

# Le Langage Python

*expliciter ses pensées, ranger son travail*

Alain Bidault

bidault.enseignement@neuf.fr

Principes

Les instructions de base

Variable, typage, et les types de bases

Arithmétique et opérations logiques

Héritage

Les Exceptions

Modules

Bibliothèques

Objets – Classes – Attributs - Encapsulation

***NE PAS IMPRIMER SVP***

## *Pourquoi Python ?*



Syntaxe simple

Types de données évolués

Extensible

Interprété

Typage dynamique (on peut se concentrer sur le reste)

En pleine explosion

Programmation Objet

Bibliothèques

## *Le Langage Python 3.xx*

L'indentation est porteuse de sens en Python

Les commentaires débutent par #

Gadgets :

On peut séparer deux instructions sur une même ligne par ;

On peut découper une instruction sur plusieurs lignes en terminant les premières lignes par \

## Le Langage Python

Contenu d'un fichier exemple.py

```
note = (3+14+17)//3 ; note2 = (3+14+17)/3.0
```

```
nom = "Raoul Gravier"
```

```
print( "bonjour ", nom, ", ta moyenne est de ", note)
```

bonjour Raoul Gravier, ta moyenne est de 11

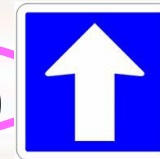
```
print( "bonjour "+ nom+ ", ta moyenne est de "+ \
      str(note2))
```

## Exemple de fonction

Deux paramètres déclarés

```
def quelleChaussure(tempsFroid, sortiePrevue) :  
    ....
```

```
print( quelleChaussure(True, True))
```



Deux arguments envoyés

## Exemple de fonction

```
def quelleChaussure(tempsFroid,sortiePrevue) :  
    if tempsFroid :  
        if sortiePrevue :  
            chaussure = "souliers 100% cuir"  
        else :  
            chaussure = "charentaises"  
    else :  
        chaussure = "sabots plastiques"  
    return chaussure
```



```
print( quelleChaussure(True,True))
```



souliers 100% cuir



## Autre exécution de la fonction

```
def quelleChaussure(tempsFroid,sortiePrevue) :  
    if tempsFroid :  
        if sortiePrevue :  
            chaussure = "souliers 100% cuir"  
        else :  
            chaussure = "charentaises"  
    else :  
        chaussure = "sabots plastiques"  
    return chaussure
```

```
print quelleChaussure(True,False)
```

charentaises



## *Importance de l'indentation : les blocs*

```
def quelleChaussure(tempsFroid,sortiePrevue) :
```

```
    if tempsFroid :
```

```
        if sortiePrevue :
```

```
            chaussure = "souliers 100% cuir"
```

```
        else :
```

```
            chaussure = "charentaises"
```

```
    else :
```

```
        chaussure = "sabots plastiques"
```

```
    return chaussure
```



## Les procédures et fonctions



```
def factoriel(n):  
    if (n==1) or (n==0):  
        f = 1  
    else : f = n*factoriel(n-1)  
    return f
```

On retourne f

```
def factProcedure(n) :  
    print(factoriel(n))  
  
print( factProcedure(6))
```

## *Fonction Récursive, utilisation des paramètres*

```
def factoriel(n) :  
    if (n==1) or (n==0):  
        f = 1  
    else : f = n*factoriel(n-1)  
    return f
```



```
print( factoriel(6))
```



Dans cet exemple, on transmet à la fonction une valeur entière (en l'occurrence 6), celle pour laquelle elle devra calculer 6!, puis 5, 4, 3, 2 et 1.

## Procédure et transmission des paramètres

```
def factProcedure(n) :  
    print( factoriel(n))
```

```
print( factProcedure(7))
```



```
>>> ===== RESTART =====
```

```
5040
```

```
None
```

```
>>>
```

5040 correspond à l'affichage : `print( factoriel(7))`

La valeur retournée affichée est « None »

## *For (1)*

```
def forRangeDebFin() :  
    for nombre in range(2000, 2009) :  
        print( "année = "+str(nombre))
```

```
forRangeDebFin()
```

## *For (2)*

```
def forRangeDebFinPas() :  
    for nombre in range(2000, 2009, 2) :  
        print( "année = "+str(nombre))
```

```
forRangeDebFinPas()
```

## *For (3)*

```
def forRangeEntier() :  
    for nombre in range(9) :  
        print( "année = "+str(nombre+2000))
```

```
forRangeEntier()
```

## *While (1)*

```
def whileFor() :  
    nombre = 2000  
    while nombre <=2009 :  
        print( "année = "+str(nombre))  
        nombre += 1  
        # nombre = nombre + 1
```

whileFor()

## While (2)

```
def whileClassique() :  
    nombre = 2000  
    while nombre<=2009 :  
        print( nombre)  
        if ((nombre % 2) == 0):  
            nombre -=1  
        else : nombre += 3
```

```
whileClassique()
```



***Apparté : nombres presque infinis***  
***On code dans un espace limité des ensembles infinis...***



```
def nombre(n) :  
    while (n >= 1) :  
        print(n)  
        n *= 2
```

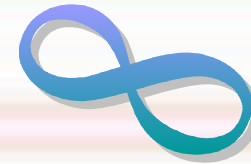
Entier : sans fin, limite = mémoire

```
nombre(1)
```

```
nombre(1E1)
```

Réel : Rapidement on n'obtient  
qu'une seule valeur : **inf**

## Apparté : notion d'infini



```
def nombre2(n):  
    while (n>0) and (n!=1E400):  
        print(n)  
        n*=2
```

Est égal à la valeur infinie

nombre2(1)

