LANGAGE DE PROGRAMMATION PYTHON

ASSOHOUN E. Stanislas

- *L3 Actuariat (IUA)*

PLAN

- **CHI – DESCRIPTION GÉNÉRALE DU LANGAGE**
- **CH2 – FONCTIONS ET MODULES**
- **CH3 – COLLECTIONS ET CLASSES**
- **CH4 – NUMPY- PANDAS ET MATHPLOTLIB**

CHIII

COLLECTIONS ET CLASSE

COLLECTIONS

I-1- Tuple

Définition

Un tuple est une séquence immuable : on ne peut pas modifier ses éléments ni lui en ajouter ou lui en enlever.
En contrepartie de cette rigidité les tuples sont très compacts (i.e. ils occupent peu de mémoire) et l'accès à leurs éléments est très rapide.
On crée un tuple en écrivant ses éléments, séparés par des virgules et encadrés par des parenthèses. Si cela ne crée aucune ambiguïté, les parenthèses peuvent être omises. Un tuple constitué d'un seul élément a doit être écrit « a, » ou « (a,) ». Le tuple sans éléments se note « () ».

Exemple:

```
>>> t=(2, 'deux', 2.0, True, (1, 2, 3, 4))
```

Accès au composant

L'acces au composants est fait via la syntaxe : nomtuple[indice]
Avec indice = 0,1, ….. Pour les accès dans le sens direct Ou -1, -2 , ……. le sens indirect.

COLLECTIONS

I-2- liste (1)

Définition

Une liste est une séquence modifiable pouvant contenir des objet de type différents. On peut changer ses éléments, lui en ajouter et lui en enlever. Cela rend les listes un peu moins compactes et efficaces que les tuples, mais considérablement plus utiles pour représenter des structures de données qui évoluent au cours du temps.

On construit une liste en écrivant ses éléments, séparés par des virgules, encadrés par les crochets. La liste vide se note [ ].

Opérations sur le listes

liste[indice]	désignation de l'élément de rang indice de la liste,
liste[indice] = expression	remplacement de l'élément de liste ayant l'indice indiqué,
liste1 + liste2	concaténation (mise bout-`a-bout) de deux listes,
liste.append(élément)	ajout d'un élément à la fin d'une liste,
liste.insert(indice, élément)	insertion d'un élément à l'emplacement de liste indiqué par l'indice donné,
élément in liste	test d'appartenance : élément ∈ liste ?
for élément in liste	parcours séquentiel

COLLECTIONS

I-2- liste (2)

Remarque

Il y a donc deux manière de commander l'opération « ajouter un élément à la fin d'une liste ». On prendra garde à cette différence entre elles :

liste + [élément]	ne modifie pas liste, mais renvoie une nouvelle liste comme résultat,
liste.append(élément)	modifie liste et ne renvoie pas de résultat.

Application : Calcul de la liste des nombres premiers inférieurs ou égaux à une certaine borne. Algorithme : passer en revue chaque nombre impair (les nombres pairs ne sont pas premiers), depuis 3 jusqu'à borne = 100.

```
premiers = [ ]
for candidat in xrange(3, borne + 1, 2):
        estPremier = True
        for premier in premiers:
                if candidat % premier == 0:
                        estPremier = False
                        break
        if EstPremier:
                premiers.append(candidat)
print premiers
```

COLLECTIONS

I-2- liste (3)

Parcours de liste

```
ma_liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

i = 0 # Notre indice pour la boucle while
while i < len(ma_liste):
    print(ma_liste[i])
    i += 1 # On incrémente i, ne pas oublier !

# ou for qui capture chaque element de la liste

for elt in ma_liste: # elt va prendre les valeurs successives des éléments de ma_liste
    print(elt)

# ou enumerate prend en paramètre une liste et renvoie l'indice et l'element de la liste parcourue

for i, elt in enumerate(ma_liste):
    print("À l'indice {} se trouve {}.".format(i, elt))
```

COLLECTIONS

I-3- Dictionnaires

Définition

Un dictionnaire, ou table associative, est une collection de couples (clé, valeur) telle que:
– il n'y a pas deux couples ayant la même clé,
– la structure est implémentée de manière que la recherche d'une valeur à partir de la clé correspondante soit extrêmement efficace.

