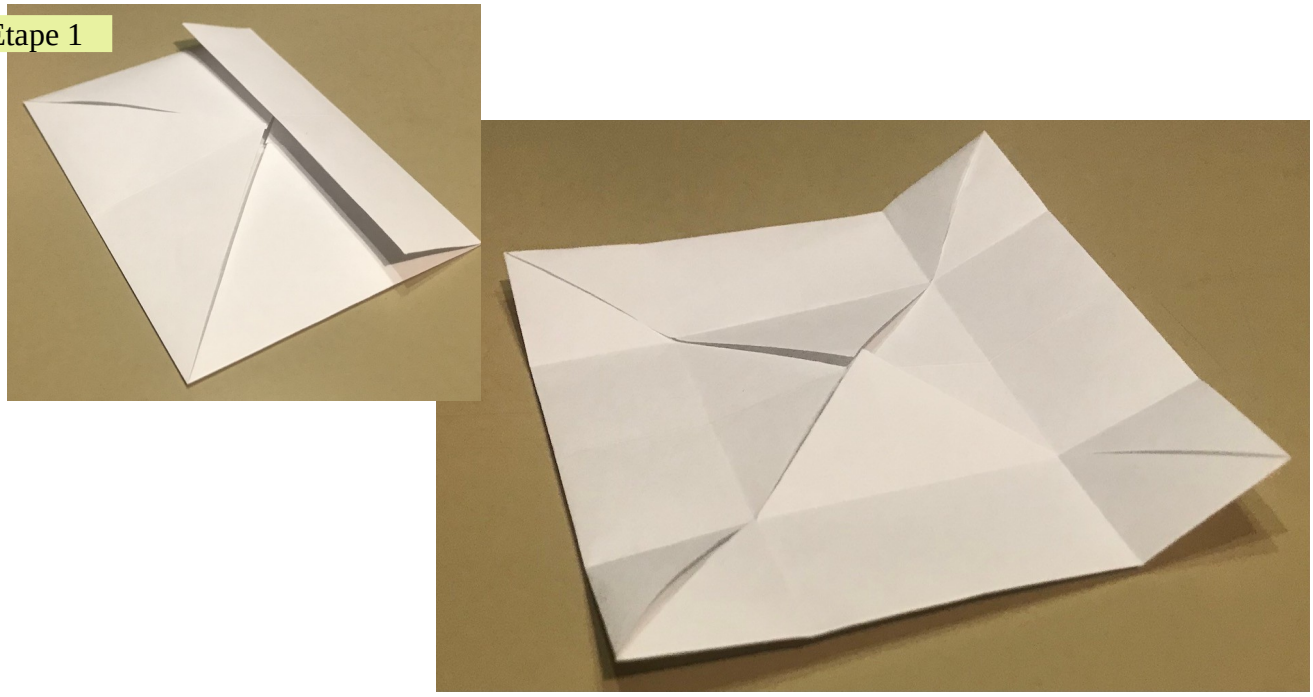
Information stockée dans l'espace local ou global ?	***informations passées en argument :***
Arguments reçus par la fonction : *- si l'argument de la fonction est de type* int, str ou tuple *alors* ***l'information est recopiée dans l'espace mémoire local** à la fonction :* ***passage par valeur** → pas de modif. de l'information présente dans l'espace global.* *- si l'argument de la fonction est de type* list, dict *ou object alors, c'est* ***l'adresse de l'espace mémoire global qui est recopié dans l'espace mémoire local** à la fonction → l'information stockée dans l'espace mémoire global peut être modifiée par les instructions de la fonction !*	*Feuille* → . . .

Instructions :

* **Faire quatre fois** : *(Utiliser une variable locale pour compter)*
- * Rabattre un côté de Feuille vers le centre du carré *(voir Etape 1 ci-dessous)*
- * Marquer le plis avec un objet rigide
- * Replacer le côté plié de Feuille à sa position initiale
- * Tourner Feuille d'un quart de tour

Procédure de fin → clôture de la fonction :	
La fonction ne renvoie pas d' info. donc inscrire None comme valeur de retour. Libérer l'espace mémoire local alloué à la fonction *(effacer l'espace mémoire local)* Renvoyer la fiche navette au script principal	Valeur de retour : → . . .