La valeur infinie n'est jamais atteinte

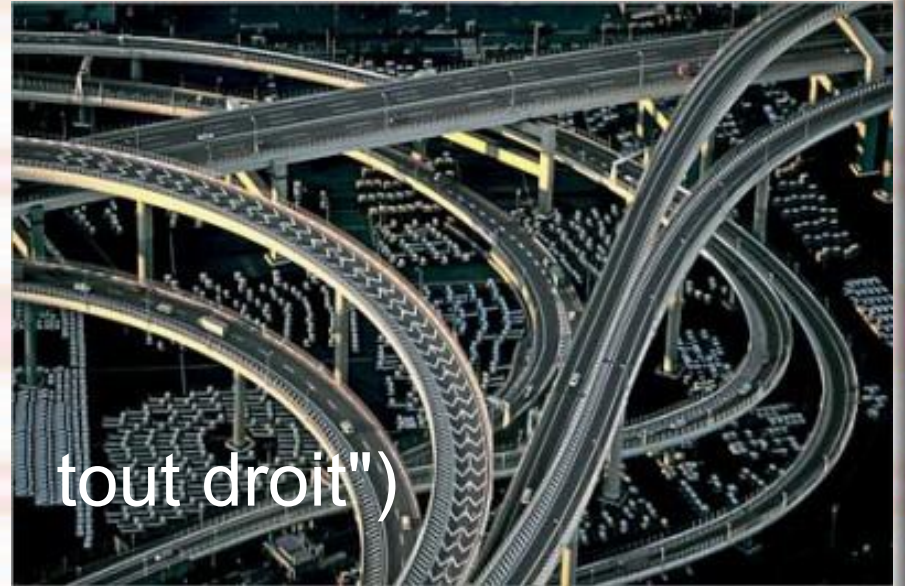
nombre2(1E0)  
nombre2(1.0)

(on obtient infini = infini)  
8.98846567431e+307

n/=2 (le zéro sera finalement atteint)  
4.94065645841e-324

## Selon le cas

```
def formeRoute(nombre) :  
    if (nombre==1) :  
        print( str(nombre) + " : tout droit")  
    elif (nombre==2) :  
        print( str(nombre) + " : 1er virage à gauche")  
    elif (nombre==5) :  
        print( str(nombre) + " : 1er virage à droite")  
    else : print( str(nombre) + " : forme de route interdite")  
  
for n in range(6) :  
    formeRoute(n)
```



## *Selon le cas*

```
def formeRoute(nombre) :  
    if (nombre==1) :  
        print( str(nombre) + " :  tout droit")  
    elif (nombre==2) :  
        print( str(nombre) + " : 1er virage à gauche")  
    elif (nombre==5) :  
        print( str(nombre) + " : 1er virage à droite")  
    else : print( str(nombre) + " : forme de route interdite")  
  
for n in range(6) :  
    formeRoute(n)
```

## Algorithme classique de permutation

```
def permut(a,b) :  
    c=a; a=b; b=c; return (a,b)
```

On passe par une troisième variable pour stocker la valeur écrasée

```
a=10; b=5
```

```
(a,b) = permut(a,b); print( a, " ", b)
```

## *Permutation magique en python grâce aux tuples*

```
def permut(a,b) :  
    c=a;  a=b;  b=c;  return (a,b)
```

```
a=10; b=5
```

```
(a,b) = (b, a)  
print( a, " ", b)
```

## **Entrée / Sortie : Saisie**

```
print( "0- plat du jour")
print( "1- entrée plat dessert")
print( "2- entrée plat")
print( "3- plat dessert")
print( "4- plat seul")
print( "5- Quitter")
choix = int(input()) ##ou raw_input()

print( choix )
```

## *Entrée / Sortie : Menu*

```
def menu(platDuJour) :  
    choixValide=False  
    while (not choixValide) :  
        print( "0- " + platDuJour)  
        print( "1- entrée plat dessert")  
        print( "2- entrée plat")  
        print( "3- plat dessert")  
        print( "4- plat seul")  
        print( "5- Quitter")  
        choix = int(input())  
        choixValide=(choix>=0 and choix <=5)  
    return choix  
  
print( menu("Canard à l'Orange"))
```



## Opérateurs divers

+ - \* /

\*\* (exposant)  $4^{**}2 = 16$

Relationnel

> >= < < > != ==

Divisions poussées (modulo)

$7/3 = 2$      $7 \% 3 = 1$     car  $7 = 3 \times 2 + 1$

Raccourcis d'écriture

nombre = 42

nombre + = 42

nombre - = 12

nombre \* = 3

nombre / = 6

## ***Bonus : opérateurs spéciaux versus codage binaire physique***

<< (ajouter des 0)

>> (supprimer les bits faibles)

& (et logique bit à bit)

| (ou logique bit à bit)

^ (ouExclusif bit à bit)

# 64	0100 0000
# 65	0100 0001
# 96	0110 0000

```
print(64<<1)
print(65<<1)
print(96<<1)
```

## ***Bonus : opérateurs spéciaux versus codage binaire physique***

<< (ajouter des 0)

>> (supprimer les bits faibles)

& (et logique bit à bit)

| (ou logique bit à bit)

^ (ouExclusif bit à bit)

# 64	0100 0000
# 65	0100 0001
# 96	0110 0000

```
print(64>>1)  
print(65>>1)  
print(96>>1)
```

## ***Bonus : opérateurs spéciaux versus codage binaire physique***

<< (ajouter des 0)

>> (supprimer les bits faibles)

& (et logique bit à bit)

| (ou logique bit à bit)

^ (ouExclusif bit à bit)

```
Print( 64&32 )  
# 64 0100 0000  
# 32 0010 0000  
# 00 0000 0000
```

```
print( 65&96 )  
# 65 0100 0001  
# 96 0110 0000  
# 64 0100 0000
```

```
print( 64|32 ) #96 0110 0000  
print( 65|96 ) #97 0110 0001
```

