Le Mans Université

Introduction au langage Python

M1 Economie de la Santé, MBFA, ETRH

Alexandre Popier
alexandre.popier@univ-lemans.fr

Modalités

- Cours magistral : 4h
- TP : 10h00
- Evaluation
  - Examen de TP sur le débogage
    - 30 minutes
    - Coefficient 1
  - Examen de TP final
    - 1h30 heure
    - Coefficient 3

Idée de l'UE

- Introduction à la programmation et à l'algorithmique
- Syntaxe et spécificités du langage
- Bonnes pratiques
- Comment programmer
  - « Ahhh !! Mais comment je fais pour écrire cette boucle ??! »
- Déboguer son code
  - « Monsieur, ça ne marche pas et je ne comprend pas pourquoi. »

Seules ressources autorisées aux examens

- Documentation officielle de Python
  - Tout ce qui se trouve derrière l'adresse

    https://docs.python.org
  - Tutoriel de la documentation officielle

    https://docs.python.org/fr/3/tutorial/
- Aide-mémoire
  - A créer soi-même
  - Autorisé en version papier aux examens
    - Un document agrafé de 2 feuilles (4 pages) maximum
- Memento Python 3 et NumPy
  - Sur Umtice (Clé espace cours : python_2223)

Travailler est gratuit !

- Anaconda (windows, mac, linux)
  - Jupyter : Notebook
  - Spyder : script et terminal
    - pour des projets plus sérieux.
    - sera utilisé en TP
  - Inclut les bibliothèque standards comme
    - NumPy
    - Pandas

    … qui seront utilisées aussi dans d'autres UE

et dans la vie de tous les jours pour les futurs « data scientists »

Le Mans Université

Introduction

La programmation, c’est quoi ?

La programmation impérative

- Nombreux langages
  - Pascal, Basic, C
  - Java, php, C++, C#, Python

- Une seule idée de départ
  - Suite d'instructions
  - Résoudre un problème, obtenir un résultat

Exemple : racines d'un trinôme

```
# importation
from math import sqrt

# Résolution de l'équation ax² + bx + c = 0
a = int(input("a: "))
b = int(input("b: "))
c = int(input("c: "))

delta = b ** 2 - 4 * a * c

if delta > 0:
    x1 = (-b - sqrt(delta)) / (2 * a)
    x2 = (-b + sqrt(delta)) / (2 * a)
    print ("Les deux racines sont {} et {}.".format(x1, x2))
elif delta == 0:
    x1 = -b / (2 * a)
    print ("La racine double est {}.".format(x1))
else:
    print("Il n'y a pas de racine.")
```

Le Mans Université

Concepts de base

Variable

Fonction

Portée des variables

Variable : définition

- Case dans la mémoire
- Contient une donnée
- Possède un type
  - Entier signé (entier relatif)
  - Flottant : nombre du type $a.10^p$ avec a et p des entiers relatifs
  - Chaîne de caractères
- Désignée par un nom dans le code
  - commence par une lettre
  - pas de caractères spéciaux

Variable

- Déclaration
  - Pas de déclaration de variable en python
  - Fait automatiquement à la première affectation
- Affectation
  - Donner une valeur à la variable
  - Effectuée après l'évaluation
- Exemple
  - a = 2 — a : 2
  - b = a + 5 — a : 2 ; b : 7
  - a = a * b — a : 14 ; b : 7

Fonction : définition

- Définie par l'utilisateur ou déjà existante
- Prend n valeurs en entrée (n >= 0)
- Renvoie (souvent) une valeur de sortie
- **Attention** : Indentation
- Exemple :

```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat = CarrePlusUn(7)
print(resultat)
```

Fonction : rôles

- Factoriser le code
  - Eviter les répétitions de blocs similaires
- Rendre le code plus lisible
  - Exprimer le code appelant sans aller voir dans la fonction
  - Regrouper le code par unités de sens
- Partager des fonctionnalités avec d'autres développeurs, d'autres projets
- Exemple : valeur absolue
  - Inutile de réécrire le détail de son calcul à chaque fois qu'on en a besoin

Portée de variables

- Une variable est accessible :
  - dans la fonction / procédure où elle est déclarée
  - dans les fonctions appelées par cette fonction / procédure (mais cela est rarement judicieux)