Etape 1



Fonction n°4/4

Former_boite	Variables et infos reçues	*texte* *str*	*Feuille* *object*	→ *type de l'info. renvoyée*	bool

Information stockée dans l'espace local ou global ?	***informations passées en argument :***
Arguments reçus par la fonction : *- si l'argument de la fonction est de type* int, str ou tuple *alors* ***l'information est recopiée dans l'espace mémoire local** à la fonction :* ***passage par valeur** → pas de modif. de l'information présente dans l'espace global.* *- si l'argument de la fonction est de type* list, dict *ou object alors, c'est* ***l'adresse de l'espace mémoire global qui est recopié dans l'espace mémoire local** à la fonction → l'information stockée dans l'espace mémoire global peut être modifiée par les instructions de la fonction !*	*texte* → . . . *Feuille* → . . .

Instructions :

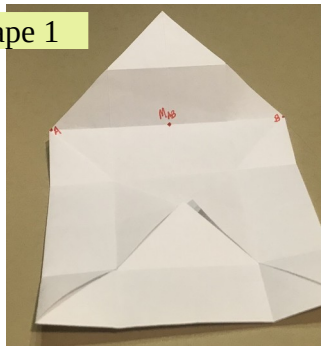
* **Faire deux fois** : *(Utiliser une variable locale pour compter le nombre d'itérations)*
- * Déplier (ouvrir) un des coins replié de Feuille
- * Marquer les points A, M_{AB}, B sur le coin déplié de Feuille *(voir photo Etape 1)*
- * Replier le coin repéré par A vers le point repéré par M_{AB} *(voir photo Etape 2)*
- * Replier le coin repéré par B vers le point repéré par M_{AB} *(voir photo Etape 3)*
- * Rabattre le coin déplié à l'intérieur de la boite selon l'axe A-M_{AB}-B pour former un côté
- * Tourner Feuille d'un demi tour pour effectuer la même opération sur le coin opposé

* Inscrire le texte référencé par la variable texte sur une des faces extérieure de Feuille.

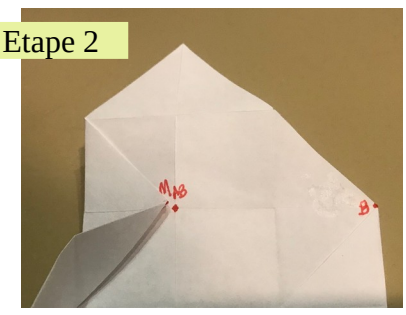
* Si la boîte est correctement formée, renvoyer l'information True , sinon renvoyer False.

Procédure de fin → clôture de la fonction :	
Inscrire l'information à renvoyer au script principal ----------▶ Libérer l'espace mémoire local alloué à la fonction *(effacer l'espace mémoire local)* Renvoyer la fiche navette au script principal	Valeur de retour : → . . .

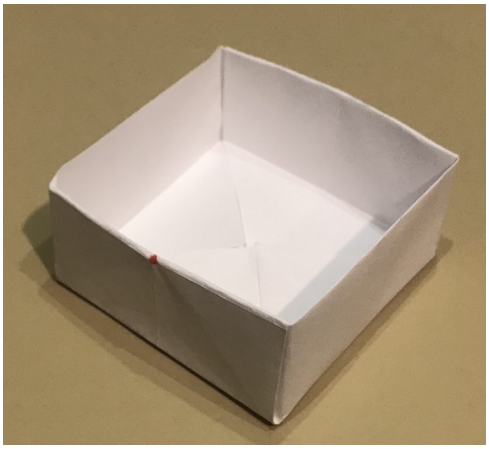
Etape 1



Etape 2



Etape 3



APPEL de la fonction : **Découper_carré** (*largeur_carre : int , Feuille : object*) *-> None* :		
Arguments passés à la fonction :		
Nom de la variable qui reçoit l'information :	largeur_carre	*Feuille*
Info. passée par valeur *recopier l'info. ->* / Référence *adresse de l'info. ->*		
Information renvoyée par la fonction :		
Instruction **return … ->**		

Exemple :