## ***Bonus : opérateurs spéciaux versus codage binaire physique***

<< (ajouter des 0)

>> (supprimer les bits faibles)

& (et logique bit à bit)

| (ou logique bit à bit)

^ (ouExclusif bit à bit)

```
# 32 0010 0000
```

```
# 64 0100 0000
```

```
# 96 0110 0000
```

```
# 98 0110 0010
```

```
print( 96^2)      #98
```

```
print( 96^98)     #2
```

```
print( 64^64)     #0
```

```
print( 64^32)     #96
```

## *Structures de Données en Python*

- ◆ On souhaite pouvoir manipuler des données, donc représenter des ensembles de données ordonnés ou non.
- ◆ En python, on utilise des listes améliorées qui ont, en plus, l'avantage d'être accessibles par indice.
- ◆ L'autre structure proposée est le tuple.  
Un tuple est non modifiable et ne possède pas d'indice.

## Listes vs Tableaux vs Tuples

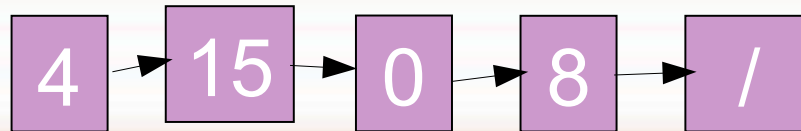
### ◆ Tableau : structure de données historique

- Accès direct par indice
- Éléments ordonnés
- Taille fixe

5	17	8	3	26	17
---	----	---	---	----	----

### ◆ Liste chaînée : structure de données historique

- Accès séquentiel
- Éléments ordonnés
- Taille variable



### ◆ Tuple

- Accès par position
- Éléments ordonnés
- Taille fixe

(dept, viande, taille, prix) = (45, lapin, 17, 23)

## Listes/Tableaux

- ◆ Définir une liste vide
  - `liste = []`
- ◆ Définir un tableau avec des valeurs
  - `liste = [12,13,14]`
- ◆ Accéder à un élément du tableau
  - `element = liste[indice]`
  - `liste[0] → 12`
  - `liste[1] → 13`
  - `liste[2] → 14`



## Listes en Python

```
liste = [12,45,6]  
liste.append(65)  
print ("liste : ", liste)  
print (liste[0])  
taille = len(liste)  
print (taille)
```

```
for i in range(taille):  
    print(" - - ",liste[i])
```

## *Menu dynamique avec listes (1)*

```
def menulteratif(listeChoix) :  
    choixValide=False  
    taille=len(listeChoix)  
    while (not choixValide) :  
        for j in range(taille) :  
            print( str(j)+"- "+listeChoix[j])  
  
        choix = int(input())  
        choixValide=(choix>=0 and choix <taille)  
    return choix
```

```
liste=["plat du jour","entrée plat dessert", \  
        "entrée plat","plat dessert","plat seul","Quitter"]  
choixeffectue = menulteratif(liste)
```

## *Menu dynamique avec listes (2)*

```
def menulteratif(choix) :  
    choixValide=False  
    taille=6  
    while (not choixValide) :  
        i=0  
        for ligne in choix :  
            print( str(i)+"- "+ligne)  
            i=i+1  
  
        choix = int(input())  
        choixValide=(choix>=0 and choix <taille)  
  
    return choix
```

## *les tuples : plusieurs valeurs structurées*

Parfois, nous avons besoin de manipuler plusieurs valeurs côte à côte :

- Un enregistrement dans une base de données (nom, prénom, adresse, date de naissance...)
- Deux valeurs complémentaires (le plus grand et le plus petit contrat)

Nous voudrions obtenir cette réponse en n'utilisant qu'une seule fonction.

## *Retourner plusieurs valeurs : les tuples*

```
def minEtMax(valeur1,valeur2,valeur3) :  
    min = valeur1; max = valeur3  
    if (valeur1>valeur2) :  
        if (valeur1>valeur3) :  
            max = valeur1;  
            if(valeur2>valeur3) : min=valeur3  
            else : min=valeur2  
        else : max=valeur3; min=valeur2  
    elif (valeur2>valeur3) :  
        max = valeur2  
        if(valeur1>valeur3) : min=valeur3  
        else : min=valeur1  
    else : max = valeur3; min=valeur1;  
    return (min, max)
```

```
(petit, grand) = minEtMax(17, 1, 150)
```

## Chaînes de caractères = tableaux

chaine1 = *"lapin"*      chaine2 = *"à la moutarde"*

**Taille** : `len(chaine1)` → 5

**Caractère** : `chaine1[2]` → 'p'

**Sous-Chaîne** :

`chaine1[2:4]` → "pi"

`chaine1[2:]` → "pin" (tout sauf les deux premiers)

`chaine1[:4]` → "lapi" (les 4 premiers)

**Concaténation** : `chaine1 + chaine2` → *"lapinà la moutarde"*

**Affectation Impossible** : `chaine1[0] = 'T'`

## Chaînes de caractères

```
rire = "Ah "*6  
print( rire )
```

```
def calculeEnvers(chaine) :  
    chaine2="";   taille = len(chaine);   fin = taille-1  
    while (fin >=0) :  
        chaine2 += chaine[fin]  
        fin-=1  
    return chaine2
```

```
palindrome = "engage le jeu que je le gagne"  
print( calculeEnvers(palindrome) )
```

## Chaînes de caractères

```
def ajouteA(chaine) :  
    reponse=""  
    for caractere in chaine :  
        if (caractere != 'r') : reponse +=caractere+"a"  
        else : reponse+=caractere  
    return reponse  
  
personnage = "brbpb"  
print(ajouteA(personnage))
```