Les clés ne doivent pas être des structures modifiables mais, à part cela, elles peuvent être de n'importe quel type ; les valeurs sont quelconques. On construit explicitement un dictionnaire par une expression de la forme.

{ clé1 : valeur 1 , clé2 : valeur 2 , ... clék : valeur k }

Operations sur les dictionnaires (1)

dict[clé] = valeur	ajoute au dictionnaire dict une paire (clé, valeur) ou, si une telle paire existait déjà, modifie sa partie valeur,
dict[clé]	renvoie la valeur correspondant à la clé donnée ; une erreur est déclenchée si une telle clé n'existe pas dans le dictionnaire,
del dict[clé]	supprime du dictionnaire dict la paire (clé, valeur),

COLLECTIONS

I-3- Dictionnaires

Operations sur les dictionnaires (2)

dict.get(clé [,valsinon])	renvoie la valeur correspondant à la clé donnée ; si une telle clé est absente, renvoie valsinon ou (si valsinon est omise) None,
dict.has_key(clé)	vrai si et seulement si la clé indiquée existe dans le dictionnaire dict,
dict.keys()	renvoie une copie de la liste des clés du dictionnaire dict,
dict.values()	renvoie une copie de la liste des valeurs du dictionnaire dict,
dict.items()	renvoie une copie de la liste des associations constituant le dictionnaire dict.

Application: Compter le nombre d'apparitions de chaque caractère d'un texte donné. Algorithme : construire un dictionnaire dont les clés sont les caractères du texte et les valeurs le nombre d'occurrences de chacun. Programme :

```
texte = input("Saisir phrase :")
dico = {}
for car in texte:
    if dico.get(car):
        dico[car] = dico[car] + 1
    else:
        dico[car] = 1
```

```
for car in dico.keys():
    print (car, ':', dico[car], 'occurrence(s)')
```

COLLECTIONS

I-3- Dictionnaires

Acces aux clé, valeurs et items de dictionnaires (2)

```
fruits = {"pommes":21, "melons":3, "poires":31}
# affichage des clés
for cle in fruits.keys():
    print(cle)
# affichage des valeurss
for valeur in fruits.values():
    print(valeur)
# affichage des items : clés et valeurs
for valeur in fruits.values():
    print("La clé {} contient la valeur {}.".format(cle, valeur))
```

COLLECTIONS

I-4- Fichiers

Définition

Quand on aborde les fichiers on s'intéresse pour la première fois à des données qui existent à l'extérieur de notre programme. Du point de vue du programmeur, un fichier ouvert « en lecture » doit être vu comme un tube par lequel arrivent des données extérieures chaque fois que le programme les demande, de même qu'il faut voir un fichier ouvert « en écriture » comme un tube par lequel s'en vont à l'extérieur les données que le programme y dépose.

On notera que cette approche permet de voir comme des fichiers le clavier et l'écran du poste utilisé. Cependant, les fichiers auxquels on pense d'abord sont ceux enregistrés sur des supports non volatils (disques durs, clés USB, cédéroms, etc.). Ils représentent des ensembles de données extérieures au programme, qui existent déjà quand l'exécution de celui-ci démarre et qui ne disparaissent pas quand celle-ci se termine.

Pour les fonctions de la bibliothèque Python tout fichier est considéré comme séquentiel, c'est-à-dire constitué par une succession d'informations lues ou écrites les unes à la suite des autres. Une fois qu'un fichier a été ouvert par un programme, celui-ci maintient constamment une position courante, représentée par un « pointeur » (fictif) qui indique constamment le prochain octet qui sera lu ou l'endroit ou sera déposé le prochain octet écrit. Chaque opération de lecture ou d'écriture fait avancer ce pointeur.

