




1



جامعة ابن توفيل  
UNIVERSITÉ IBN TOFÄÏL  
Faculté des Sciences – Kenitra

2019\2020

Département d'Informatique

# P00 en Python

**Master:** Master Spécialisé Génie Logiciel pour le Cloud  
**Présenté le:** 30/Octobre/2019 (S1)  
**Contact:** [khalid.ounachad@uit.ac.ma](mailto:khalid.ounachad@uit.ac.ma)

2

# POO en langage Python

- Introduction,
- Classe,
- Encapsulation,
- Heritage,
- Polymorphism et surcharge ,

3

- Introduction,

4

## Introduction

- Dans les années 1970, la programmation se développe et de nouvelles technologies logicielles apparaissent: mode multi use, interfaces graphiques,....
- Taille des programmes augment: 10000 lignes de code pour un logiciel de bonne taille dans les années 1970(et des millions de nos jours)
- La **crise du logiciel** des années 70 est provoquée par:
  - L'impossibilité de **maîtriser la sûreté** des logiciels(pannes++)
- De bonnes **techniques de développement logiciel** deviennent **nécessaire**.

5

## Introduction

- Programmation objet est inventée dans les années 1970 (Extension de la programmation procédurale)
- Idée: concevoir les programmes comme des **objets** qui se **dialoguent**.
- Fin des années 60(Norvège): Simula 64 est le premier langage à utiliser ce paradigme(mais de façon limitée)
- Smalltalk: développé par **Alan kay** 1971 et publié en 1980 est le premier vrai langage objet.
- **Python en 1991**

6

- **Class,**

7

## Classe

### Définition:Class

Une classe définit :

- une **Unité d'encapsulation** :
  - elle **regroupe la déclaration des attributs et la définition des méthodes** associées dans une même construction syntaxique
    - attributs = stockage d'information (état de l'objet).
    - méthodes = unités de calcul (sous-programmes : fonctions ou procédures).
  - C'est aussi un **espace de noms** : deux classes différentes peuvent avoir des membres de même nom
- un **TYPE** qui permet de :
  - créer des objets (la classe est un moule, une matrice) ;

Les méthodes et attributs définis sur la classe comme « unité d'encapsulation » pourront être appliqués à ses objets

8

## Classe

### Syntaxe:

```
class (superclass,...):
    donnee = valeur
    def methode(self,...):
        self.membre = valeur
```

Classe Etudiant	
Attributs :	
Nom : char[]	nom
Prenom : char[]	prenom
Age : int	a
Méthodes :	
Lecture-etudiant()	
Ecriture-etudiant()	
Calcul-age()	

```
class MaClasse:
    "Une classe simple pour exemple "
    i = 12345
    def f(self):
        return 'bon'
```

**MaClasse.i** : référence d'attribut valide ; renvoie un entier

**MaClasse.f** : référence d'attribut valide ; renvoie un objet  
fonction

9

## Classe: Objet

### Définition: Objet

-Objet : donnée en mémoire qui est le représentant d'une classe.  
• l'objet est dit **instance** de la classe .

-Un objet est caractérisé par :

**un état** : la valeur des attributs (x, y, etc.) ;

**un comportement** : les méthodes qui peuvent lui être appliquées

**une identité** : identifie de manière unique un objet (p.ex. son adresse en mémoire).

-Un objet est créé à partir d'une classe (en précisant les paramètres effectifs de son constructeur, sauf le premier, self)

-Les identités des objets sont conservées dans des variables (des noms)

10

## Classe: Attributs

### Définition: Attributs

On distingue :

- les **attributs d'instance** : spécifique d'un objet
- les **attributs de classe** : attaché à la classe et non aux objets

class A:

```
    ac = 10 # Attribut de classe
```

```
    def __init__(self, v):
```

```
        self.ai = v # ai attribut d'instance
```

11

## Classe: Méthodes

### Méthodes

-Une méthode d'instance est un sous-programme qui

exploite l'état d'un objet (en accès et/ou en modification).

- Le premier paramètre désigne nécessairement l'objet. Par convention, on l'appelle **self**.

-Une méthode de classe est une méthode qui travaille sur la classe (et non l'objet).

- Elle est décorée **@classmethod**.
- Son premier paramètre est nommé **cls** par convention.