APPEL de la fonction : ***Découper_carré*** (*largeur_carre : int, Adresse_Feuille : objet*) → *None :*			
Arguments passés à la fonction :			
Nom de la variable qui reçoit l'information ->		largeur_carre	*Feuille*
Info. passée par	valeur *recopier l'info. ->*	4	*Adresse 0x0000*
	Référence *adresse de l'info. ->*		
Information renvoyée par la fonction -> . . .			
Instruction **return …**			*None*

APPEL de la fonction : **Plier_coins** (*Feuille : object*) *-> None* :		
Arguments passés à la fonction :		
Nom de la variable qui reçoit l'information :		*Feuille*
Info. passée par valeur *recopier l'info. ->* / Référence *adresse de l'info. ->*		
Information renvoyée par la fonction :		
Instruction **return … ->**		

APPEL de la fonction : **Plier_moitié** (*Feuille : object*) *-> None* :		
Arguments passés à la fonction :		
Nom de la variable qui reçoit l'information :		*Feuille*
Info. passée par valeur *recopier l'info. ->* / Référence *adresse de l'info. ->*		
Information renvoyée par la fonction :		
Instruction **return … ->**		

APPEL de la fonction : **Former_boite** (*texte : str , Feuille : object*) *-> None* :		
Arguments passés à la fonction :		
Nom de la variable qui reçoit l'information :	texte	*Feuille*
Info. passée par valeur *recopier l'info. ->* / Référence *adresse de l'info. ->*		
Information renvoyée par la fonction :		
Instruction **return … ->**		

Questions de synthèse de l'activité et compléments d'informations

Quelles différences faites vous entre une variable utilisée dans la partie principale du script et une variable locale à une fonction ?

Réponse attendue:

- localisation dans l'espace mémoire,
- « durée de vie » de la variable,
- accessibilité par les instructions *(de la fonction # du script principal)*, …

IMPORTANT : A propos du « nom de la variable », l'activité n'aborde pas le problème de la confusion engendrée dans le langage Python par des variables du script principal accessibles depuis une fonction sans utiliser le mot clé global. *(voir le script joint local_global_implicite.py)*

- soit il n'y a pas d'affectation et la variable est accessible en lecture seule *(équivalent à un passage par valeur)*
- soit il y a affectation et la fonction utilise une variable locale qui porte le même nom que la variable du script principal mais qui n'est pas dans le même espace mémoire ! *(variable locale)* ***A CONDITION*** *qu'aucune instruction de la fonction n'ait utilisé accéder à cette variable en lecture avant affectation (voir f2( ) du script cité ci-dessus)*

Ce point pourrait faire l'objet d'une autre fiche navette dans une autre activité d'approfondissement ou de pédagogique individualisée.

local_global_implicite.py

```
1  def f1 ( )->None:
2      """ utilisation implicité de arg comme
3      variable global MAIS en lecture seule ! """
4      if arg != 0 : print("\tf1 -> ",arg)
5
6  def f2 ( arg:int )->None:
7      if arg != 0 : arg = 20 # variable locale
8      print("\tf2 -> ",arg)
9
10
11 def f3 ( )->None:
12     global arg
13     if arg != 0 :arg = 30 # variable globale
14     print("\tf3 -> ",arg)
```

local_global_implicite.py

```
16 arg = 1
17 print("=== Appel de f1() ..." )
18 f1()
19
20 print("=== Appel de f2( ) ...")
21 f2(arg)
22 print(" // arg après f2 :", arg)
23
24 print("=== Appel de f3( ) ...")
25 f3()
26 print(" // arg après f3 :", arg)
```

Console

```
>>> %Run local_global_implicite.py
 === Appel de f1() ...
         f1 ->  1
 === Appel de f2( ) ...
         f2 ->  20
  // arg après f2 : 1
 === Appel de f3( ) ...
         f3 ->  30
  // arg après f3 : 30
```

Quelle est la signification « durée de vie » d'une variable d'un point de vue de la mémoire allouée ?

Réponse attendue :

Si cette variable est déclarée dans le script principale elle sera disponible jusqu'à la fin du script principal. Si cette variable a été déclarée dans une fonction, l'espace mémoire est libéré à la fin de l'exécution du script de la fonction.