## Chaînes de caractères

Ordre alphabétique :

```
def compare(chaine1, chaine2):  
    if (chaine1 < chaine2):  
        return "\"" + chaine1 + "\" est avant \"" + chaine2 + "\""  
    elif (chaine1 > chaine2):  
        return "\"" + chaine1 + "\" est après \"" + chaine2 + "\""  
    else : return "les chaines sont identiques !"  
  
print( compare(chaine1, personnage) )  
print( compare(palindrome, calculeEnvers(palindrome)) )  
print( compare(chaine1, chaine1) )
```

## ***Autres Formats***

### Code ASCII

```
print(str(ord('a'))+" "+str(ord('A')))  
print(str(ord('z'))+" "+str(ord('Z')))  
print(chr(97)+" "+chr(65))  
print(chr(122)+" "+chr(90))
```

Voir aussi - float("12E4")  
int("1235")

### Formatage de chaine (%s %d %f)

```
note = (3+14+17)/3  
nom = "Emilien Pigeon"  
print("bonjour %s, ta moyenne est de %s" % (nom, note))
```

## Module String

Le module string (*import string 2.7*)

```
lower=chaine1.lower()
```

```
upper=string.upper(chaine2)
```

Découpage de chaînes

```
phrase = "les trois cartes poires"
```

```
print( phrase )
```

```
decoupe = phrase.split()
```

```
print( decoupe )
```

```
decoupe = phrase.split("t")
```

```
print( decoupe )
```

Voir `capitalize()` – majuscule  
`swapcase()` – inverse

## Module String

Jointure de chaînes

```
decoupe = ['les', 'trois', 'cartes', 'poires']  
print( " ".join(decoupe) )  
decoupe = ['les ', 'rois car', 'es poires']  
print( "t".join(decoupe) )
```

Trouver/compter une sous-chaine

```
print( phrase.find("s") )  
  
print( phrase.find("tes") )  
  
print( phrase.count("s ") )
```

## Module String

Nettoyage

```
phrase = " les trois cartes poires "  
print( "#" + phrase + "#" )  
print( "#" + phrase.strip() + "#" )
```

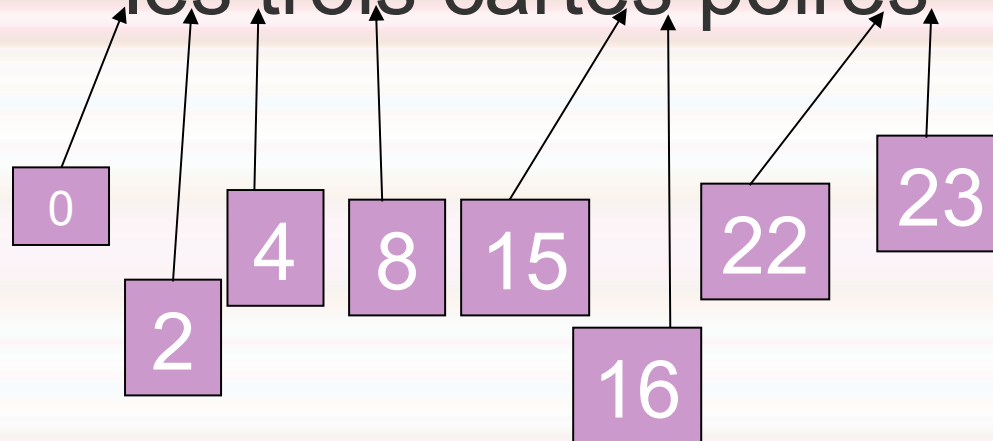
Remplacement de caractères

```
print( phrase.replace("o","a") )
```

## Module String

### Occurrences

phrase = "les trois cartes poires"



```
print( phrase.index("s") )  
print( phrase.index("s",4) )  
print( phrase.index("s",16) )
```

## *Dictionnaires*

Dictionnaire vs tableau/liste

L'index est un objet quelconque.

$d = \{cle1:valeur1, cle2:valeur2, cleN:valeurN\}.$

$d[cle] = valeur.$

## ***Dictionnaires (parcours)***

keys() pour obtenir les clés

values() pour les valeurs

items() pour les couples clé-valeur.

if clé in dico : teste si la clé est déjà présente  
(versions précédentes : *has\_key(clé)* )



## *Dictionnaires*

`del d[entree]` → supprimer

`pop d[entree]` → supprimer et retourner

## *Dictionnaires (capture de paramètres)*

```
def affiche(*mots):  
    for mot in mots:  
        print( mot+"-",end=' ' )  
    print( "" )
```

```
def affiche2(**mots):  
    print( mots )  
    print( "" )
```

```
affiche("toto","tata", "titi", "tutu")  
print( "toto","tata", "titi", "tutu")  
affiche2(a="toto",b="tata",c="titi",d="tutu")
```

## *Fonction Lambda*

Définir sur une seule ligne une fonction synthétique

Permet une programmation fonctionnelle ( $\lambda$ -calcul)