12

## Classe: Méthodes

### Méthodes d'instance/de classe

Exemple:

```
class A:
    nb = 0

    def __init__(self):
        print("creation objet de type A")
        A.nb = A.nb + 1
        print("il y en a maintenant ", A.nb)

    @classmethod
    def get_nb(cls):
        return A.nb

print("Partie 1 : nb objets = ", A.get_nb())
a = A()
print("Partie 2 : nb objets = ", A.get_nb())
b = A()
print("Partie 3 : nb objets = ", A.get_nb())
```

13

## Classe: Méthodes

### Méthodes

-Une méthode statique est une méthode définie dans l'espace de nom de la classe mais est indépendante de cette classe.

-Elle est décorée **@staticmethod**

Exemple:

class Date:

...

**@staticmethod**

def est\_bissextile(annee):

return annee % 4 == 0 and (annee % 100 != 0  
or annee % 400 == 0)

14

## Classe: Méthodes

### Méthodes spéciales:

`__init__(self, ...)` : le constructeur : méthode d'initialisation nécessairement appelée quand on crée un objet. Voir aussi `__new__`. C'est le constructeur qui devrait définir les attributs d'instance de la classe.

`__new__(cls, ...)` : méthode implicitement statique qui crée l'objet (et appelle `__init__`).

`__del__(self)` : le destructeur, appelé quand une instance est sur le point d'être détruite.

`__bool__(self)` : utilisée quand l'objet est considéré comme

15

## Classe: Méthodes

### Méthodes spéciales:

`__bool__(self)` : utilisée quand l'objet est considéré comme booléen et avec la fonction prédéfinie `bool()`

. ... Voir: <https://docs.python.org/3/reference/datamodel.html#special-method-names>

16



- **Constructeur,**

17

## Classe: Constructeur

### Initialiseur: Constructeur

class Robot:

""" Robot qui sait avancer d'une case et pivoter à droite de 90°. Il est repéré par son abscisse x, son ordonnée y et sa direction. """

def \_\_init\_\_(self, x, y, direction): """ Initialiser le robot self à partir de sa position (x, y) et sa direction. """

self.x = x

self.y = y

self.direction = direction

18

## Classe: Constructeur

Exemple: instance

```
>>> class Complexe:
...     def __init__(self, reel, imag):
...         self.r = reel
...         self.i = imag ...
>>> x = Complexe(3.0, -4.5)
>>> x.r, x.i
3.0, -4.5
```

19

- Encapsulation,

20

## Classe: Encapsulation

**Syntaxe:** `_NomAtt` ou `__NomAtt` (de même pour les méthodes)

**Accès:**

• On utilise les getters et les setters:

- les **accesseurs** (en anglais accessor) qui fournissent des informations relatives à l'état d'un objet, c'est-à-dire aux valeurs de certains de ses attributs (généralement privés) sans les modifier ;
- les **mutateurs** (en anglais mutator) qui modifient l'état d'un objet, donc les valeurs de certains de ses attributs.

On rencontre souvent l'utilisation de noms de la forme

`get_XXXX()` pour les accesseurs et `set_XXXX()` pour les mutateurs, y

21

## Classe: Encapsulation

**Exemple:Accès**

```
class Point:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y

    def set_x(self, x):
        self.__x = x

    def set_y(self, y):
        self.__y = y

a = Point(3, 7)
print("a : abscisse =", a.get_x())
print("a : ordonnee =", a.get_y())
a.set_x(6)
a.set_y(10)
print("a : abscisse =", a.get_x())
print("a : ordonnee =", a.get_y())
```

22

## Classe: Encapsulation

### Exemple2: Autre façon

```
class Date:
    def __init__(self, jour, mois, annee):
        self.__jour = jour
        self.__mois = mois
        self.__annee = annee

    @property
    # accès en lecture à mois, comme si c'était un attribut
    def mois(self):
        return self.__mois

    @mois.setter
    # accès en écriture à mois, comme si c'était un attribut
    def mois(self, mois):
        if mois < 1 or mois > 12:
            raise ValueError
        self.__mois = mois
```

23

- Utilisation,

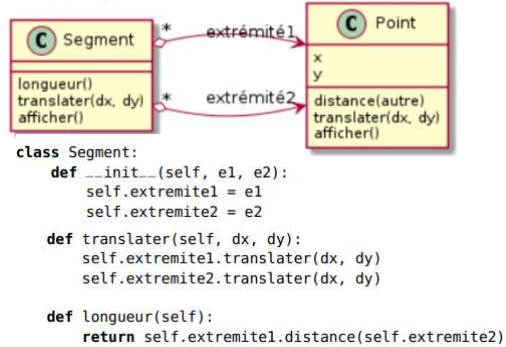
24

## Classe: relation d'utilisation

Exemple où:

Une classe utilise une autre classe (en général, ses méthodes)

```
import math
class Point:
    def __init__(self, x=0, y=0):
        self.x = float(x)
        self.y = float(y)
    def translater(self, dx, dy):
        self.x += dx
        self.y += dy
    def distance(self, autre):
        dx2 = (self.x - autre.x) ** 2
        dy2 = (self.y - autre.y) ** 2
        return math.sqrt(dx2 + dy2)
```



25

## Classe: relation d'utilisation

Exemple:

```
def exemple():
    # créer les points sommets du triangle
    p1 = Point(3, 2)
    p2 = Point(6, 9)
    p3 = Point(11, 4)

    # créer les trois segments
    s12 = Segment(p1, p2)
    s23 = Segment(p2, p3)
    s31 = Segment(p3, p1)

    # créer le barycentre
    sx = (p1.x + p2.x + p3.x) / 3.0
    sy = (p1.y + p2.y + p3.y) / 3.0
    barycentre = Point(sx, sy)
```

26

- Héritage,

27

## Classe: Héritage

**Principe:** définir une nouvelle classe par spécialisation d'une (ou plusieurs) classes existantes.

### Exemple:

Définir une classe PointNommé sachant que la classe Point existe. La classe

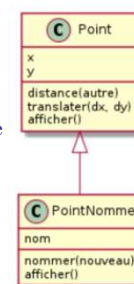
PointNommé :

hérite (récupère) tous les éléments de la classe Point

ajoute un nom et les opérations pour manipuler le nom

redéfinit la méthode afficher (pour afficher le nom et les coordonnées du point)

Point est la **super-classe**, PointNommé la **sous-classe**



28

## Classe: Héritage

### Exemple:

```
class PointNomme(Point):    # La classe PointNommé hérite de Point
    def __init__(self, nom, x=0, y=0):
        super().__init__(x, y) # initialiser la partie Point du PointNommé
        self.nom = nom         # un nouvel attribut

    def nommer(self, nouveau_nom): # une nouvelle méthode
        self.nom = nouveau_nom
```

**super()** est recommandé pour appeler la méthode des super classes. ici

Ancienne solution : **Point.\_\_init\_\_(self, x, y)**

La classe **object** C'est l'ancêtre commun à toutes les classes.

Quand on ne précise aucune superclasse, la classe hérite implicitement de object

29

## Classe: Héritage Multiple

Il suffit de lister toutes les super classes séparées par des virgules

**class S(A, B, C):** # la classe S hérite de A, B et C

```
class carnivore:
    def __init__(self,p):
        self._poidsViande = p
    def devorer(self):
        print("Je mange",self._poidsViande,"kilogs de steack par jour")

class herbivore:
    def __init__(self,p):
        self._poidsHerbe = p
    def brouter(self):
        print("Je mange",self._poidsHerbe,"kilogs de gazon par jour")

class omnivore(carnivore,herbivore):
    def __init__(self,pv,ph,h):
        carnivore.__init__(self,pv)
        herbivore.__init__(self,ph)
        self._humain = h

teddy = omnivore(10,5,False)
teddy.devorer()
teddy.brouter()

Je mange 10 kilogs de steack par jour
Je mange 5 kilogs de gazon par jour
```

30

- polymorphisme,

31

## Classe: Polymorphisme

**Rappel:** mécanisme qui permet à une classe fille de redéfinir une méthode dont elle a hérité de sa classe mère, tout en gardant la même signature.

```
class rectangle:
    def __init__(self,x,y):
        self._x = x
        self._y = y
    def surface(self):
        return self._x*self._y

class paveDroit(rectangle):
    def __init__(self,x,y,z):
        super().__init__(x,y)
        self._z = z
    def surface(self):
        return 2*(self._x*self._y+self._x*self._z+self._y*self._z)

photo = rectangle(3,4)
print(photo.surface())
weston = paveDroit(3,4,10)
print(weston.surface())
```

Console

```
12
164
```

32



- **Surcharge des opérateurs**

33

## Classe: Surcharge

**Rappel: Surcharge des opérateurs:** Lorsque l'on crée des classes, il est naturel de vouloir adapter les opérateurs usuels afin de pouvoir les appliquer sur des objets de nos propres classes.