Une variable alpha « existe » dès la première instruction alpha = … . Cette instruction signifie que l'interpréteur Python va utiliser la référence alpha et l'associer à une information en mémoire.

Soit cette information **est déjà** en mémoire, alors il n'y a pas d'utilisation supplémentaire de mémoire, alpha fera référence à l'information déjà disponible. *Il peut donc y avoir plusieurs variables qui référencent la même information en mémoire.*

Soit cette information **n'est pas** en mémoire et l'interpréteur va alors utiliser un nouvel espace mémoire pour y placer l'information puis référencer cet espace avec le nom de la variable : alpha.

```
>>> a = "maison"
>>> b = "maison"
>>> id(a)
140063862049648
>>> id(b)
140063862049648
```

Nous n'aborderons pas ici les instructions qui permettent de libérer des espaces mémoire alloués. Cette opération est automatisé sous Python : garbage collector => http://www.mypycode.com/fr/Python/1001000649.html

Là aussi, le type de l'information et donc le type de la variable va impliquer un mode de fonctionnement différent que l'on retrouve selon que l'affectation se fasse par valeur (int, str, …) ou par référence (list, dict, …) voire en fonction de l'information en elle même.

En effet, Python stocke par défaut les valeurs de -5 à 256 car ces valeurs sont très régulièrement utilisées. Donc nul besoin de réserver de l'espace mémoire, il suffit d'y faire référence.

Console

```
>>> from ctypes import cast
>>> id(2)
140470175924496
>>> for adr in range( id(0), id(0)+4*0x20, 0x20 ):
        print(f"Adresse : {adr} information : {ctypes.cast( adr, ctypes.py_object).value}")

 Adresse : 140470175924432 information : 0
 Adresse : 140470175924464 information : 1
 Adresse : 140470175924496 information : 2
 Adresse : 140470175924528 information : 3
```

L'incrémentation par pas de 0x20 octets correspond à 32 octets par valeur !

A titre informatif : le nom d'une liste ne fait pas référence à l'adresse où est stocké l'élément d'indice 0 mais fait référence à l'adresse mémoire où est placée l'instance de class list avec tous les attributs propres à cette instance de classe (longueur, …). Le premier élément de la liste est placé ailleurs en mémoire. Tout dépend de sa nature int, str, tuple, ...

memoire_liste.py

```
""" Le type list : par référence ou par valeurs ?
-> copie de la référence ou des éléments de la liste """

l0 = [ "l0", 1, 2, 3 ]
print(f"Adr l0 -> 0x{id(l0):x} | l0 : {l0}")

l1 = l0 # l0 et l1 vont référer le même espace mémoire
print(f"Adr l1 -> 0x{id(l1):x} | l1 : {l1}")

l1[0] = "l1" # modifier l1 implique une modification de l0
print('\nAprès modification de l1[0] = "l1" ')
print(f"\t l0 est aussi modifiée  : {l0}")

l2 = l0[:] # recopier l'info. sur un autre espace mémoire
print(f"\nAdr l2 -> 0x{id(l2):x} | l2 : {l2}")

l2[0] = "l2" # l2 est à un autre espace mémoire que l0
print(f"\t l0 inchangée : {l0}")
print(f"\t l2 modifiée  : {l2}")
```

Console

```
>>> %Run memoire_liste.py
Adr l0 -> 0x7f6339dfe8c0 | l0 : ['l0', 1, 2, 3]
Adr l1 -> 0x7f6339dfe8c0 | l1 : ['l0', 1, 2, 3]

Après modification de l1[0] = "l1"
         l0 est aussi modifiée  : ['l1', 1, 2, 3]

Adr l2 -> 0x7f6339ddb700 | l2 : ['l1', 1, 2, 3]
         l0 inchangée : ['l1', 1, 2, 3]
         l2 modifiée  : ['l2', 1, 2, 3]
```

Peut-on accéder aux variables locales à une fonction depuis la partie principale du script ?

Réponse attendue :

Par principe, le script principal est suspendu, pendant le temps d'exécution de la fonction. celui-ci attend le retour de la « fiche navette ».