- Exemple :

```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

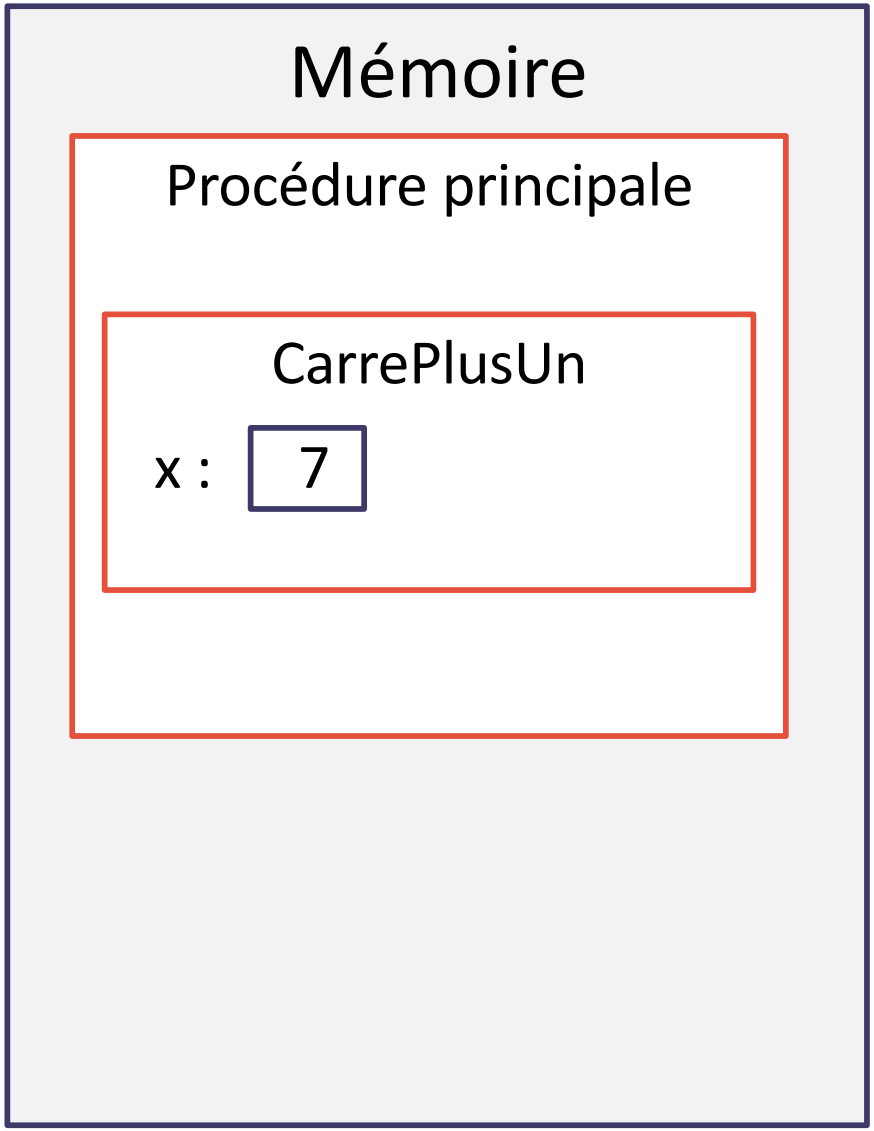
x et **valeur** sont accessibles uniquement dans **CarrePlusUn**

resultat1 et **resultat2** sont accessibles dans la procédure principale.
Ils le sont également depuis **CarrePlusUn**, mais même si cela est techniquement possible, cela est souvent déconseillé.

Portée des variables

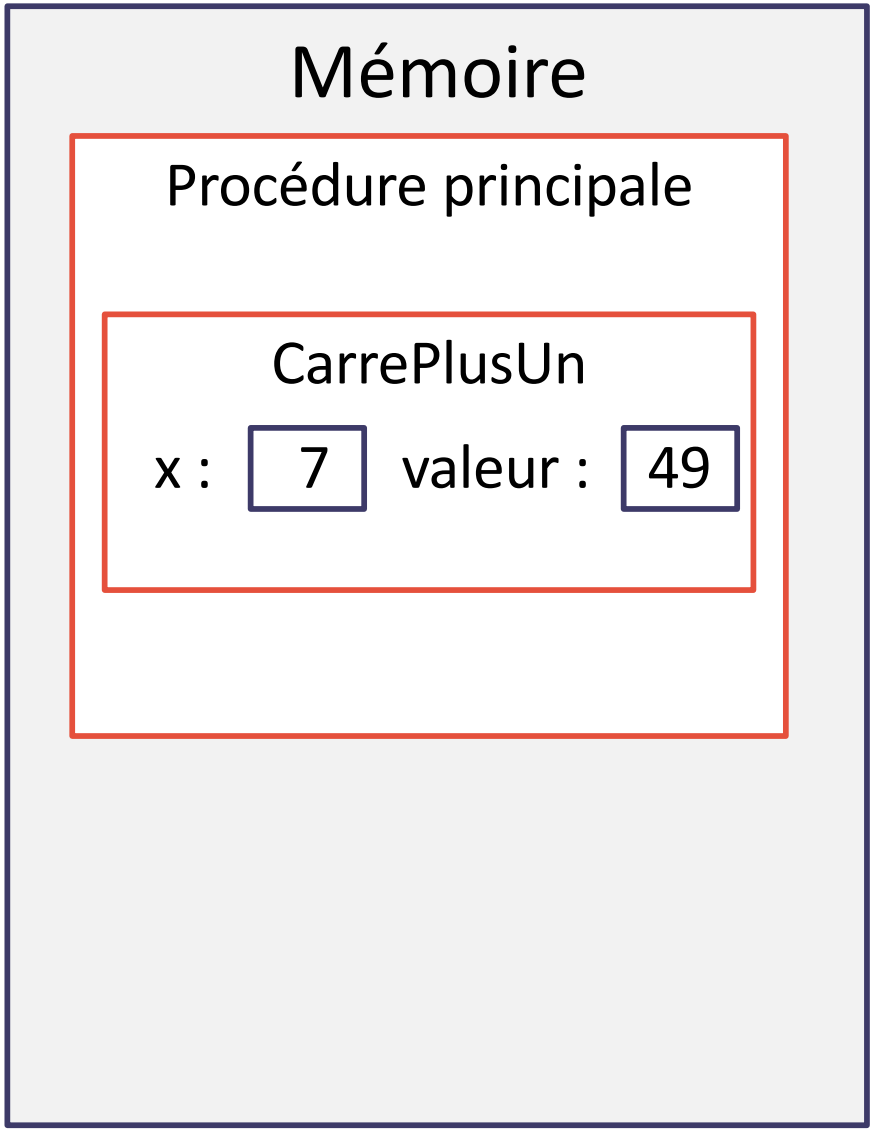
```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

La procédure principale appelle CarrePlusUn avec 7 passé en paramètre.