Cette création de nouvelles versions s'appelle une **surcharge**.

```
class rectangle:
    def __init__(self,x,y):
        self.__x = x
        self.__y = y
    def surface(self):
        return self.__x*self.__y
    def __mul__(self, other):
        return rectangle(self.__x*other.__x,self.__y*other.__y)
    def __imul__(self, other):
        self = self*other
        return self

a = rectangle(3,4)
b = rectangle(2,5)
c = a*b
print(c.surface())
c *= b
print(c.surface())
```

Console

```
120
1200
```

**Surcharge des opérateurs \* et \*= dans la classe "rectangle"**

On définit ici le résultat de la multiplication de deux rectangles comme étant le rectangle dont la largeur (resp. longueur) est le produit des largeurs (resp. longueurs) :

34

جامعة ابن توفيل  
UNIVERSITÉ IBN TOFAÏL  
Faculté des Sciences

MASTER D'UNIVERSITÉ SPÉCIALISÉ  
... Informatique ...

**GÉNIE LOGICIEL POUR  
LE CLOUD**

Exercez les métiers de demain

Business Intelligence

BIG DATA

Big Data

Cloud Computing

Coordinateur  
Pr. Moulay Youssef Hadi  
Département d'informatique  
hadiyoussef@gmail.com 0694 500 534

35

جامعة ابن توفيل  
UNIVERSITÉ IBN TOFAÏL  
Faculté des Sciences – Kenitra

2019\2020

Département d'Informatique

**Collections d'objets**

Master: Master Spécialisé Génie Logiciel pour le Cloud  
Présenté le: 30/Octobre/2019 (S1)  
Contact: khalid.ounachad@uit.ac.ma

36

## Collections d'objets

- Liste,
- Tuple,
- dictionnaire,
- Ensemble.

37

## List

### Exemples

- `[1,2,3,4,5,6]` # e n t i e r s
- `[3.14, 0.693, 2.78]` # f l o t t a n t s
- `[True, True, False]` # b o o l é e n s
- `['a', 'bra', 'ca', 'da', 'bra']` # c h a î n e s d e a r a t è r e s
- `[1, 1.609, 'bonjour', None]`
- `range(10), range(2, 10, 2)`

38

## List

### Listes vides

- `L=[]`
- `L1=list()`

**Longueur:** Le nombre d'éléments d'une liste est la longueur de la liste. La longueur de la liste **L** est donnée par **len(L)**.

- Pour une liste de longueur **N**, les indices valides varient entre  $-N$  et  $(N - 1)$  au sens large .
- L'utilisation d'un indice non valide provoque une erreur.

IndexError: list index out of range

39

## List

### Accès à un élément et modification

On considère une liste **L** de longueur **N**.

- L'indice du premier élément est nul.
- L'indice du dernier élément est égal à  $(N - 1)$ .
- L'élément d'indice **k** est **L[k]**.
- **L[i]=v** # modification du ième par v

### Exemples:

```
jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
print(jour[2])
mercredi
jour[-3]=200;print(jour[-3])
200
```

40

## List

### Sous-listes:

On suppose que  $i$  et  $j$  sont positifs.

- La tranche  $L[i:j]$  est la sous-liste

$$(L_k)_{i \leq k < j} \quad [L[i], L[i+1], \dots, L[j-1]]$$

- La tranche  $L[i:]$  est la sous-liste

$$(L_k)_{i \leq k} = (L_k)_{i \leq k < N}$$

- La tranche  $L[:j]$  est la sous-liste

$$(L_k)_{k < j} = (L_k)_{0 \leq k < j}$$

- La tranche  $L[i:j:p]$  est la sous-liste de la tranche  $L[i:j]$  constituée des éléments  $L[k]$  pour lesquels l'indice  $k$

vérifie  $k = i \bmod p$ .

- En particulier,  $L[::-1]$  est une copie en miroir de  $L$ .

41

## List

### Concaténation

```
>>> sept_zeros = [0]*7; sept_zeros
```

```
[0, 0, 0, 0, 0, 0, 0]
```

```
>>> L1, L2 = [1, 2, 3], [4, 5]
```

```
>>> L1 + L2
```

```
[1, 2, 3, 4, 5]
```

### Opérations algébriques

On suppose que les éléments de la liste  $L$  sont comparables : nombres ou chaînes de caractères.