COLLECTIONS

I-4- Fichiers

Fonctions de traitement des fichiers (1)

fichier = open(nom_du_fichier, mode)

Ouverture du fichier. Cette fonction crée et renvoie une valeur qui représente dans le programme l'extrémité d'une sorte de tube dont l'autre extrémité est le fichier en question (entité extérieure au programme, qui vit sa propre vie dans le système environnant). nom du fichier est une chaîne de caractères qui identifie le fichier selon la syntaxe propre du système d'exploitation,

- Nom_du_fichier : est une chaîne de caractères qui identifie le fichier selon la syntaxe propre du système d'exploitation,
- mode est une des chaînes suivantes :
  - "r" : ouverture en lecture ; le fichier doit exister, le pointeur est positionné au début du fichier. Sur ce fichier seules les opérations de lecture sont permises.
  - "w" : ouverture en écriture ; le fichier existe ou non ; s'il existe il sera écrasé, (entièrement vidé). Le pointeur est positionné à la fin du fichier, qui est aussi son début. Seules les opérations d'écriture sont permises.
  - "a" : ouverture en allongement ; le fichier existe ou non ; s'il existe, il ne sera pas remis `a zéro. Le pointeur est positionné à la fin du fichier. Seules les opérations d'écriture sont permises.

    Exemple : fichier = open("resultats.txt", "r")

COLLECTIONS

I-4- Fichiers

Fonctions de traitement des fichiers(2)

fichier.read() : Lit tout le contenu du fichier indiqué et le renvoie sous forme de chaîne de caractères.
Attention, cette fonction ne convient qu'aux fichiers de taille raisonnable, dont le contenu tient dans la mémoire de l'ordinateur.

fichier.readline(): Lit une ligne du fichier indiqué (lors du premier appel la première ligne, au second appel la deuxième, etc.).
Attention. Dans les fichiers de texte, la fin de chaque ligne est indiquée par un caractère spécial, qu'on peut noter '\n' dans les programmes. En toute rigueur cette marque ne fait pas partie de la ligne, cependantles fonctions readline et readlines l'ajoutent à la fin de la ligne lue et le renvoient dans le résultat : ne pas oublier de l'enlever si nécessaire (voyez les exemples de la section suivante).

fichier.readlines(): Lit tout le contenu du fichier indiqué et le renvoie sous la forme d'une liste de chaînes de caractères (une chaîne par ligne).

fichier.write(chaîne) : Ecrit la chaîne indiquée dans le fichier en question.

fichier.writelines(séquence): Ecrit les chaînes composant la séquence (liste, tuple, ensemble, etc.) de chaînes donnée dans le fichier indiqué.

fichier.close(): Ferme le fichier indiqué. Prendre garde au fait que tant qu'un fichier nouvellement créé (c'est-à-dire un fichier ouvert en écriture) n'a pas été fermé son statut dans le système n'est pas bien défini : le fichier peut ne pas apparaître sur le disque dur, ou ne pas ˆetre complet, etc. Pour un fichier en ´ecriture, l'appel final de close est donc obligatoire.

COLLECTIONS

I-4- Fichiers

Fonctions de traitement des fichiers(2)

fichier.read() : Lit tout le contenu du fichier indiqué et le renvoie sous forme de chaîne de caractères.
Attention, cette fonction ne convient qu'aux fichiers de taille raisonnable, dont le contenu tient dans la mémoire de l'ordinateur.

fichier.readline(): Lit une ligne du fichier indiqué (lors du premier appel la première ligne, au second appel la deuxième, etc.).
Attention. Dans les fichiers de texte, la fin de chaque ligne est indiquée par un caractère spécial, qu'on peut noter '\n' dans les programmes. En toute rigueur cette marque ne fait pas partie de la ligne, cependantles fonctions readline et readlines l'ajoutent à la fin de la ligne lue et le renvoient dans le résultat : ne pas oublier de l'enlever si nécessaire (voyez les exemples de la section suivante).

fichier.readlines(): Lit tout le contenu du fichier indiqué et le renvoie sous la forme d'une liste de chaînes de caractères (une chaîne par ligne).

fichier.write(chaîne) : Ecrit la chaîne indiquée dans le fichier en question.

fichier.writelines(séquence): Ecrit les chaînes composant la séquence (liste, tuple, ensemble, etc.) de chaînes donnée dans le fichier indiqué.

fichier.close(): Ferme le fichier indiqué. Prendre garde au fait que tant qu'un fichier nouvellement créé (c'est-à-dire un fichier ouvert en écriture) n'a pas été fermé son statut dans le système n'est pas bien défini : le fichier peut ne pas apparaître sur le disque dur, ou ne pas être complet, etc. Pour un fichier en écriture, l'appel final de close est donc obligatoire.

