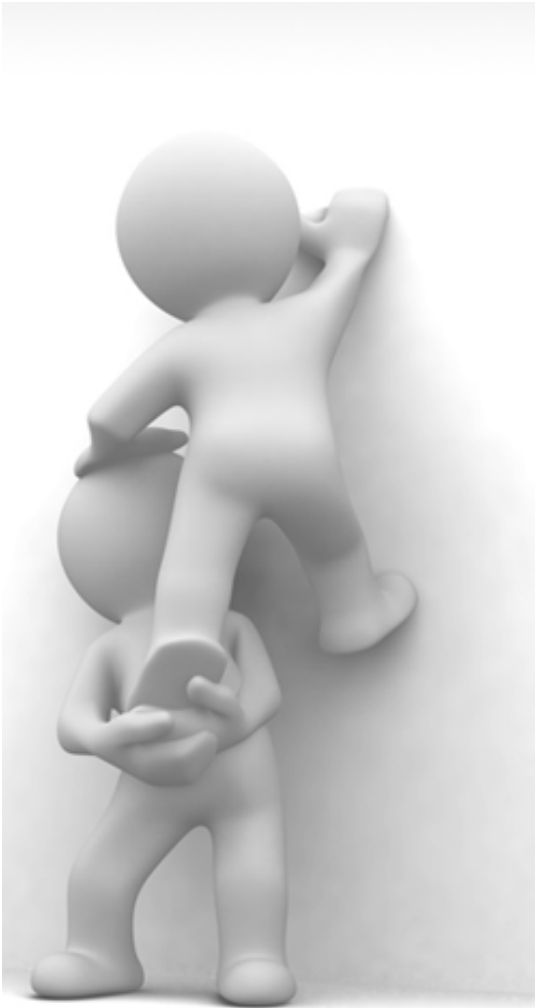


L3 Informatique - 2020-2021

Atelier 2 – Fonctions et paramètres



Objectifs de ce cours



- Vous donner les connaissances minimales pour réaliser les exercices de l'atelier 2
- Maîtriser la notion de fonction paramétrée :
 - Savoir les définir
 - Savoir les appeler
- Comprendre les notions de variables et les références en python

Notions abordées dans ce cours

- Notion de sous-programme
- Fonction sans résultat (procédure)
- Fonction paramétrée
- Fonction renvoyant un résultat
- Portée des variables
- Notion d'objet et de référence (id)
- Objets mutables et immutables
- Principe de transmission des paramètres





Notion de sous-programme

Procédures et fonctions

Notion de sous-programme

- Un sous-programme est un programme dont l'exécution est déclenchée par un autre programme.
- On dit alors que le Programme appelle le Sous-programme.
- On distingue deux catégories de sous-programmes :
 - Les procédures : fonction renvoyant None en python (fonctions sans résultat)
 - Les fonctions avec résultat

def **saisie**():
 """procédure de saisie des notes"""
 algorithme

APPEL

```
def gestion_etudiants():  
    """Procédure principale de gestion des étudiants"""  
    print( "Programme de gestion des etudiants ")  
    saisie()  
    calcul()  
    print( moyenne() )
```

def **calcul**():
 """procédure de calcul des moyennes des étudiants..."""
 algorithme

def **moyenne**() :
 """fonction de calcul de la moyenne générale."""
 algorithme
 return resultat

En python, tout est fonction!

Intérêt des sous-programmes

- Structuration, Lisibilité, Facilité de maintenance
- Partage des tâches de programmation
- Factorisation, Réutilisation
- Les sous-programmes permettent d'éviter des répétitions inutiles.

Si un même traitement doit être exécuté plusieurs fois au cours du déroulement d'un programme , le même sous-programme sera appelé plusieurs fois.