**min(L).**

**max(L).**

**sum(L).** Si les éléments de la liste  $L$  sont des nombres, on obtient la somme de ces nombres avec l'instruction

42

## List

### Copie d'une liste **ATTENTION !**

```
>>> L = ['Dans', 'python', 'tout', 'est', 'objet']
>>> T = L
>>> T[4] = 'bon'
>>> T ['Dans', 'python', 'tout', 'est', 'bon']
>>> L ['Dans', 'python', 'tout', 'est', 'bon']
>>> L = T[:]
>>> L[4] = 'objet'
>>> T; L
['Dans', 'python', 'tout', 'est', 'bon'] ['Dans', 'python', 'tout', 'est', 'objet']
```

### Correction:

```
>>> from copy import copy; T = copy( L )#copie superficielle
>>> L1= [[1, 2, 3], [4, 5]]
>>> L2 = copy(L1)# même problème qu'avant
>>> from copy import deepcopy; T =deepcopy( L )#copie profonde
```

43

## List

### Une liste L a ses propres méthodes

len(L) : taille de la liste

L.sort : trier la liste L

L.append : ajout d'un élément à la fin de la liste L

L.reverse : inverser la liste L

L.index : rechercher un élément dans la liste L

L.remove : retirer un élément de la liste L

L.pop : retirer le dernier élément de la liste L

L.pop(i) : retirer le ième élément de la liste L

del(L[i]): supprime le ième élément de L

L.count : compter les occurrences dans la liste L

La commande sorted(L) renvoie une liste triée par ordre croissant

Voir (help(list)) ou dir(list) pour autres méthodes..

44

# Tuple

## Initialisation

- (),
- tuple(),
- (2,), 'a', 'b', 'c', 'd',
- ('a', 'b', 'c', 'd')

## Concaténation

```
>>> (1, 2)*3
(1, 2, 1, 2, 1, 2)
>>> t1, t2 = (1, 2, 3), (4, 5)
>>> t1 + t2
(1,2,3,4,5)
```

45

# Tuple

## • Modification

- un tuple **n'est pas modifiable**

```
>>> t = 'a', 'b', 'c', 'd'
>>> t[0] = 'master'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>> t = ('master',) + t[1:]
>>> t
('master', 'b', 'c', 'd')
```

46

## Tuple

```
>>> t = (1, 2, [3, 4], 6)
```

```
>>> t[2][0] = 1
```

```
>>> t
```

```
(1, 2, [1, 4], 6)
```

Un tuple `t` a ses propres méthodes

(`help(tuple)`)

- Sont presque les mêmes que pour les listes (sauf modification)

47

## dict

- Un tuple `t` a ses propres méthodes

(`help(tuple)`)

- Sont presque les mêmes que pour les listes (sauf modification)

48



## dict

### Initialisation

```
{}, dict(),
>>>dico={'point': 1, 'ligne': 2, 'triangle': 3}
>>> dico['quad'] = 4
>>> dico
{'quad': 4, 'ligne': 2, 'triangle': 3, 'point': 1}
>>> dico['point'] = 3;dico
{'quad': 4, 'ligne': 2, 'triangle': 3, 'point': 3}
```

49

## dict

### Attention!:Remarques

- un dictionnaire n'est pas une séquence
- un dictionnaire est constitué de clés et de valeurs
- on ne peut pas concaténer un dictionnaire avec un autre

50

## dict

### Copie d'un dictionnaire

```
>>> dico = {'computer':'ordinateur', 'mouse':'souris',
'keyboard':'clavier'}
>>> dico2 = dico
>>> dico3 = dico.copy()
>>> dico2['printer'] = 'imprimante'
>>> dico2
{'computer': 'ordinateur', 'mouse': 'souris', 'printer': 'imprimante',
'keyboard': 'clavier'}
>>> dico
{'computer': 'ordinateur', 'mouse': 'souris', 'printer': 'imprimante',
'keyboard': 'clavier'}
>>> dico3
{'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}
```

51

## dict

### Un dictionnaire a ses propres méthodes (help(dict))

- **len(dico)** : taille du dictionnaire
- **dico.keys** : renvoie les clés du dictionnaire sous forme de liste
- **dico.values** : renvoie les valeurs du dictionnaire sous forme de liste
- **dico.has\_key** : renvoie True si la clé existe, False sinon
- **dico.get** : donne la valeur de la clé si elle existe, sinon une valeur par défaut

52

## dict

### Tableaux associatifs

Un tableau associatif (dictionnaire) est une collection modifiable de couples <clé non modifiable, valeur modifiable> permettant un accès à la valeur si on fournit la clé. On peut le voir comme une liste dans laquelle l'accès à un élément se fait par un code au lieu d'un indice. L'accès à un élément est optimisé en Python.