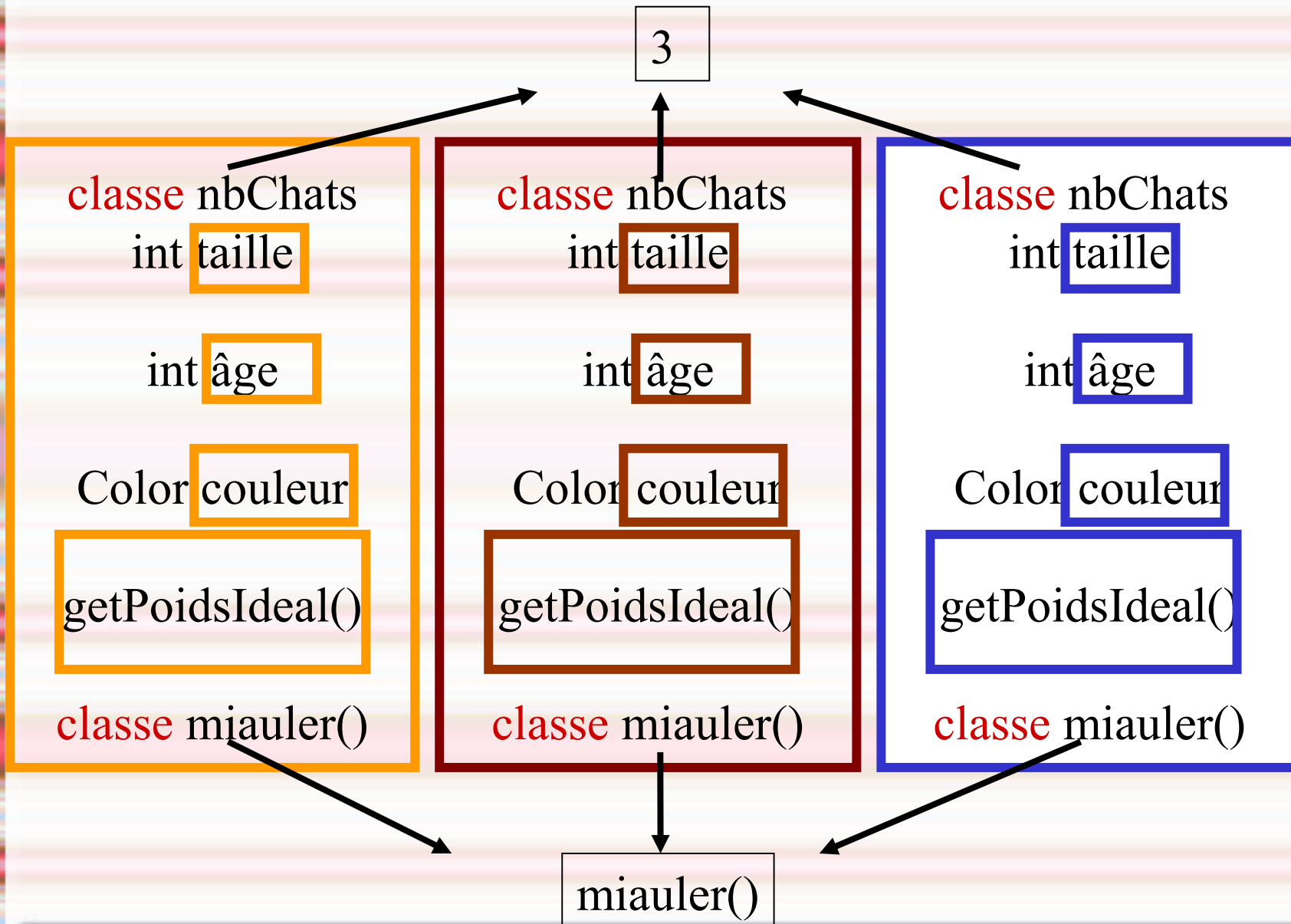
```
f = lambda x: x * x  
print( f(4))  
p = lambda x, y: x + y  
print( p(4,8))
```

## *Python Avancé*

### *La programmation Objet*

- ◆ Ranger son code
- ◆ Passerelle entre les classes des diagrammes UML (autrefois également appelées entités) et les objets exécutables (présents dans les fichiers class).
- ◆ Décrire ce qui est manipulé, compartimenter et organiser le code, factoriser le code commun.

## La programmation Objet



## *La programmation Objet (self, attributs et constructeurs)*

```
class Chat:  
    def __init__(self):  
        self.taille=35  
        self.age=1  
        self.couleur="rouge"  
  
minou = Chat()  
print( minou.age, minou.couleur)
```

nbChats

int **taille**

int **âge**

Color **couleur**

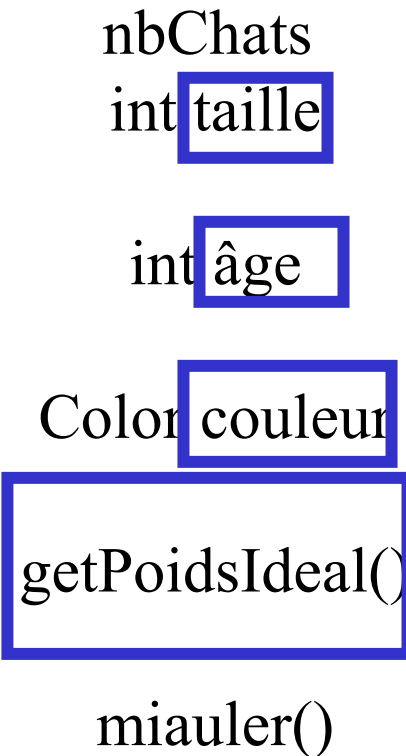
**getPoidsIdeal()**

miauler()

## *La programmation Objet (variables de classe et constructeurs)*

```
class Chat:
    nbChats=0
    def __init__(self, age=1):
        ...
        self.age=age
        Chat.nbChats+=1

minou = Chat()
print( minou.age, Chat.nbChats)
garfield = Chat(2)
print( garfield.age, Chat.nbChats)
```



A diagram enclosed in a blue rectangular border. It lists the attributes and methods of the Chat class. The attributes are: nbChats, int taille, and int âge. The methods are: Color couleur, getPoidsIdeal(), and miauler().

nbChats  
int taille  
int âge  
Color couleur  
getPoidsIdeal()  
miauler()

- ◆ NB : un seul constructeur avec valeurs par défaut
- ◆ \*\*paramètres pour la surcharge éventuelle.