Cet intérêt est d'autant plus explicite avec
l'utilisation de paramètres



Procédures

Fonction renvoyant None
en python

Fonction sans résultat et sans paramètre

variable locale à la fonction date_rentree_L3

```
def date_rentree_L3():  
    """Fonction d'affichage de la date de début des cours"""  
    # message : variable locale de type str  
    message= "Début des cours : " + str(datetime.date.today())  
    print(message)
```

APPELS

```
def presentation_L3():  
    """ Fonction de présentation de la L3 """  
    #choix : variable locale de type str $  
    print("Bienvenue en L3 informatique")  
    date_rentree_L3()  
    choix=input("Voulez-vous revoir la date de rentrée (O/N)? ")  
    if choix=="O" or choix=="o" :  
        date_rentree_L3()
```


Fonction paramétrée sans résultat

paramètre formel

Simple annotation

Pas de vérification du typage à l'interprétation.

Ce travail n'est réalisé qu'à l'exécution

```
def bienvenue (nom:str):  
    """ Fonction de bienvenue en L3 """  
    #message : variable locale de type str  
    message= "L3 informatique \n" \  
    +"Bienvenue Monsieur ou Madame " + str(nom)  
    print(message)
```

paramètres effectifs

```
def presentation_L3():  
    """ Fonction principale de présentation de la L3 """  
    #nom1,nom2 de type String  
    nom1 = input("Tapez le nom de la première personne :")  
    bienvenue(nom1)  
    nom2 = input("Tapez le nom de la deuxième personne :")  
    bienvenue(nom2)  
    date_rentree_L3()
```

Vérification de compatibilité entre appels et déclaration

- Nombre de paramètres effectifs identique au nombre de paramètres formels
- Types deux à deux compatibles:
 - chaque paramètre effectif doit avoir un type compatible avec le type du paramètre formel qui lui correspond

Déclaration

```
def  nom_fonction(f1 : type1, f2: type2, f3: type3)  
...
```

Appels

```
def fonct_appelante()  
# p1, x de type1  
#p2,y de type2  
#p3,z de type3  
...  
    nom_fonction( p1, p2, p3)  
...  
    nom_fonction( x, y, z)
```

Un paramètre effectif peut-être une variable, une constante ou une expression

Spécificités de python

- En python, il est possible d'affecter une valeur par défaut à un paramètre. Les appels pourront ainsi omettre certains paramètres effectifs.

```
def ma_fonction(pos_1, pos_2, mot_1='a', mot_2=3, mot_3='bonjour') :  
    # ...  
    # exemples d'appels:  
ma_fonction(2) # erreur de syntaxe: 2 arguments positionnels requis  
ma_fonction(2, 3) # pos_1 aura la valeur 2; pos_2 aura la valeur 3;  
# les arguments mot_1, mot_2 et mot_3 auront leur valeur par défaut  
ma_fonction(2, 3, 4) # pos_1 aura la valeur 2; pos_2 aura la valeur  
3; # mot_1 aura la valeur 4; # mot_2 et mot_3 auront leur valeur par  
défaut  
ma_fonction(2, 3, mot_2=4) # pos_1 aura la valeur 2; pos_2 aura la  
valeur 3; # mot_2 aura la valeur 4; # mot_1 et mot_3 auront leur  
valeur par défaut
```



Fonction renvoyant
un résultat

Résultat d'une fonction

- Une fonction peut renvoyer un **résultat**
- Le résultat de la fonction peut être d'un type quelconque: int, float, boolean, list, tuple...
- Le corps d'une fonction renvoyant un résultat comporte au moins une instruction **return** suivi d'une expression conforme au type du résultat de la fonction

```
def nom_fonction (x) -> typeResultat :  
    instructions  
    return expression
```

Annotation spécifiant le type du résultat
Non obligatoire mais conseillé!!

Fonction renvoyant un résultat

Annotation spécifiant le type du résultat

```
import datetime
def date_rentree_L3() -> str :
    """ Procédure d'affichage de la date de début des cours """
    # message de type str
    message= "Début des cours : " + str(datetime.date.today())
    return message
```