Portée des variables

```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```



Evaluation de x ** 2 : 49.

Affectation à la variable **valeur** déclarée dans la fonction.

valeur n'existe pas dans la procédure principale.

Portée des variables

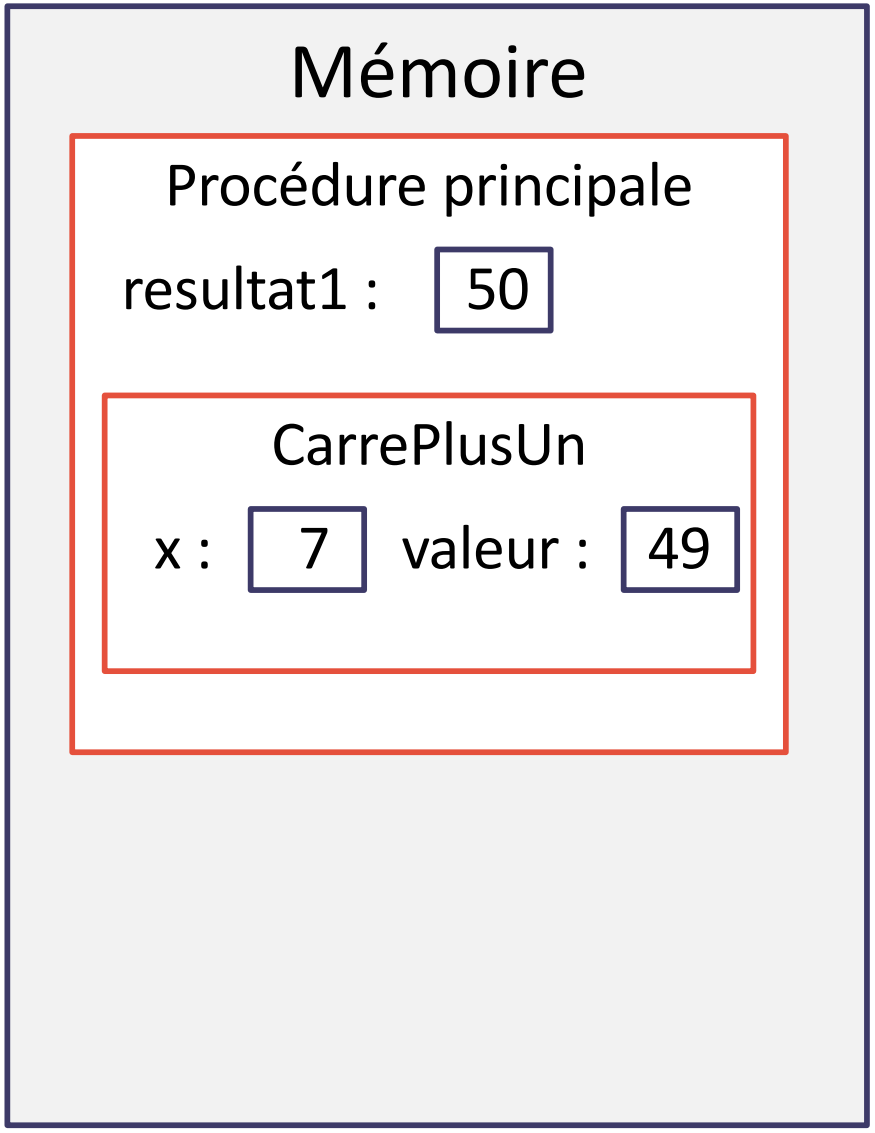
```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

Evaluation de **valeur + 1** : 50

Fin de la fonction.

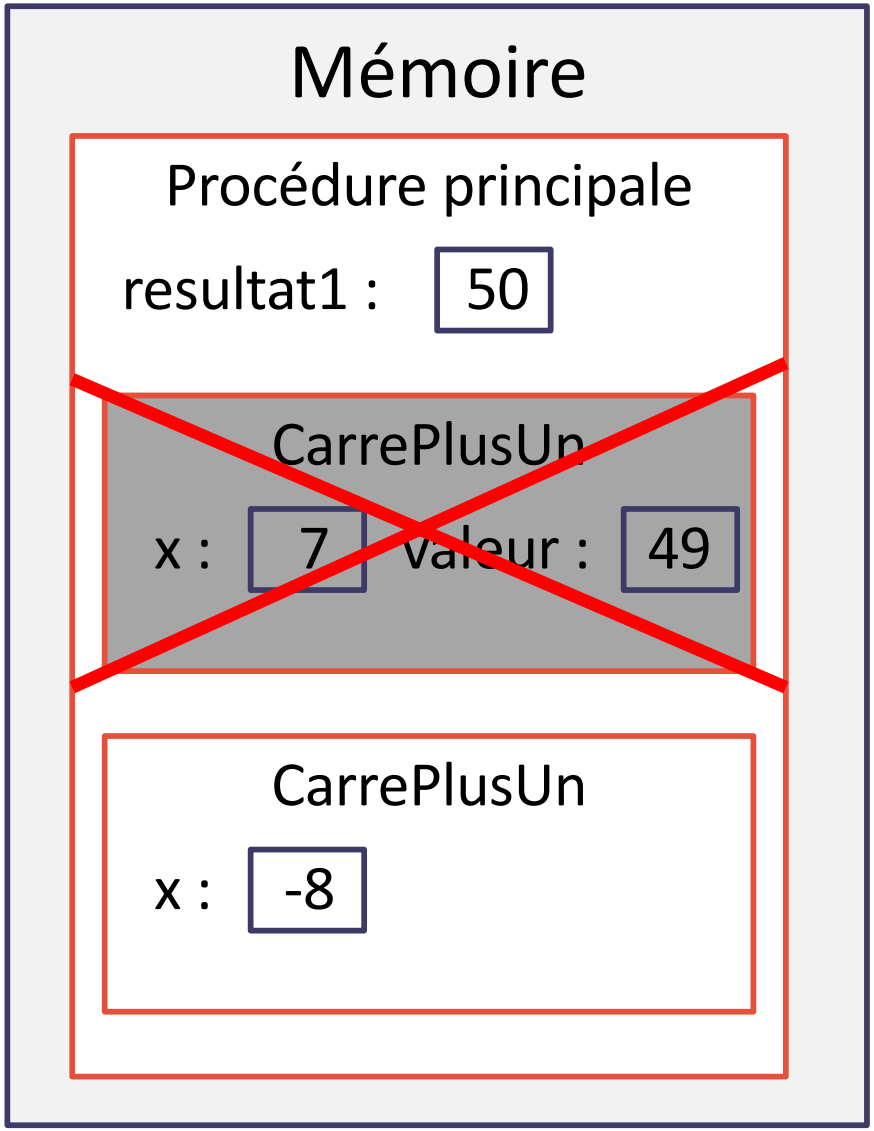
La valeur 50 est envoyée à la procédure principale comme retour de la fonction.

Affectation de **resultat1** dans la procédure principale.