## *La programmation Objet (autres méthodes)*

```
def getPoidsIdeal(self):  
    return self.taille*0.87+(self.age/2)
```

```
minou = Chat(4)  
print( minou.getPoidsIdeal())  
print( Chat.getPoidsIdeal(minou))  
garfield = Chat(2)  
print( garfield.getPoidsIdeal())
```

nbChats

int **taille**

int **âge**

Color **couleur**

**getPoidsIdeal()**

miauler()



## *La programmation Objet (self et docstring)*

- ◆ self est une convention (comme this) et est explicite.
- ◆ help(NomClasse) vous informe sur la classe et exploite les docstring :

```
class Chat:
    """Cette classe decrit des chats"""

    def getPoidsIdeal(self):
        """calcul du poids ideal pour un chat selon
        sa taille et son age"""
        return self.taille*0.87+(self.age/2)
```

## *La programmation Objet (affichage)*

- ◆ Il existe deux équivalents à la méthode toString de java qui permet d'afficher l'objet de la façon souhaitée

```
def __str__(self):  
    return "2- "+str(self.taille)+"cms, "+str(self.age)+" ans."
```

```
def __repr__(self):  
    return str(self.taille)+"cms, "+str(self.age)+" ans."
```

```
print (minou)  
print (repr(minou))
```

## *La programmation Objet (affichage)*

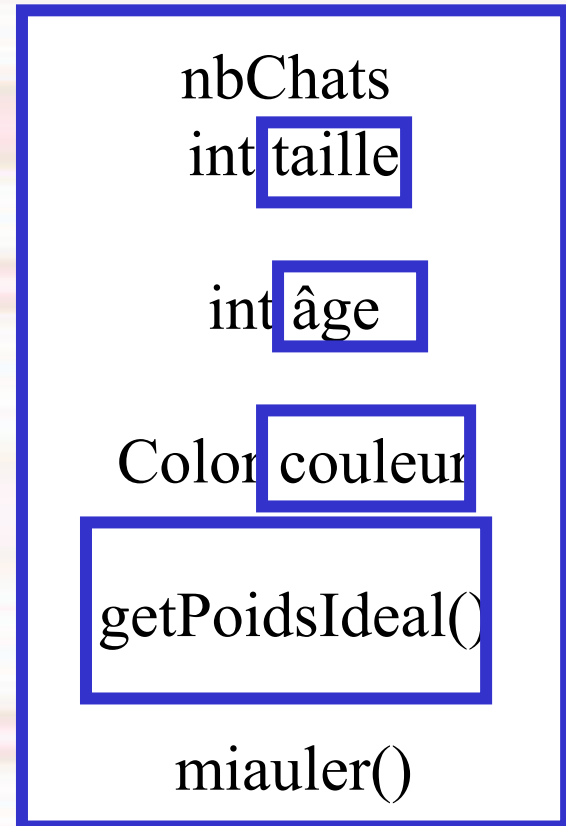
- ◆ str → affichable et lisible par un utilisateur.  
Par défaut c'est repr
- ◆ repr → pour corriger notre code (debug).

## La programmation Objet (encapsulation, syntaxe python 3.xx)

On manipule un attribut de self préfixé par le caractère souligné (convention)

```
def _get_taille(self):  
    print ("je lis taille")  
    return self._taille  
  
def _set_taille(self, t):  
    print ("je modifie taille")  
    if (t<1):  
        self._taille=int(t*100)  
    else:  
        self._taille=t
```

On précise les accesseurs et mutateurs  
taille=property(\_get\_taille, \_set\_taille)



## *La programmation Objet (méthodes de classe)*

```
def miauler(cls):  
    print( "miaouuuuu"*cls.nbChats)
```

```
miauler=classmethod(miauler)
```

```
minou = Chat(4)  
Chat.miauler()  
garfield = Chat(2)  
Chat.miauler()
```

nbChats

int **taille**

int **âge**

Color **couleur**

**getPoidsIdeal()**

miauler()

## *La programmation Objet (méthodes statiques)*

```
def miauler2():  
    print( "mawwww")
```

```
miauler2=staticmethod(miauler2)
```

```
minou = Chat(4)  
Chat.miauler2()  
garfield = Chat(2)  
Chat.miauler2()
```

nbChats

int **taille**

int **âge**

Color **couleur**

**getPoidsIdeal()**

miauler()

## *La programmation Objet (introspection et surcharge)*

- ◆ On peut passer le nom de l'attribut comme argument.

```
__getattr__ __setattr__ __delattr__ __hasattr__  
print (getattr(minou, "taille"))  
print (setattr(minou, "taille", 74))  
print (delattr(minou, "age"))  
print (hasattr(minou, "age"))
```

- ◆ Il existe ce même genre de méthodes pour les conteneurs (liste, dictionnaire, chaîne de caractères...)

```
__getitem__ __setitem__ __delitem__  
__contains__ __len__
```

## *La programmation Objet (surcharge des opérateurs prédéfinis)*

- ◆ On peut surcharger les opérateurs arithmétiques

`__add__`, `__sub__`, `__mul__`...

- ◆ On peut surcharger les opérateurs de comparaison

`__eq__`, `__ne__`, `__gt__`...