#### Création de nouveaux tableaux associatifs

avec `A = {'a': 1, 'b': 2, 'c': 3, 'd': 4}`

Description	Opération	exemple	résultat
tableau associatif vide	<code>{}</code>	<code>{}</code>	<code>{}</code>
	<code>dict</code>	<code>dict()</code>	<code>{}</code>
en extension	<code>{...}</code>	<code>{'a': 1, 'b': 2, 'c': 3}</code>	<code>{'c': 3, 'a': 1, 'b': 2}</code>
par itérable	<code>dict</code>	<code>dict([('a', 1), ('b', 2)])</code>	<code>{'a': 1, 'b': 2}</code>
nommé	<code>dict</code>	<code>dict(a=1, b=2, c=3)</code>	<code>{'c': 3, 'a': 1, 'b': 2}</code>
par copie superficielle	<code>copy</code>	<code>A.copy()</code>	<code>{'d': 4, 'c': 3, 'a': 1, 'b': 2}</code>
à partir d'un dict	<code>fromkeys</code>	<code>dict.fromkeys(A, 0)</code>	<code>{'d': 0, 'c': 0, 'a': 0, 'b': 0}</code>

53

## dict

#### Informations/consultations

avec `A = {'a': 1, 'b': 2, 'c': 3, 'd': 4}` et `B = {'x': 24, 'y': 25, 'z': 26}`

Description	Opération	exemple	résultat
cardinal	<code>len</code>	<code>len(A)</code>	<code>4</code>
appartenance de clé	<code>in</code>	<code>'c' in A</code>	<code>True</code>
égalité	<code>==</code>	<code>A == B</code>	<code>False</code>
inégalité	<code>==</code>	<code>A != B</code>	<code>True</code>
accès à une valeur (erreur si absent)	<code>[]</code>	<code>A['b']</code>	<code>2</code>
accès à une valeur (défaut si absent)	<code>get</code>	<code>A.get('a', 0); A.get('z', 17)</code>	<code>1 17</code>
itérateur sur les couples	<code>items</code>	<code>[x for x in A.items()]</code>	<code>[('a', 1), ('c', 3), ('d', 4), ('b', 2)]</code>
itérateur sur les clés	<code>keys</code>	<code>[x for x in A.keys()]</code>	<code>['a', 'c', 'd', 'b']</code>
itérateurs sur les valeurs	<code>values</code>	<code>[x for x in A.values()]</code>	<code>[1, 3, 4, 2]</code>

54

## dict

### Modifications

avec `A = {'a': 1, 'b': 2, 'c': 3, 'd': 4}`

Description	Opération	exemple	résultat
vidage	<code>clear</code>	<code>A.clear(); A</code>	<code>{}</code>
suppression	<code>del</code>	<code>del A['c']; A</code>	<code>{'d': 4, 'b': 2, 'a': 1}</code>
modification d'une valeur	<code>[]</code>	<code>A['b']={'r', 'v', 'b'}; A</code>	<code>{'d': 4, 'c': 3, 'b': {'r', 'b', 'v'}, 'a': 1}</code>
retrait et valeur	<code>pop</code>	<code>A.pop('a', 9); A.pop('a', 9); A</code>	<code>1 9 {'d': 4, 'c': 3, 'b': 2}</code>
retrait et couple	<code>popitem</code>	<code>A.popitem(); A</code>	<code>('d', 4) {'c': 3, 'b': 2, 'a': 1}</code>
accès et/ou modification d'une valeur	<code>setdefault</code>	<code>A.setdefault('b', 9); A.setdefault('z', 9); A</code>	<code>2 9 {'z': 9, 'd': 4, 'c': 3, 'b': 2, 'a': 1}</code>
modification plurielle	<code>update</code>	<code>A.update({'a': 7}, {'x': 9}, {'y': 8}); A</code>	<code>{'y': 8, 'd': 4, 'b': 2, 'x': 9, 'c': 3, 'a': 7}</code>