Portée des variables

```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```



L'appel à **CarrePlusUn** est terminé donc la mémoire réservée est libérée.

Evaluation de **CarrePlusUn(-8)**

Appel à **CarrePlusUn**

Portée des variables

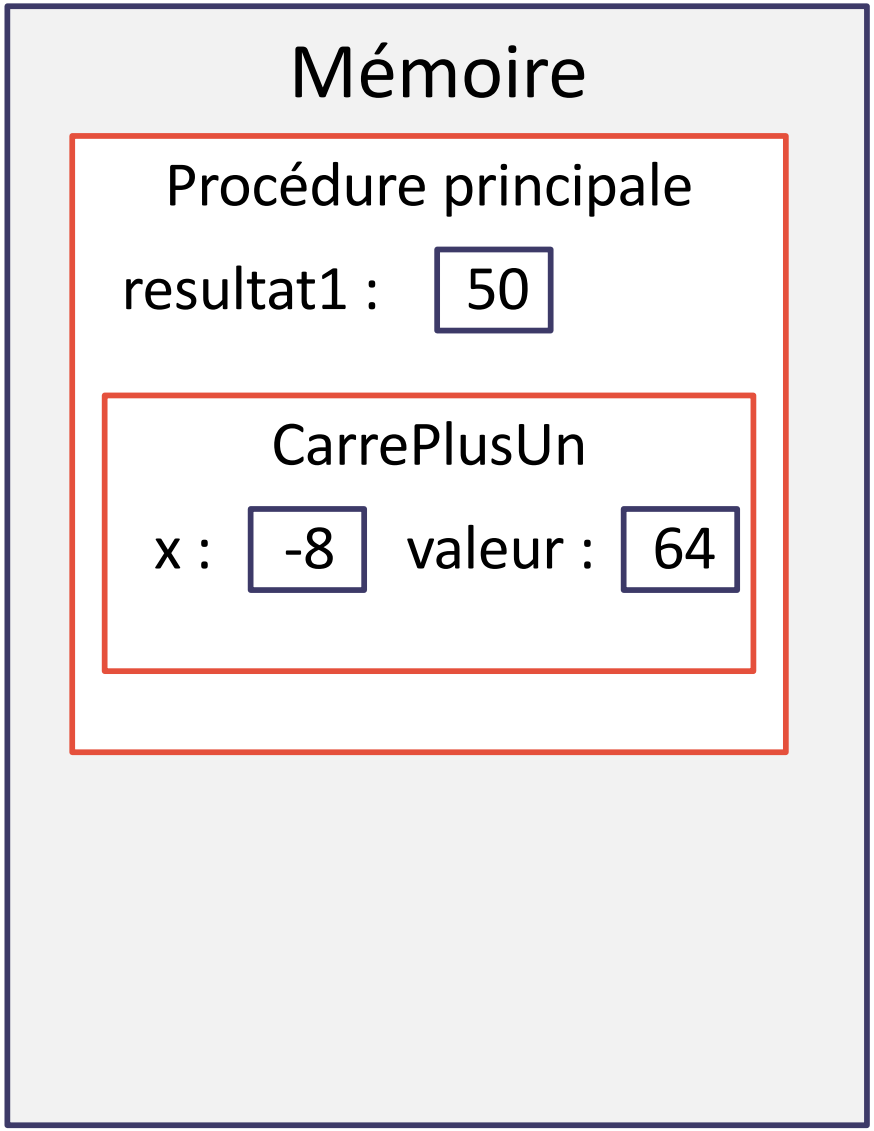
```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

Evaluation de x ** 2 : 64.

Affectation à la variable **valeur** déclarée dans la fonction.

valeur n'existe pas dans la procédure principale.

resultat1 est accessible depuis **CarrePlusUn** mais il n'est pas utilisé (cela ne serait pas propre).