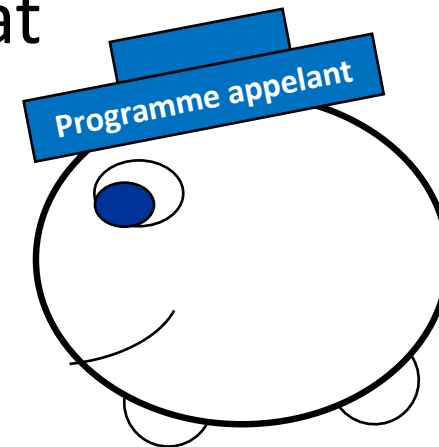
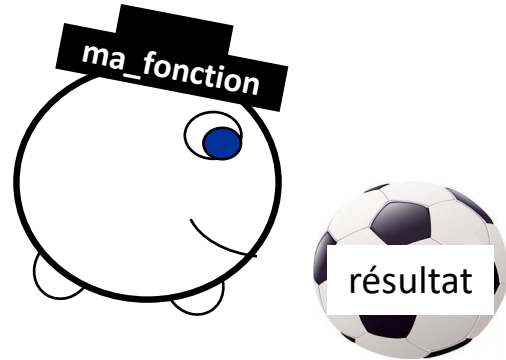
Renvoi du résultat de la fonction

```
def presentation_L3() -> None:
    """ Procédure de présentation de la L3 """
    #choix de type str
    print("Bienvenue en L3 informatique")
    print(date_rentree_L3())
    choix=input("Voulez-vous revoir la date de rentrée (O/N)? ")
    if choix=="O" or choix=="o" :
        print(date_rentree_L3())
```

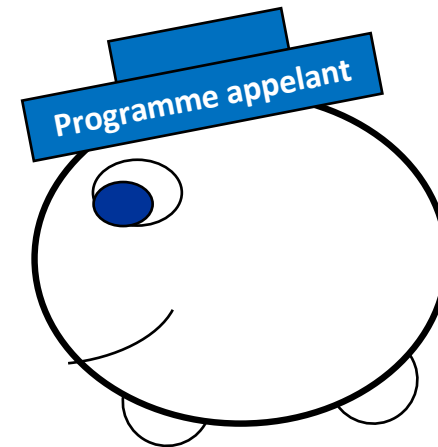
Annotation spécifiant que la fonction ne renvoie aucun résultat

Appel d'une fonction renvoyant un résultat

- Si une fonction renvoie un résultat, le programme qui l'appelle **doit récupérer** ce résultat



- Si le programme appelant ne le récupère pas, le résultat sera perdu!!



Appel d'une fonction renvoyant un résultat

- L'appel de la fonction doit apparaitre dans une **expression** conformément au type de son résultat.
- Exemples

Partie droite d'une affectation

```
x = nom_fonction ()
```

#x de même type que le résultat de la fonction

Affichage

```
print(nom_fonction())
```

Condition

```
if nom_fonction() == x :
```



Fonction avec
résultat et
paramètres

Fonction paramétrée sans résultat

Annotations spécifiant le type du résultat
Et du paramètre

```
def bienvenue (nom:str)-> str:
    """Fonction de bienvenue en L3"""
    #message : variable locale de type str
    message= "L3 informatique \n" \
    +"Bienvenue Monsieur ou Madame " + nom
    return message
```

```
def presentation_L3()->None:
    """Procédure principale de présentation de la L3"""
    #nom1,nom2 de type String
    nom1 = input("Tapez le nom de la première personne :")
    print(bienvenue(nom1))
    nom2 = input("Tapez le nom de la deuxième personne :")
    print(bienvenue(nom2))
    print(date_rentree_L3())
```

Exemple de Fonction de type float

```
def celsius(temperature :float)->float:
    """convertit la température donnée en paramètre en
    degrés Fahrenheit en degrés Celsius"""
    #tcelcius de type float
    tcelsius = 5 * (temperature - 32) / 9
    return tcelsius
```

```
def exfonction1():
    """Exemple de fonction avec résultat de type entier"""
    #temp_f, temp_c de type float
    temp_f = float(input("Entrez la température en degrés
        Fahrenheit :"))
    temp_c = celsius(temp_f)
    print("La température en degrés Celsius est :", temp_c)
```