CLASSES

II-1- Définition

Une classe est un type permettant de regrouper dans la même structure : les informations (champs, propriétés, attributs) relatives à une entité ; les procédures et fonctions permettant de les manipuler (méthodes). Champs et méthodes constituent les membres de la classe.

II-2- Remarques :

- La classe est un type structuré qui va plus loin que l'enregistrement (ce dernier n'intègre que les champs)
- Les champs d'une classe peuvent être de type quelconque
- Ils peuvent faire référence à d'instances d'autres classes

II-3- Termes techniques :

- « Classe » est la structure ;
- « Objet » est une instance de la classe (variable obtenue après instanciation) ;
- « Instanciation » correspond à la création d'un objet
- L'objet est une référence (traité par le ramasse-miettes, destruction explicite inutile)

CLASSES

II-3- implémentation d'une classe

Soit la classe personne ci- contre :

- class est un mot clé permettant de définir la structure
- Personne est le nom de la classe ici
- """Classe Personne""" sert à documenter la classes
- Regarder le rôle des « : » et des indentations
- __init__ est une méthode standard appelée constructeur, automatiquement appelée lors de l'instanciation
- self représente l'instance elle-même, elle doit apparaître en première position dans la définition de toutes les méthodes, mais il ne sera pas nécessaire de la spécifier lors de l'appel
- nous exploitons le constructeur pour énumérer les champs de la classe (pas obligatoire, mais bonne habitude à prendre), ils sont non typés

```
class Personne:
    """Classe Personne"""
    #constructeur
    def __init__(self):
        #lister les champs
        self.nom = ""
        self.age = 0
        self.salaire = 0.0
    #fin constructeur
    #saisie des infos
    def saisie(self):
        self.nom = input("Nom : ")
        self.age = int(input("Age : "))
        self.salaire = float(input("Salaire : "))
    #fin saisie
#fin definition
```

CLASSES

II-3- implémentation d'une classe

Soit la classe personne ci- contre : (suite)

- le constructeur peut prendre des paramètres en entrée. Ex. initialisation des champs
- contrairement aux autres langages, un seul constructeur par classe seulement
- noter le rôle du « . » dans l'accès aux champs

II-4- Instanciation et utilisation dans le programme principal

```
### Programme principal

#instanciation
p = Personne()
affiche tous les membres de p
print(dir(p))
#affectation aux champs
p.nom = input("Nom : ")
p.age = int(input("Age : "))
p.salaire = float(input("Salaire : "))
affichage
print(p.nom,", ",p.age,", ",p.salaire)
```

CLASSES

II-4- Instanciation et utilisation dans le programme principal

- Pour la création de l'instance p, nous spécifions le module puis le nom de la classe
- Il y a () parce que c'est bien une méthode que nous appelons à la création : le constructeur
- Le paramètre self du constructeur n'est pas à spécifier sur l'instance
- Noter le rôle de « . » lors de l'accès aux champs de l'instance
- Les champs sont accessibles en lecture / écriture
- L'accès direct aux champs n'est pas conseillé en programmation objet

NB: (sous Python, pour rendre un attribut privé, il faut mettre un double _ devant le nom du champ et créer alors des *accesseurs* et des mutateurs, à voir en TP)

II-5- Héritage

Idée : L'héritage permet de construire une hiérarchie de classes. Les classes héritières héritent des champs et méthodes de la classe ancêtre.

Ce mécanisme nécessite des efforts de modélisation et de conception. Mais au final, on améliore la lisibilité et la réutilisabilité du code.

CLASSES

II-5- Héritage (suite)

Idée : L'héritage permet de construire une hiérarchie de classes. Les classes héritières héritent des champs et méthodes de la classe ancêtre.

Ce mécanisme nécessite des efforts de modélisation et de conception. Mais au final, on améliore la lisibilité et la réutilisabilité du code.