Portée des variables

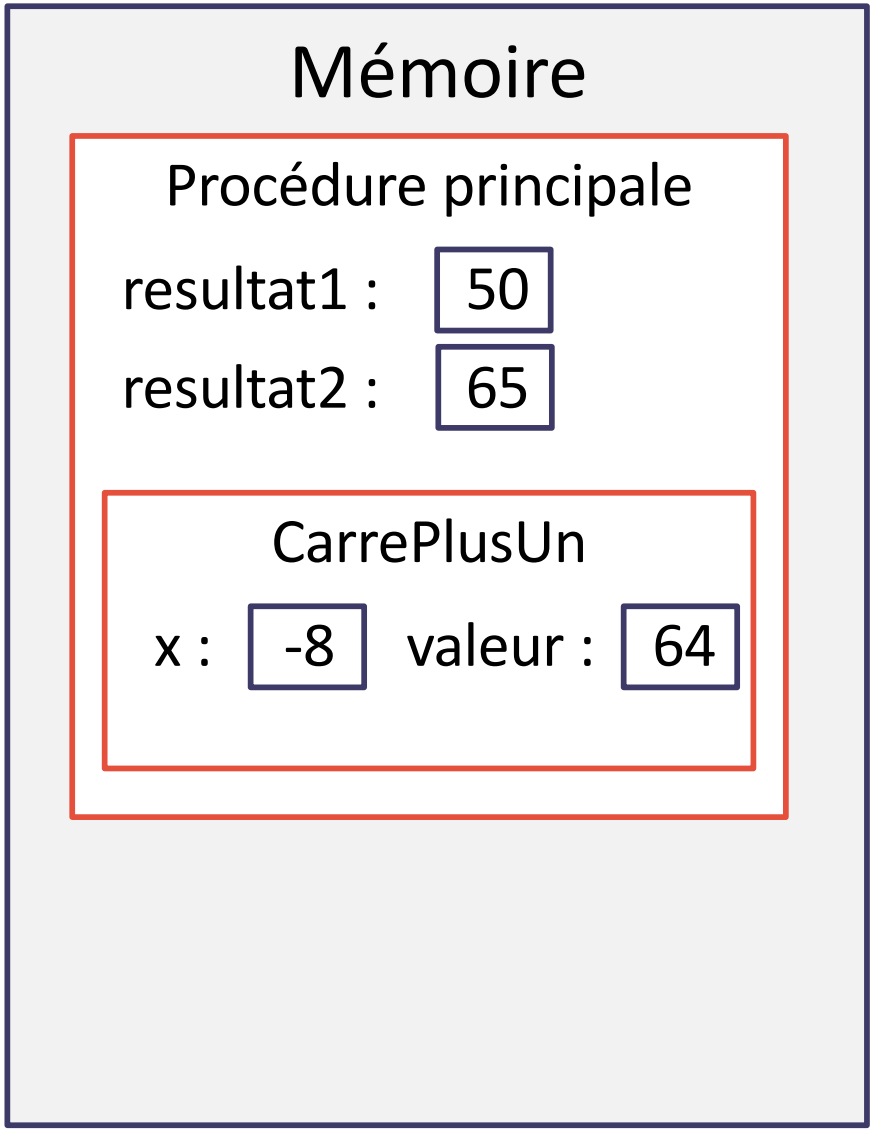
```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

Evaluation de **valeur + 1** : 65

Fin de la fonction.

La valeur 65 est envoyée à la procédure principale comme retour de la fonction.

Affectation de **resultat2** dans la procédure principale.

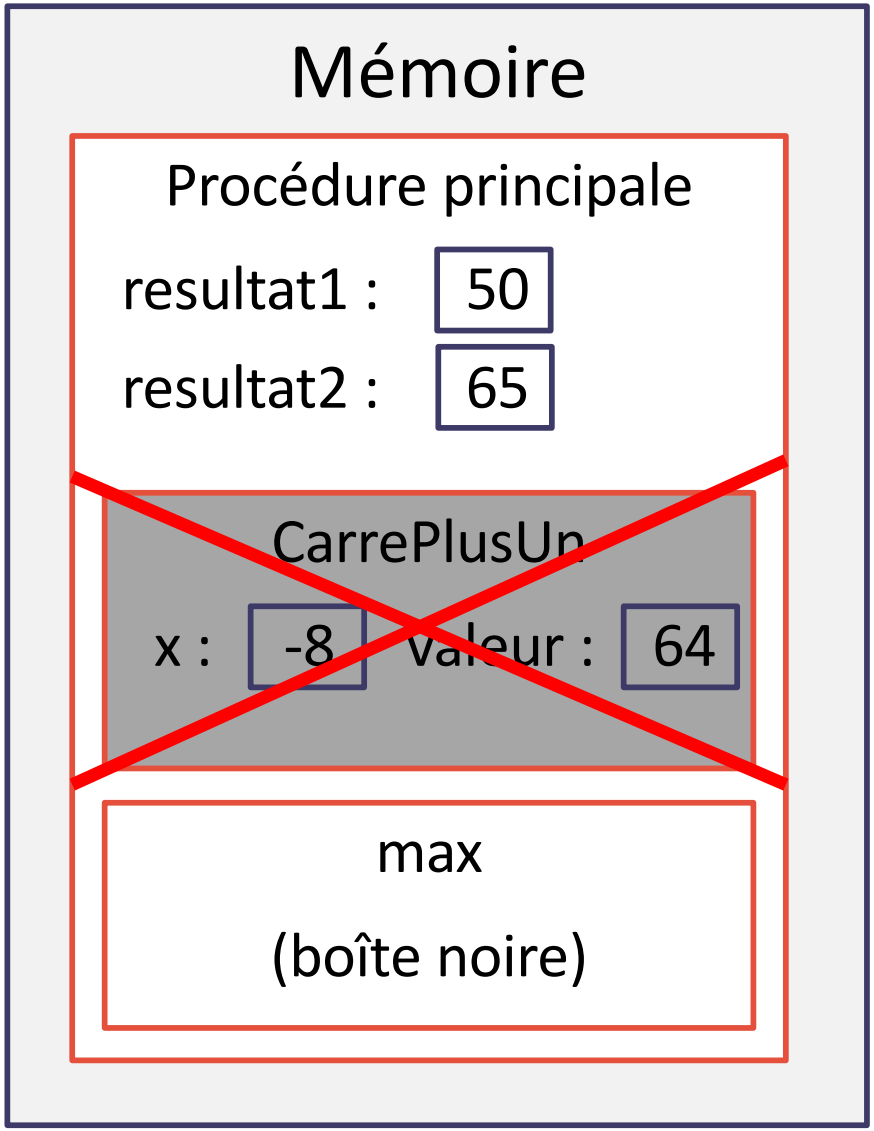


Portée des variables

```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

Evaluation de max(50, 65).

Appel à **max** qui est une fonction fournie par Python.