Le script principal ne peut accéder qu'à l'information renvoyée par l'instruction return …

Soit cette instruction transmet une valeur, soit une référence à une adresse où est placée l'information renvoyée par la fonction mais elle ne peut être récupérée que si celle-ci est référencée par une variable au retour de la fonction : ma_variable = fonction (…).

Peut-on modifier l'information référencée par une variable du script principal depuis une fonction ?

Réponse attendue :

Oui, depuis une fonction on peut modifier l'information référencée par une variable du script principal à condition que le mot clé global soit utilisé dans la fonction et que les variables portent le même nom dans le script principal et le script de la fonction puisqu'il s'agit alors de la même variable *(référence)*.

Lorsqu'on passe une information référencée par une variable dans la script principal, à une variable déclarée dans les arguments de la fonction, ces deux variables doivent-elles avoir le même nom ?

Exemple : ici peut-on utiliser la variable rayon dans le script principal et r dans le script de la fonction ?

volume_cylindre.py

```
def volume_cylindre ( r:int, h:int )->int:
    """ Calcule le volume d'un cylindre de
    rayon r et de hauteur h """
    return 3.14 * r**2 * h

rayon = 5
hauteur = 2
volume = volume_cylindre( rayon, hauteur )
```

Réponse attendue :

Non les variables peuvent avoir des noms différents puisque les deux espaces mémoire sont différents.

En revanche **l'ordre des arguments est important** si le nom de la variable local n'est pas cité dans l'appel.

Ici, l'appel volume_cylindre (5, 2) aurait le même effet que volume_cylindre (h = 2, r = 5)

Réaliser la fiche navette qui correspondrait à l'appel de la fonction surface_carre (…) ci dessus.

APPEL de la fonction : **surface_carre (** *r : int , h : int* **)** *-> int* **:**		
Arguments passés à la fonction :		
Nom de la variable qui reçoit l'information :	r	h
Info. passée par valeur *recopier l'info. ->* / Référence *adresse de l'info. ->*		
Information renvoyée par la fonction :		
Instruction **return … ->**		

2eme partie :

Analyse de scripts Python

Objectifs :

- Lors de l'appel d'une fonction, comprendre les transferts d'informations vers cette fonction (passage d'arguments) et la récupération de l'information renvoyée par cette fonction en fin de traitement (return …).

- Faire la distinction entre une variable locale à une fonction et une variable globale déclarée dans le script principal.

L'activité pratique est constituée de :
- trois scripts Python contenant des déclarations de fonctions ainsi que les informations affichées en console,

Cette partie ne demande qu'à être développée avec différents exercices de niveau de difficulté variés.

Elle peut faire l'objet d'un travail individuel puis une autocorrection en binômes pour échanger sur les résultats obtenus.

Fonction n°1

```
1 from random import randint
2 # ================================== trouver_val
3 def trouver_val ( val:int, n:int )->int :
4     fin = False
5     essais = 0
6     print(f"Vous pouvez faire {n} essais.")
7     while fin == False :
8         essais = essais + 1
9         nombre = int( input("Entrez un nombre entier positif->") )
10        if nombre < val :
11            print("Trop petit")
12        elif nombre > val :
13            print("Trop grand")
14        else :
15            print("Bravo !")
16            fin = True
17
18        if essais >= n :
19            print("Perdu !")
20             fin = True
21
22     return essais
23 # ================================ main
24 x = randint( 0, 15 ) # Déterminer un nombre aléatoire [ 0 ; 15 ]
25 n = 5  # Proposer 5 essais
26
27 nb_essais = trouver_val( x, n )
28
29 print(f"Vous avez fait {nb_essais}.")
```

nom de la variable	x	n *(ligne 25)*	nb_essais	val	nombre	n *(ligne 3)*	essais	fin
information référencée								*True ou False*
type de l'information								*booléen*
emplacement mémoire global ou local ?								*local*
Si la variable est un argument de la fonction l'information est-elle passée par valeur ou par référence ?								- - -

Attribuer un emplacement mémoire dans l'ordre de déclaration des variables :

Espace mémoire global du script principal		
Adresse	***Référence***	***Information***
0000		
0020		
0040		
0060		
0080		

Espace mémoire local à la fonction		
Adresse	***Référence***	***Information***
1000		
1020		
1040	*fin*	*True ou False*
1060		
1080		