Exemple de Fonction de type bool

```
def est_valide(temp :float) ->bool:
    """prend la valeur true si la température passée en
    paramètre est comprise entre 50 et 120"""
    #valide de type boolean
    return (temp >= 50 and temp <= 120 )
```

```
#autre écriture possible
valide=False
if temp >= 50 and temp <= 120 :
    valide=True
return valide
```

```
def exfonction()->None:
    """Exemple d'appel de fonction booléenne"""
    #temp_f de type float
    temp_f = float(input("Entrez la température en degrés
        Fahrenheit : "))
    if not est_valide(temp_f):
        print("La température doit être comprise entre 50
            et 120 ! ")
    else:
        print("La température en degrés Celsius est : "
            ,celsius(temp_f))
```



Portée des variables

Notions de variables
locales et globales

Notion de portée des variables

- Les règles de portée des variables définissent quand et où une variable est accessible
- Une règle générale
 - Une variable est accessible à l'intérieur du bloc de code où elle est définie (première affectation)

Bloc de code

- module
- fonction
- structure de contrôle

Variables Locales et Globales

- Les variables et les constantes définies à l'intérieur d'une fonction sont dites **locales** à la fonction. Elles ne peuvent être utilisées que dans la fonction où elles sont définies.
- En python, les variables et les constantes définies à l'extérieur d'une fonction peuvent être utilisées dans une fonction à condition de le spécifier explicitement en utilisant le mot clé **global**.
Elles sont alors dites **globales**.

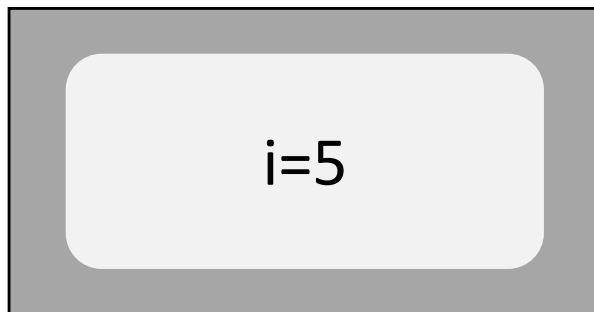
Règle d'homonymie

En cas d'homonymie entre une variable locale et une variable globale, c'est la variable locale qui est considérée.

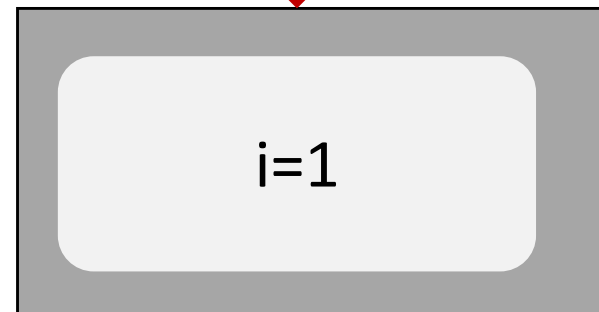
Variables Locales et Globales

exemple

```
i=5
def init_i() :
    #fonction initialisant i
    i=1
init_i()
print("i=" ,i)
```



```
i=5
def init_i() :
    #fonction initialisant i
    global i
    #i est déclarée globale
    i=1
init_i()
print("i=" ,i)y
```



Pourquoi il faut éviter les variables globales ?

- Difficultés de mise au point et de débogage
 - Interactions dangereuses en cas de modifications par plusieurs fonctions.
- Fonctions non réutilisables

Attention aux effets de bords...

Il faut privilégier les paramètres

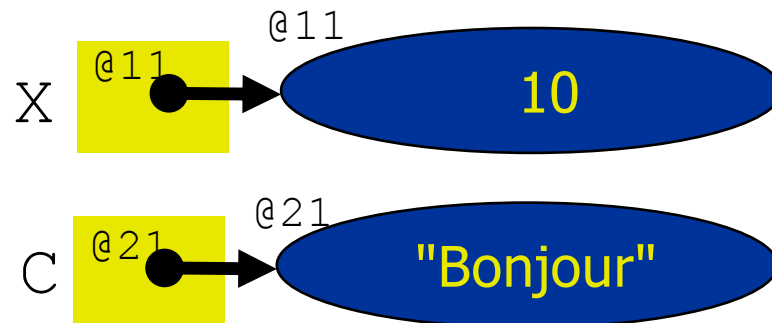


Objets mutables et immutables

Notion d'objet

- En python tout est **objet**!
- Les variables ne contiennent pas une valeur mais contiennent une **référence** (adresse mémoire ou id) vers un objet

```
X=10 #variable de type entier  
C = "Bonjour" #variable de type chaine
```