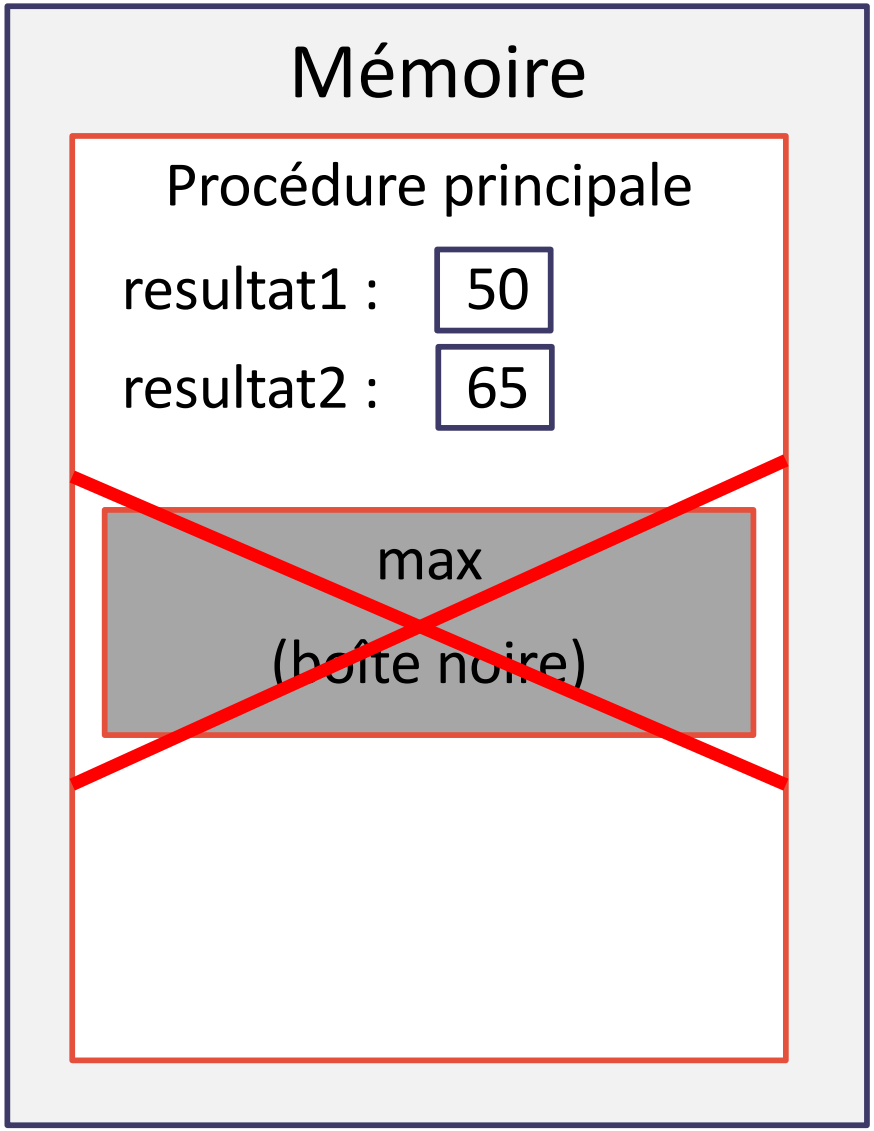


Portée des variables

```
def CarrePlusUn(x):
    valeur = x ** 2
    return valeur + 1

resultat1 = CarrePlusUn(7)
resultat2 = CarrePlusUn(-8)

print(max(resultat1, resultat2))
```

max renvoie 65.

Affichage du résultat de l'évaluation (65) sur la console.



Le Mans Université

Structures de données

Listes

Tuples

Ensembles

Liste

- Remplace le tableau classique des autres langages
- Suite de valeurs
- Pas de type unique imposé
  - Mais usuellement le cas
- Syntaxe
  - Déclaration et affectation :

    maSuperListe = [18, 24, -7, 5, 2]
  - liste de longueur 5 :

    len(maSuperListe) renvoie 5

Liste

- maSuperListe = [18, 24, -7, 5, 2]
- Accès aux données
  - Accéder à une valeur (l'indexation commence à zéro)
    - maSuperListe[2] renvoie -7. C'est un entier
  - Accéder à une sous-liste
    - maSuperListe[1:3] renvoie [24, -7]. C'est une liste
    - Remarque : l'indice de début est inclus, celui de fin est exclu
  - Ces accès sont en lecture et en écriture.
    - maSuperListe[2] = 0
      - maSuperListe vaut alors [18, 24, 0, 5, 2]
    - maSuperListe[0:3] = [1, 2, 3]
      - maSuperListe vaut alors [1, 2, 3, 5, 2]

Liste

- Méthodes associées aux listes
  - maSuperListe.append(37)
    - Ajoute 37 à la fin de la liste
  - maSuperListe.insert()
  - maSuperListe.pop()
  - maSuperListe.remove()
  - maSuperListe.index(18)
  - maSuperListe.clear
  - …
- Regarder la doc pour plus d'information
  - Chapitre 3.1.3 du tutoriel de la documentation
  - Chapitre 5.1 pour des informations avancées

Tuple

- Suite ordonnée de valeur
- Saisi entre parenthèses
  - Sauf si ce n'est pas indispensable
- Exemple
  - tup = (1, 2, 3)
  - tup = 1, 2, 3
- Non mutable
  - On peut accéder à seulement un élément ou une sous-partie
    - print(tup[1]) # renvoie 2
    - print(tup[0: 2]) # renvoie (1, 2)
  - Mais pas modifier une partie
    - tup[1] = 6 # illégal
    - tup[0: 2] = (5, 6) # illégal

Tuple : usage fréquent

- Récupération des retours d'une fonction

```
def AireEtPerimetreRectangle(longueur, largeur):
    aire = longueur * largeur
    perimetre = 2 * (longueur + largeur)
    return aire, perimetre

a, p = AireEtPerimetreRectangle(8, 4)
if a == p:
    print("Incroyable !")
```

- (aire, perimetre) est un tuple créé avec ces deux valeurs : (32, 24)
  - Emballage de tuple
- (a, p) est un tuple auquel on affecte la valeur de retour de la fonction **AireEtPerimetreRectangle**
  - Déballage de séquence
  - a et p sont déclarées et affectées par les valeurs 32 et 24

Ensemble (set)

- Ensemble au sens mathématique
  - Pas d'ordre entre les éléments
  - Pas de doublons
- Création
  - eleves = {"Marc", "Olivia", "Marceline", "Germain"}
- Plus d'info dans la doc
  - Chapitre 5.4 de la documentation

Dictionnaire

- Paires clés – valeurs
  - Unicité des clés
- Création
  - tel = {'jack': 4098, 'sape': 4139}
- Modification
  - Ajout : tel['guido'] = 4127
  - Modification tel['sape'] = 7544
  - Suppression : **del** tel['sape']
- Consultation
  - print (tel['guido'])

Dictionnaire : opérations diverses

- Liste (des clés) depuis dictionnaire
  - list(tel)
    - ['jack', 'guido', 'irv']
  - sorted(tel)
    - ['guido', 'irv', 'jack']

- Création par compréhension
  - {x: x**2 **for** x **in** (2, 4, 6)}
    - {2: 4, 4: 16, 6: 36}