Fonction n°2

```
1  # ================================ liste_val
2  def liste_val ( lst : list )->None:
3      for val in lst :
4          if val == 0 :
5              val = "zero"
6
7  # ================================ liste_ref
8  def liste_ref ( lst : list )->None:
9      n = len( lst )
10     for i in range(n):
11         if lst[i] == 0 :
12             lst[i] = "zero"
13
14 # ================================ main
15 liste = [ 0, 1, 2, 0, 3, 4 ]
16 print(f"Déclaration de liste : {liste}")
17
18 liste_val( liste )
19 print (f"Après liste_val( liste ) : {liste}")
20
21 liste_ref( liste )
22 print (f"Après liste_ref( liste ) : {liste}")
```

Variables de la mémoire locales :		
Variable	lst	val
Référence ou Valeur		
Information		

Variables de la mémoire locale :			
Variable	lst	i	n
Référence ou Valeur			
Information			

Variables de la mémoire globale :						
Variable	liste					
Adresse	*00*	*02*	*04*	*06*	*08*	*10*
Information	0	1	2	0	3	4

Informations affichées en console lors de l'exécution du script :

```
>>> %Run fonction_2.py
Déclaration de liste : [0, 1, 2, 0, 3, 4]
Après liste_val( liste ) : [0, 1, 2, 0, 3, 4]
Après liste_ref( liste ) : ['zero', 1, 2, 'zero', 3, 4]
```

1 - D'après les scripts des fonctions liste_val(...) et liste_ref(...) ainsi que les sorties en console suite aux print(...) du script principal, **quelles sont les conséquences de la notation utilisée** pour accéder aux informations affectées aux éléments de la liste liste [ ] ?

Accès aux informations :	Instructions en langage Python :	Conséquence(s) sur les informations stockées en mémoire :
Parcourir la liste par valeurs :	for val in liste : val = "zero"	
Parcourir la liste par références :	for i in range (len(liste)) : liste [i] = "zero"	

2 – Dans les arguements de la fonction liste_val ( ) ci-contre, la variable lst sera-elle affectée à une liste par valeur ou par référence ?	1 def liste_val (lst : list)->None: 2 for val in lst : 3 if val == 0 : 4 val = "zero"

Fonction n°3

```
1  # ============================ ecart
2  def ecart ( vals : list )->int :
3      mini = vals[0]
4      maxi = vals[0]
5      for v in vals :
6          if v < mini :
7              mini = v
8          if maxi < v :
9              maxi = v
10     return maxi - mini
11
12  # ============================ main
13
14 liste = [ 2, 15, 18, 1 ]
15
16 v = ecart ( liste )
17
18 print(f"Ecart entre mini et maxi de l : {v}")
```

Indiquez les informations contenues dans les mémoires globale et locale lorsque l'interpréteur a terminé d'exécuter l'instruction à la ligne 5 et que la valeur 18 a été affectée à la variable v.

nom de la variable	liste	v *(ligne 16)*	vals	mini	maxi	v *(ligne 5)*
information référencée						
type de l'information						
emplacement mémoire global ou local ?						
Si la variable est un argument de la fonction l'information est-elle passée par valeur ou par référence ?						

Note : les valeurs des adresses mémoire ne sont données qu'à titre indicatif pour comprendre le mécanisme de mémorisation des informations référencées par des variables. Attribuer un emplacement mémoire dans l'ordre de déclaration des variables :

Espace mémoire global du script principal					
Adresse	***Référence***	***Information***			
0000	liste	*0000*	*0002*	*0004*	*0006*
		2	15	18	1
0020					
0040					

Espace mémoire local à la fonction		
Adresse	***Référence***	***Information***
1000		
1020		
1040		
1060		

Réaliser la fiche navette au moment du retour de la fonction ecart (…) ci dessus.

APPEL de la fonction : **ecart (** *vals : list* **)** *-> int* **:**	
Arguments passés à la fonction :	
Nom de la variable qui reçoit l'information :	
Information passée par : valeur *recopier l'info. ->* / Référence *adresse de l'info. ->*	
Information renvoyée par la fonction :	
Instruction **return … ->**	