## *La programmation Objet (les décorateurs)*

```
@classmethod  
def miauler3(cls):  
    print( "miaouuuuu"*cls.nbChats)
```

```
@staticmethod  
def miauler4():  
    print( "mawwww")
```

```
Chat.miauler3()  
Chat.miauler4()
```

## *La programmation Objet (l'héritage, par défaut héritage de la classe object)*

- ◆ `class Felin:`  
    `"""contient tout ce qui est lié à la taille"""`  
    `def __init__(self):`  
        `self._taille=35`  
    `...`
- ◆ `class Chat (Felin):`  
    `"""Cette classe decrit des chats et hérite des Felins"""`
- ◆ Il faut supprimer dans Chat tout ce qui est lié à la taille et appeler explicitement le constructeur de Felin :  
        `Felin.__init__(self)`

## *La programmation Objet (héritage et outils d'introspection)*

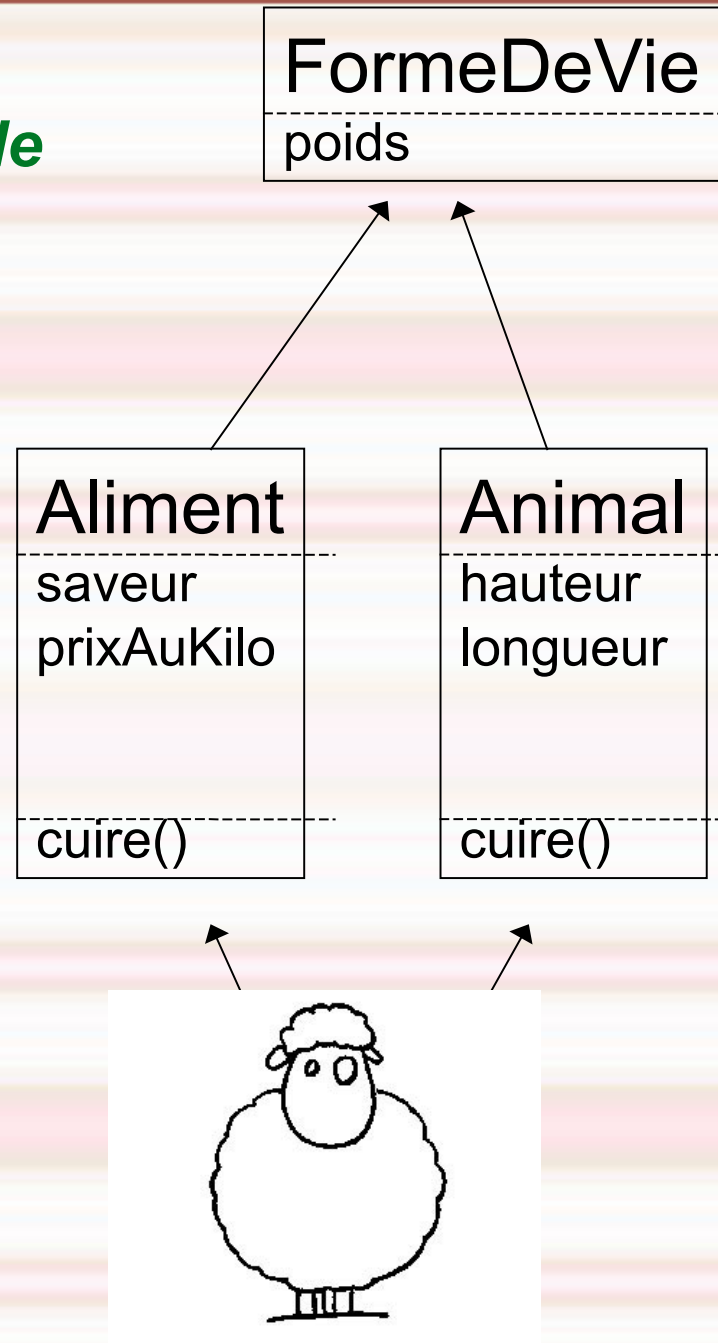
- ◆ `print(issubclass(Chat, Felin))`
- ◆ `print(not issubclass(Felin, Chat))`
- ◆ `print(isinstance(minou, Chat))`
- ◆ `print(isinstance(minou, Felin))`
- ◆ `minou.__dict__`
- ◆ `dir(minou)`

## Héritage Multiple

Nous pourrions souhaiter indiquer que le mouton est un animal ET un aliment.

Mais quelle méthode cuire() devrions-nous utiliser pour cuire le mouton ?

Quel Poids considérer pour le mouton si le Poids n'a pas le même usage pour un aliment et un animal ?



## *Héritage Multiple autorisé*

L'ordre de déclaration des héritages indique l'ordre de prise en compte des attributs et méthodes.

- ◆ `class AnimalCompagnie:`  
    `"""contient tout ce qui est lié à l'âge d'un animal"""`  
    `def __init__(self,age=1):`  
        `self.age=age`
  
- ◆ `class Chat (Felin, AnimalCompagnie):`  
    `def __init__(self, age=1):`  
        `Felin.__init__(self)`  
        `AnimalCompagnie.__init__(self,age)`

## *Les exceptions*

Détecter et gérer les déroulements non habituels du programme.

Fonctionnement de base

On essaie d'exécuter du code (try)

Si une exception survient on l'intercepte (except)

En option :

On peut cibler certaines exceptions.

Si l'exception ne survient pas, on peut effectuer un traitement particulier (else)

On peut effectuer un traitement systématique pour terminer (finally).

## *Les exceptions (toute une hiérarchie de classes Python)*

```
fini = False
while not fini:
    try:
        n = int(input("entrez un nombre "))
    except ValueError:
        print ("perdu : on vous demandait un nombre")
        fini = True
    else:
        print ("gagne, rejouez")
    finally:
        print ("a suivre")
```

## *Faire ses propres exceptions* (exemple open classrooms)

```
class MonException(Exception):  
    def __init__(self,raison):  
        self.raison = raison  
  
    def __str__(self):  
        return self.raison  
  
def multiplier_par_5(n):  
    if n > 20:  
        raise MonException("Le nombre est trop grand !")  
    else:  
        return n * 5
```



## *Faire du test avec des exceptions*

```
try:
    n = int(input("entrez un nombre pair "))
    assert n%2==0
except ValueError:
    print ("perdu : on vous demandait un nombre")
    fini = True
except AssertionError:
    print ("perdu : on vous demandait un nombre pair")
    fini = True
else:
    print ("Bravo !")
```

## ***Faire du test avec unittest***

Automatiser vos tests sans avoir besoin de les regarder de près, méthode par méthode.