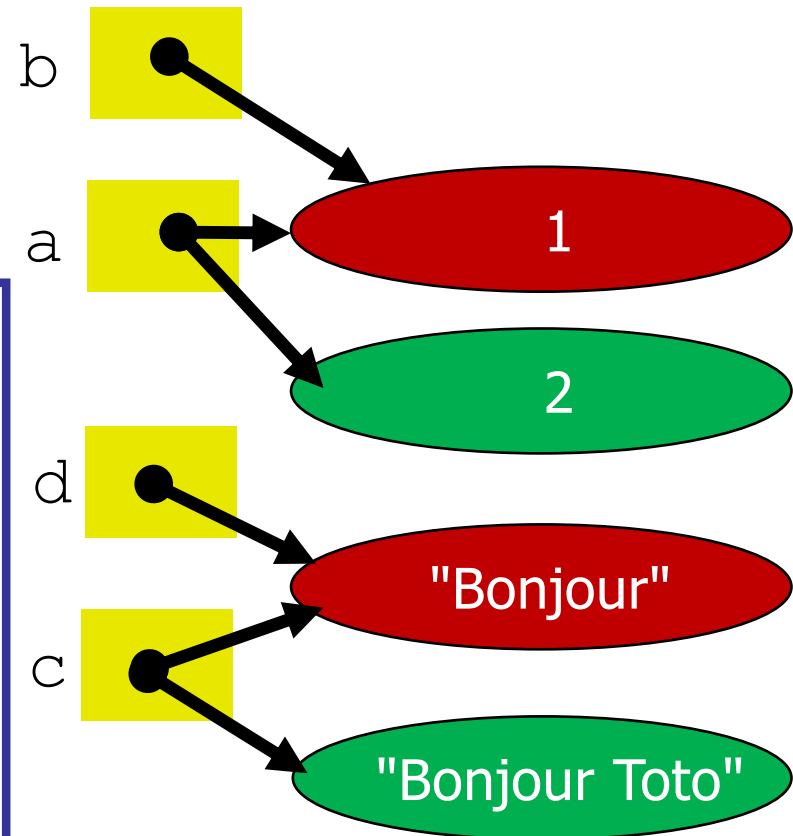


Notion d'objets immutables

- Certains objets en python sont **immuables** : leurs valeurs ne peuvent pas être modifiées
 - Types de base: entier, réels, booléens
 - Chaines
 - Tuples

```
a=1 #variable de type entier
b=a
a=a+1

c = "Bonjour" #variable de type chaine
d=c
c= c + " Toto"
```



Fonction id

- La fonction **id()** en python renvoie un entier, représentant l'identifiant interne (adresse mémoire) d'une variable quel que soit son type.

```
print("id(1) ", id(1))  
a=1  
print("id(a) ", id(a))  
b=a  
print("id(b) ", id(b))  
a=a+1  
print("id(a) ", id(a))  
print("id(b) ", id(b))  
c="bonjour"  
print("id(c) ", id(c))  
d="bonjour"  
print("id(d) ", id(d))  
c=c+" Toto"  
print("id(c) ", id(c))  
print("id(d) ", id(d))
```



```
id(1) 4297148528  
id(a) 4297148528  
id(b) 4297148528  
id(a) 4297148560  
id(b) 4297148528  
id(c) 4444000072  
id(d) 4444000072  
id(c) 4581526192  
id(d) 4444000072
```