Soit la classe Employe est une Personne,

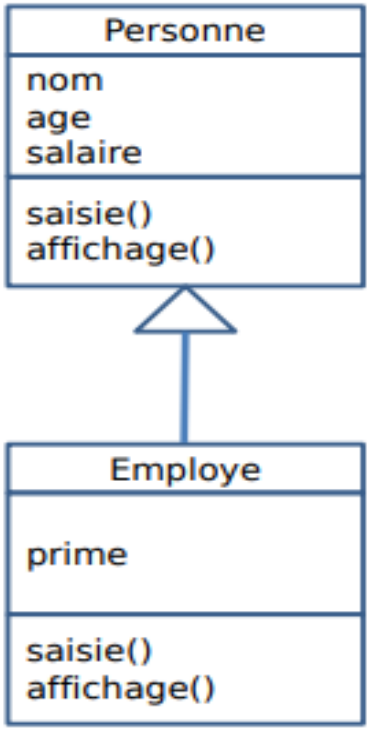
avec le champ supplémentaire prime.

Cela va nécessiter la reprogrammation

des méthodes saisie() et affichage().

On peut éventuellement ajouter d'autres

méthodes spécifiques à Employe.



Déclaration en Python

```
class Employe (Personne):
    ...
```

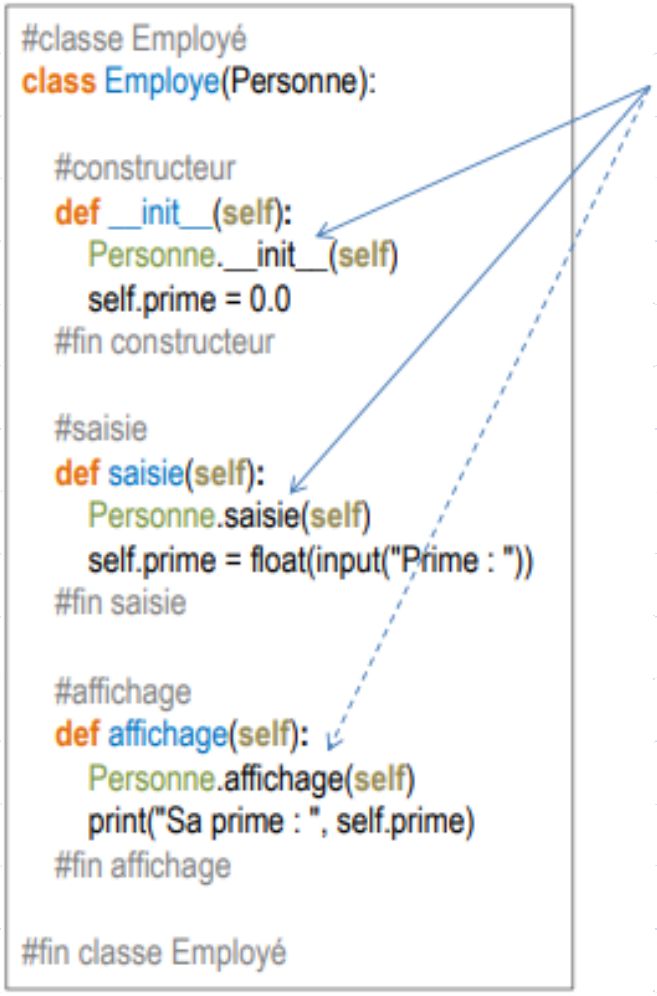
CLASSES

II-6- Surcharge des méthodes

La surcharge de des méthodes consiste à

Réécriture des méthodes des la classe ancêtre /mère (Personne) Lors de l'implémentation de la classe fille (Employer).

Noter comment sont réutilisées les méthodes programmées dans la classe ancêtre Personne



```
#classe Employé
class Employe(Personne):

    #constructeur
    def __init__(self):
        Personne.__init__(self)
        self.prime = 0.0
    #fin constructeur

    #saisie
    def saisie(self):
        Personne.saisie(self)
        self.prime = float(input("Prime : "))
    #fin saisie

    #affichage
    def affichage(self):
        Personne.affichage(self)
        print("Sa prime : ", self.prime)
    #fin affichage

#fin classe Employé
```