Le Mans Université

Syntaxe

Entrées / Sorties

Tests

Boucles

Entrées / Sorties console

- Affichage
  - **print**(*chaine*) : affiche une chaîne de caractères sur la console
- Formatage de chaîne
  - *chaîne*.**format(***paramètres*)
  - La chaine contient une ou plusieurs fois la séquence {}
  - Chaque {} est remplacé par une des valeurs passées en paramètre, prises dans l'ordre
  - Exemple :

```
poids = 3.2
quantite = 7
phrase = "Il y a {} pièces de {} kg".format(quantite, poids)
print(phrase)
# renvoie sur la console "Il y a 7 pièces de 3.2 kg"
```

Entrées / Sorties console

- Saisie clavier
  - input(*chaine*) : affiche la chaine et invite l'utilisateur à saisir une chaine de caractères au clavier.
  - Renvoie la valeur saisie sous forme de chaine
  - Exemple :

```
animal = input("Nommez un animal.")
# si l'utilisateur saisit « Tortue », la variable de type chaine animal contiendra alors "Tortue"
```

Entrées / Sorties console

- Cast (changement de type)
  - La valeur de retour de **input** est toujours une chaine, même si l'utilisateur saisit un nombre
  - **int**(*chaine*) : traduit la chaine en entier
    - Si la chaine est un nombre non entier, prend sa partie entière
  - **float**(*chaine*) : traduit une chaine en nombre flottant
  - **str**(*valeur*) : traduit une valeur en chaîne
- Exemple d'utilisation

```
nombre = int(input("Quel âge avez-vous ?"))
naissance = 2022 - nombre    # on manipule bien des entiers
print("Vous êtes né en {} ou en {}.".format(naissance - 1, naissance))
```

Traitement conditionnel : if, else, elif

- Désigner un bloc de code à exécuter seulement si une condition est vérifiée…
- … et éventuellement un bloc à exécuter dans le cas contraire
- Syntaxe

```
if condition:
    code
    code
else:
    code
    code
code en dehors des conditions
```

- Indentation : délimite les blocs de code

Traitement conditionnel : if, else, elif

- Plusieurs alternatives sous conditions : **elif**
  - contraction de else if (sinon, si…)
  - pas de limite au nombre de **elif**
- Syntaxe

```
if condition:
        bloc de code
elif condition:
        bloc de code
elif condition:
        bloc de code
else:
        bloc de code
code en dehors des conditions
```

Expression de condition

- L'évaluation d'une condition renvoie un booléen : **True** ou **False**
- Opérateurs usuels de comparaison : ==, <, >, <=, >=
- Opérateur entre élément et liste / ensemble / tuple : **in**
- Opérations sur les booléens : **or**, **and**
- Exemple :

```
nombre = int(input("Quel âge avez-vous ?"))
if nombre < 0 or nombre >= 120:
        print("Ne me prenez pas pour un jambon !")
elif nombre <= 6:
        print("Tu es bien jeune pour savoir lire et écrire…")
else:
        naissance = 2022 – nombre
        print("Vous êtes né en {} ou en {}.".format(naissance - 1, naissance))
```

Boucle while

- Syntaxe :

  while *condition*:

  bloc de code

- La condition est évaluée puis le bloc de code est exécuté si elle est vaut **True**
- Après le bloc, cela recommence (évaluation, exécution)
- S'arrête quand la condition n'est plus vérifiée et le bloc n'est pas exécuté
  - Attention aux boucles infinies !

```
while True:
    print("Encore !")
```

Boucle while : exemples

- Exemple

```
nombre = -1
while nombre < 0 :
    nombre = float(input("Saisir un nombre positif : "))
print("Le nombre est {}".format(nombre))
```

- Exemple avec break
  - break : sortir de la boucle

```
while True:
    nombre = float(input("Saisir un nombre positif : "))
    if nombre >= 0 :
        break
print("Le nombre est {}".format(nombre))
```

Boucle for

- Parcours avec plage d'indice déterminée à l'avance
- Exemple

```
ages = [19, 21, 23, 16, 31]

for i in range(len(ages)):
    print("La personne n°{} a {} ans.".format(i, ages[i]))
```

- range(debut, fin) ou range(fin)
  - Prend toutes les valeurs entières de **debut** (inclus) à **fin** (exclu)
  - Itérateur : différent d'une liste
    - Ne contient pas les données de la liste
    - Prend les valeurs une par une
    - **for** sait travailler avec un itérateur

Boucle for

- Parcours d'une collection
  - la "variable de boucle" prend alors successivement les valeurs des éléments de la collection (liste, ensemble, tuple)
- Exemple :

```
ensemble = {12, 54, 1, 12, 777}

print("Les éléments de cet ensemble, mis au carré, sont :")

for element in ensemble:
    print(element ** 2)
```

Le Mans Université

Spécificités du Python

Langage interprété

- Langage interprété (Java, VBA)
  - L'interpréteur exécute le programme ligne par ligne
    - Plus facile pour mettre en pause et déboguer
    - Pas besoin de compiler. L'interpréteur suffit.
  - Bien plus lent qu'un langage compilé
- Langage compilé (C, C++)
  - Traduit en langage machine
    - Doit être fait pour chaque système d'exploitation (contrainte)
  - Langage machine très rapide
  - Moins souple sur la programmation de haut niveau
- Le Python est interprété
  - Mais compile certaines parties de code

Affectation de fonctions

- Possibilité d'affecter une fonction à une variable (qui désigne alors la même fonction)

```
def AireEtPerimetreRectangle(longueur, largeur):
    aire = longueur * largeur
    perimetre = 2 * (longueur + largeur)
    return aire, perimetre

f = AireEtPerimetreRectangle

a, p = AireEtPerimetreRectangle(8, 4)
print(a) # affiche 32

a, p = f(8, 4)
print(a) # affiche 32
```

Autres caractéristiques

- Très nombreuses bibliothèques
- Utilisé dans de nombreux logiciels
  - S'interface facilement avec d'autres langages
- Très utilisé dans le milieu du traitement de données
  - Nettoyage / mise en forme de données
  - Etudes statistiques
  - Simulations
  - …