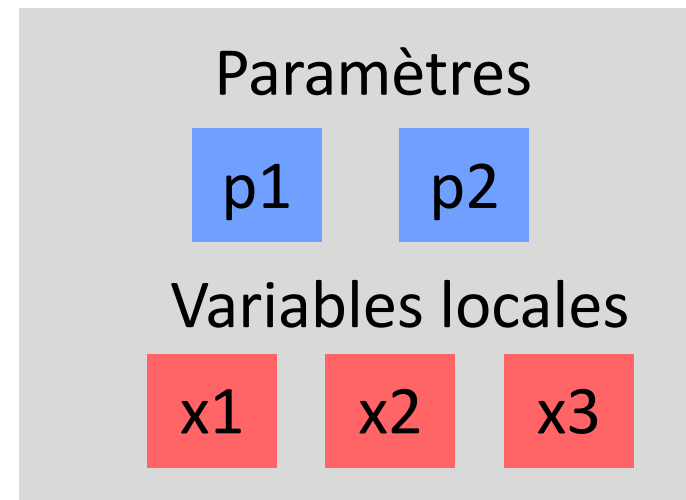

Principes de transmission des paramètres

Principes de transmission des paramètres à une fonction



Lorsque une fonction est appelée:

1. Une zone mémoire est allouée (empilée) pour
 - Ses paramètres
 - Ses variables locales
2. Ses paramètres sont initialisés en fonction des paramètres effectifs utilisés dans l'appel
3. La fonction s'exécute

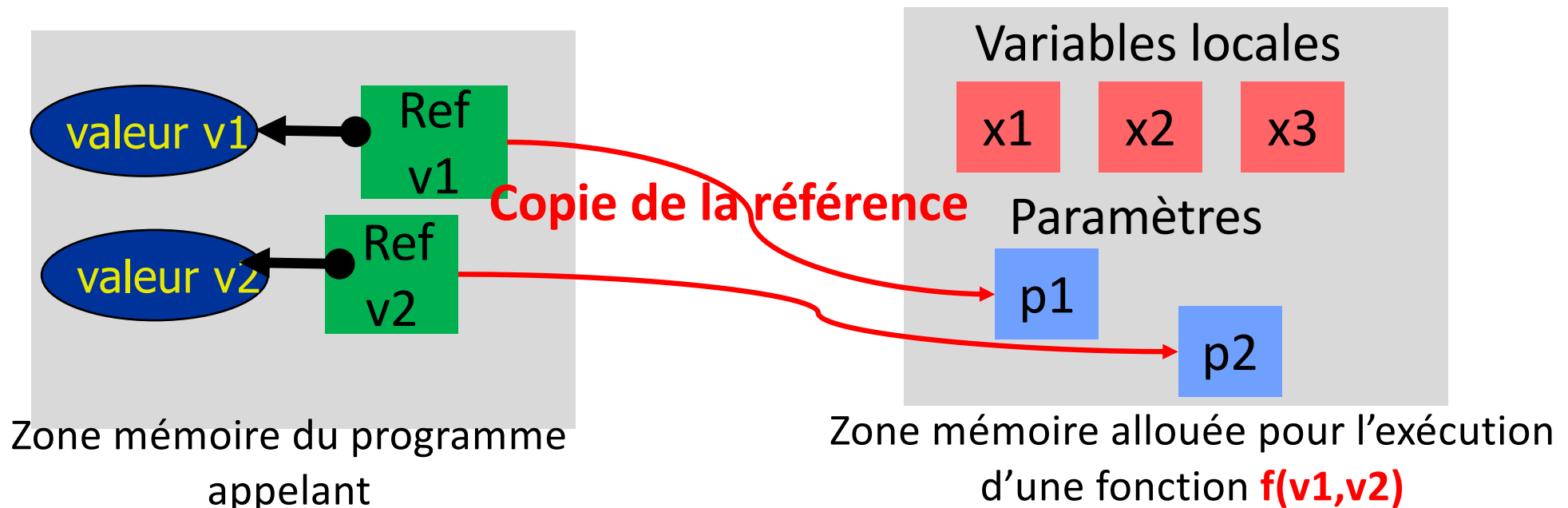


Zone mémoire allouée pour l'exécution d'une fonction $f(v1, v2)$

Mise en place de la Transmission des paramètres

Principes de transmission des paramètres à une fonction

- En python, la transmission se fait « **par valeur (copie)** » mais comme toutes les variables contiennent des références cela revient à une copie de références :
 - Les références contenues dans les paramètres effectifs (utilisés dans l'appel) sont copiés dans les paramètres de la zone mémoire de la fonction



Fonctions en python

Transmission d'un paramètre de type immutable

```
def modif_entier(x:int)->None :  
    → x=x+5
```