## *Ranger son code = Modules = Imports*

- ◆ principaux modes d'utilisation des modules

**import** *nomModule*

le nom du module devra toujours être présent devant le nom de la fonction qui est utilisée.

exemple : import string  
lower=string.lower(chaine1)

## *Modules = Imports*

◆ **from** nomModule **import** nomFonction  
le nom de la fonction utilisée est donné, il n'est pas possible d'utiliser d'autres fonctions (non déclarées) du module.

exemple : `from string import lower`  
`lower=lower(chaine1)`

◆ **from** nomModule **import** \*  
toutes les fonctions du module pourront être utilisées.

exemple : `from string import *`  
`lower=lower(chaine1)`

## ***Modules = Imports***

- ◆ Les fonctions et les procédures peuvent être stockées dans des fichiers différents de celui où elles sont utilisées.

**monModule.py**

```
def maFonction \
    (par1, par2)
    ...
    return(parSortie)

def maProcédure \
    (par4)
```

**monProgramme1.py**

```
from monModule \
    import *
...
val = maFonction \
    (val1, val2)
...
maProcédure(val)
```

**monProgramme2.py**

```
from monModule \
    import maFonction
...
val = maFonction \
    (val1, val2)
...
maProcédure(val)
```

## Exemples de modules

### ◆ math

`ceil(x)` : renvoie le plus petit entier  $i$  supérieur ou égal à  $x$

`exp(x)` : renvoie  $e$  à la puissance  $x$

`sin(x)`, `cos(x)`, `log(x)`

```
import math
print( math.ceil(12.345)
print( math.ceil(12.645)
```

### ◆ string

### ◆ os (Operating System)

vue uniforme des possibilités des différents systèmes

`chdir(path)` : *path* devient le répertoire courant

`rmdir(path)` : le répertoire *path* est supprimé (s'il est vide)

## *Autres exemples*

sys → permet d'effectuer dans Python les principales commandes Systèmes.

random, fractions → génération quasi aléatoire de nombres, utilisation de fractions

Vispy, tKinter → Des outils d'interaction et de visualisation graphique.

## ***Fichiers : Sérialisation***

- ◆ Le programme s'exécute en mémoire vive, quand il est terminé, les données sont « perdues ».
- ◆ Pour rendre les données persistantes, les conserver sur le disque, il faut sérialiser nos variables et constantes pour stocker leur contenu dans un fichier.



## *Fichiers : Sérialisation*

- ◆ Identifier un fichier sur le disque dur
  - Nom du fichier dans le système d'exploitation
    - monFichierDonnees.txt
    - monRepert1/monSousRepert2/monFichierDonnees.txt
- ◆ Modifier le répertoire de lecture : par défaut, avec idle, le répertoire est paramétré dans les propriétés de lancement du programme.
  - Fonction chdir du module os (operating system)
    - from os import chdir
    - chdir("monRepert1/monSousRepert2")

## *Fichiers : Sérialisation*

- ◆ Il existe plusieurs modes de lecture d'un fichier :
  - Séquentiel je lis un roman du début à la fin
  - Indexé je vais chercher le bon mot dans le dictionnaire
  
- ◆ Il existe plusieurs droits sur des fichiers :
  - Écriture
  - Lecture
  - Lecture et Écriture
  
- ◆ Après avoir manipulé un fichier, il faut le fermer et le libérer pour les autres applications
  - `monFichier.close()`

## *Fichiers : Sérialisation*

- ◆ Ouverture du fichier (modes décrits page suivante)
  - `open(<nomFichier>, <mode>)`
  - (mode : lecture et/ou écriture)
  - `open("monRepert1/monSousRepert2/  
monFichierDonnees.txt",'r')`
  - `from os import chdir`
  - `chdir("monRepert1/monSousRepert2")`
  - `open("monFichierDonnees.txt",'r')`
- Lorsqu'un fichier est ouvert en mode modification par un programme, il ne peut pas être utilisé par d'autres programmes.

## *Fichiers : Sérialisation*

- ◆ Les modes d'ouverture de fichier
  - 'r' : le fichier existe déjà, ouverture en lecture seule
  - 'w' : le fichier est ouvert en écriture seule, s'il existe déjà, il est tronqué et écrasé ; il est créé sinon.
  - 'a' : le fichier est ouvert en écriture seule. S'il existe son contenu est conservé et les données ajoutées à la suite ; il est créé sinon.
  - 'r+' : le fichier doit déjà exister, est ouvert en lecture et écriture.
  - 'w+' : le fichier est ouvert en lecture et en écriture, s'il existe déjà, il est tronqué et écrasé ; il est créé sinon.

## *Fichiers : Lecture*

- ◆ Utilisation de la méthode `readline()`
  - `monObjetFichier = open("monFichier.txt","r")`
  - `ligne = monObjetFichier.readline()`
- ◆ Fin du fichier
  - chaîne vide
  
- ◆ Exemple : affichage du contenu d'un fichier :
  - `fichPersonne = open("listePersonne.txt","r")`
  - `ligne = fichPersonne.readline()`
  - `while ligne != "" :`
    - `print( ligne)`
    - `ligne = fichPersonne.readline()`
  - `fichPersonne.close()`

## *Fichiers : Écriture*

- ◆ Utilisation de la méthode write()
  - `monObjetFichier = open("monFichier.txt","w")`
  - `monObjetFichier.write(ligne)`
  
- ◆ Exemple : écriture dans un fichier des chaînes de caractères saisies au clavier ("zzz" dernière chaîne)
  - `fichSortie = open("listeSortie.txt","w")`
  - `ligne = input()`
  - `while ligne!="zzz" :`
    - `fichSortie.write(ligne+"\n")`
    - `ligne = input()`
  - `FichSortie.close()`