Le Mans Université

Bonnes pratiques

Nommer les variables

- Noms explicites
  - éviter a, b, c
  - préférer largeur, perimetre, tauxAccroissement
- lower Camel Case
  - minuscule au début
  - Majuscule au début de chaque autre mot
  - Que des lettres non accentuées ou des chiffres
  - Pas de séparateur
  - Ex : premiereDeriveeDroite, derivee, borneSup
- Ne pas avoir peur des noms trop longs
  - Il ne faut pas exagérer non plus

Nommer les fonctions

- Mêmes remarques que pour les variables
  - On a vu que les fonctions étaient désignées par des variables
- Upper Camel Case
  - Majuscule au début
  - Ex : AireEtPerimetreRectangle

Commenter le code

- Code qui est ignoré par l'interpréteur
- Commentaire sur une ligne (ou en fin de ligne)
  - Commencer par #
  - Exemple :

```
aire = largeur * longueur # Calcul de l'aire du rectangle
```

- Commentaire multiligne
  - Entre triple guillemets
  - Ex :

```
"""
Ceci est
un commentaire sur plusieurs lignes
"""
```

Quand commenter ?

- Fonction
  - Description de son rôle
  - Les éventuels cas d'usage particuliers
  - Les entrées et sorties
- Variable
  - Si elle joue un rôle important, pas forcément facile à deviner
- Tout passage de code compliqué, subtil ou important

Pour qui commenter ?

- Pour soi-même, plusieurs jours / mois / années après
- Pour un collègue
- Pour un autre développeur
- Pour l'enseignant qui corrige

Tester le code

- Vérifier le résultat final
- Tester au fur et à mesure
  - Appeler les fonctions pour les tester
  - … avant de les utiliser pour répondre au problème
  - **Lire les messages d'erreur**
    - Lire le premier
- Exemple :
  - Fichier Erreurs.py

Le Mans Université

Comment écrire du code ?

« Ahhh !! Mais comment je fais pour écrire cette boucle ??! »

Réflexion à la main

- Bonne habitude
- Gagner du temps sur les problèmes complexes
- Dessins
  - Objets manipulés
  - Evolution du problème
- Pseudo-code, algorithme graphique
  - Ecrire cela sous forme de commentaire avant d'écrire le code
- Exemple :
  - Suite de Syracuse : $u_{n+1} = u_n / 2$ si u_n est pair
    $u_{n+1} = 3\ u_n + 1$ si u_n est impair
  - Indice du premier terme valant 1
  - L'utilisateur saisit u_0

Écrire une boucle

- Expert :
  - Ecrire directement la boucle
- Pas suffisamment expert
  - Ecrire le début des instructions à la main
  - Identifier ce qui se répète
  - Introduire un indice
  - for ou while ?
    - Nombre d'itérations déterminé facilement à l'avance ? → for
    - Parcours d'une collection ? → for
  - Factoriser
- Exemple :
  - Retourner une liste en ne gardant que les valeurs paires
  - Résultat dans une nouvelle liste

Le Mans Université

Déboguer son code

« Monsieur, ça ne marche et je ne comprend pas pourquoi. »

Sur Spyder (comme en TP)

- Utiliser les outils de debug

- Positionner un point d'arrêt
  - Double clic dans la marge à gauche

- Consulter le contenu des variables

Nom	Type	Taille	Valeur
ages	list	5	[19, 21, 23, 16, 31]
i	int	1	4

Démo débogueur

- Bug1 : Liste d'aires et de périmètres à partir d'une liste de rectangles sous la forme (longueur, largeur)

```
def AireEtPerimetreRectangle(longueur, largeur):
    aire = longueur * largeur
    perimetre = 2 * longueur * largeur
    return aire, perimetre

rectangles = [(1, 2), (3, 2), (2.2, 1.3), (3.1, 0.2), (1.6, 7.2)]

aires = []
perimetres = []

i = -1

for i in range(len(rectangles)):
    i = i + 1
    a, p = AireEtPerimetreRectangle(rectangles[i][0], rectangles[i][1])
    aires.append(a)
    perimetres.append(p)

print(aires)
print(perimetres)
```

Démo débogueur

- Bug2 : produit des premiers termes d'une suite

```
def ProduitPremierTermes(suiteInitia1e):
    suiteProduits = []
    produit = 1
    for terme in suiteInitiale:
        produit *= terme
    suiteProduits.append(produit)

    return suiteProduits

suiteValeurs = [1, 2, 3, 4, 5, 6]
suiteP = ProduitPremierTermes(suiteValeurs)
print(suiteP)
```

Démo débogueur

- Bug3 : Remplacer toutes les voyelles d'une chaine par des y.

```
a=[input("entrer une chaine de caractère : ")]
L=["a","e","i","u","y"]
for i in a:
    for j in L:
        if L[j] == a[i]:
            a[i]="y"
            j+=1
        else:
            j+=1
i+=1
print(a)
```

Sans débogueur

- Reproduire le fonctionnement des outils de debug
  - Placer des **print**() qui affichent le contenu de variables
    - Voir comment ces contenus évoluent
    - Voir à quel moment le programme s'arrête

- Toujours moins pratique que le débogueur…

Le Mans Université

Bibliothèques
Modules

Ex : Numpy, Pandas

Module

- Fichier Python ***.py
  - Objets Python :
    - Fonctions
    - Variables
- Peut s'exécuter
  - Script (petit programme)
  - Application (gros programme)
- Peut s'importer

Bibliothèque (library)

- Ne s'exécute pas
- Inclut un ensemble d'outils autour d'une thématique
  - Destiné à être importé
- Est un module ou un package…
  - … donc un module
- Ex :
  - Bibliothèque standard Pyhton
  - NumPy
- Rq : concept commun à tous les langages
  - N'a pas d'existence technique en Python

NumPy

- Tutoriel
  - https://numpy.org/doc/stable/user/quickstart.html
- Memento sur UMTICE
- Pour faire des calculs : fonctions, matrices, équations, etc.

Pandas

- Tutoriel
  - https://pandas.pydata.org

- Memento sur UMTICE

- Pour la gestion de données.