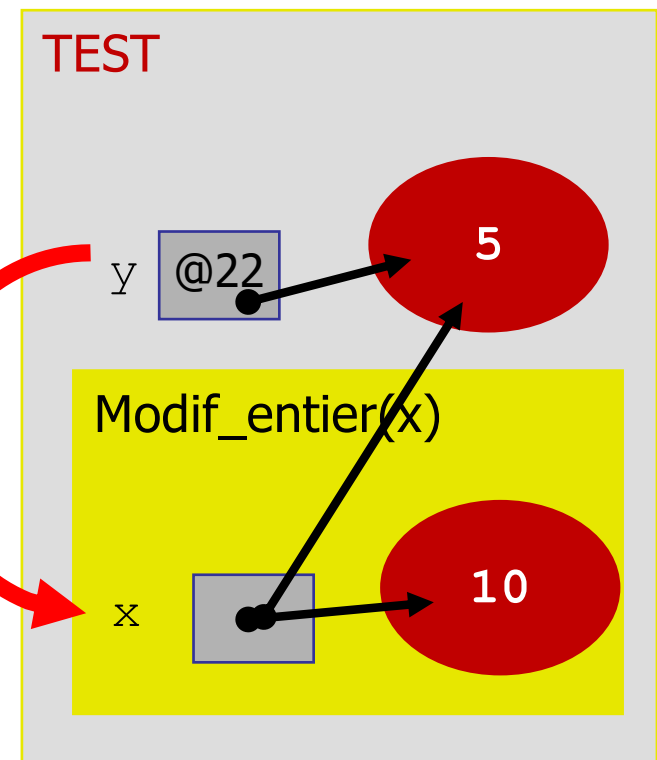
#TEST

```
y=5  
print("Avant appel y= ",y)  
modif_entier(y)  
print("Après appel y= " ,y)  
}
```

Aucun effet de bord possible pour les objets immutables

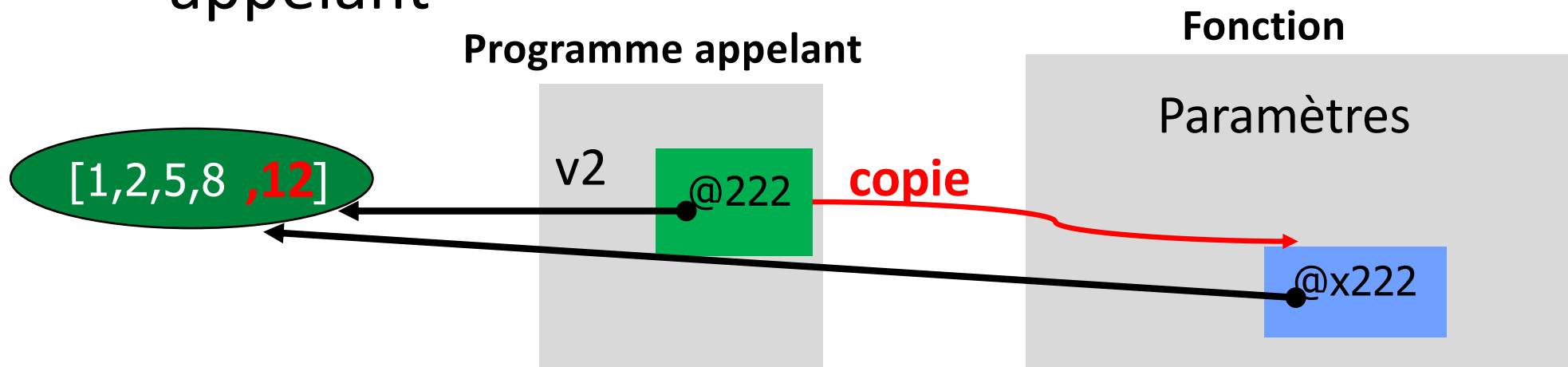
Avant appel y= 5
Après appel y= 5

Copie



Principes de transmission des paramètres à une fonction

- La transmission d'un paramètre de type **mutable** permet à une fonction de modifier le paramètre effectif associé dans le programme appelant



Nous y reviendrons dans le prochain atelier!