55

## set

### Création de nouveaux ensembles

avec `A = {0, 1, 2, 3, 4}` et `B = {3, 4, 5, 6}`

Description	Opération	exemple	résultat
ensemble vide	<code>set</code>	<code>set()</code>	<code>set()</code>
ensemble non vide	<code>{...}</code>	<code>{1, 2, 3, 4}</code>	<code>{1, 2, 3, 4}</code>
itérable	<code>set</code>	<code>set("ESIEE PARIS")</code>	<code>{'I', 'A', 'P', 'S', 'E', ' ', 'R'}</code>
copie superficielle	<code>copy</code>	<code>A.copy()</code>	<code>{0, 1, 2, 3, 4}</code>
intersection	<code>&amp;</code>	<code>A &amp; B</code>	<code>{3, 4}</code>
	<code>intersection</code>	<code>A.intersection(B)</code>	<code>{3, 4}</code>
union	<code> </code>	<code>A   B</code>	<code>{0, 1, 2, 3, 4, 5, 6}</code>
	<code>union</code>	<code>A.union(B)</code>	<code>{0, 1, 2, 3, 4, 5, 6}</code>
différence	<code>-</code>	<code>A - B</code>	<code>{0, 1, 2}</code>
	<code>difference</code>	<code>A.difference(B)</code>	<code>{0, 1, 2}</code>
différence symétrique	<code>^</code>	<code>A ^ B</code>	<code>{0, 1, 2, 5, 6}</code>
	<code>symmetric_difference</code>	<code>A.symmetric_difference(B)</code>	<code>{0, 1, 2, 5, 6}</code>

56

## set

### Informations/consultations

avec  $A = \{0, 1, 2, 3, 4\}$  et  $B = \{3, 4, 5, 6\}$

Description	Opération	exemple	résultat
cardinal	<code>len</code>	<code>len(A)</code>	5
appartenance	<code>in</code>	<code>1 in A</code>	True
inclusion	<code>&lt;=</code>	<code>{1, 2} &lt;= A</code>	True
	<code>issubset</code>	<code>A.issubset(A)</code>	True
inclusion stricte	<code>&lt;</code>	<code>A &lt; A</code>	False
contenance	<code>&gt;=</code>	<code>A &gt;= {1, 2}</code>	True
	<code>issuperset</code>	<code>A.issuperset(A)</code>	True
contenance stricte	<code>&gt;</code>	<code>A &gt; {1, 2}</code>	True
égalité	<code>==</code>	<code>A == B</code>	False
inégalité	<code>!=</code>	<code>A != B</code>	True
intersection vide	<code>isdisjoint</code>	<code>A.isdisjoint(B)</code>	False

57

## set

### Modification d'un ensemble

avec  $A = \{0, 1, 2, 3, 4\}$  et  $B = \{3, 4, 5, 6\}$

Description	Opération	exemple	résultat
ajout	<code>add</code>	<code>A.add(2014); A</code>	$\{0, 1, 2, 3, 4, 2014\}$
vidage	<code>clear</code>	<code>A.clear(); A</code>	<code>set()</code>
extraction (retourne et supprime un élément)	<code>pop</code>	<code>A.pop(); A</code>	$\{0, 1, 2, 3, 4\}$
suppression (erreur si absent)	<code>remove</code>	<code>A.remove(2); A</code>	$\{0, 1, 3, 4\}$
suppression (sans effet si absent)	<code>discard</code>	<code>A.discard(4); A</code>	$\{0, 1, 2, 3\}$
union	<code> =</code>	<code>A  = B; A</code>	$\{0, 1, 2, 3, 4, 5, 6\}$
	<code>update</code>	<code>A.update(B); A</code>	$\{0, 1, 2, 3, 4, 5, 6\}$
intersection	<code>&amp;=</code>	<code>A &amp;= B; A</code>	$\{3, 4\}$
	<code>intersection.update</code>	<code>A.intersection.update(B); A</code>	$\{3, 4\}$
différence symétrique	<code>^=</code>	<code>A ^= B; A</code>	$\{0, 1, 2, 5, 6\}$
	<code>symmetric.difference.update</code>	<code>A.symmetric.difference.update(B); A</code>	$\{0, 1, 2, 5, 6\}$
différence	<code>-=</code>	<code>A -= B; A</code>	$\{0, 1, 2\}$
	<code>difference.update</code>	<code>A.difference.update(B); A</code>	$\{0, 1, 2\}$

58

**MERCI**

59

## Travaux Pratiques:

- TP2

60