```
#-------------------------------------------------------------
#Installer :
# - Anaconda 2.3.0
# - Jet Brains PyCharm Community Edition 4.0.3
# - Pyodbc
#   o https://code.google.com/archive/p/pyodbc/downloads
#   o 3.0.7 32-bit Windows Installer for Python 2.7
#-------------------------------------------------------------

#-------------------------------------------------------------
# Moocs : Youtube
# Moocs : https://openclassrooms.com (2.5)
#          Laurent.tu...                mv.3
#-------------------------------------------------------------


#Type numérique
    --> type(variable)
    int
    long    #Précision illimité
    float   #Précision limité
    complex
        c = 1 + 3 j
        c.real  --> 1.0
        c.imag  --> 3.0
    #Conversion
        int(4.3)    --> 4
    #Opération
        5 / 3   --> 1
        5 % 3   --> 2       #modulo
        5 / 3.0     --> 1.6666666
        5 / float(3)  --> 1.6666666
        5 // 3.1  --> 1.0   # Division entière
        1 < 3   --> True
        3**2    --> 9       #Puissance
        sqrt(9) --> 3       #Racine carré (from math import *)
    #Autres opérations
        #On peut affecter une même valeur à plusieurs
            variables :
        x = y = 3
        #Permutation de variables
        a,b = b,a # permutation
    # condition
        if a.isdigit():

#Séquence
    String
    Liste
```

```
    Tuple

#String
    #on peut couper une instruction, pour l'écrire sur deux
        lignes
    1 + 4 - 3 * 19 + 33 - 45 * 2 + (8 - 3) \
    ... -6 + 23.5
    #le symbole « \ » permet, avant un saut de ligne,
    #d'indiquer à Python que « cette instruction se poursuit à
        la ligne suivante ».

    #Pour écrire un véritable anti-slash dans une chaîne, il
        faut le doubler : « \\ »

    #Permet d'écrire plusieurs lignes
    chaine3 = """Ceci est un nouvel
    ... essai sur plusieurs
    ... lignes"""

    #« \n » symbolise un saut de ligne

    #Youtube
    s = 'egg, bacon'        #String de 10 caractères
    S[0]    --> 'e'
    s[9]    --> 'n'
    'x' in s    -->  False
    s + ' and spam'  --> 'egg, bacon and spam'
    len(s)      --> 10
    min(s)      --> ' '
    max(s)      --> 'o'
    s.index('g')  --> 1 #Position du premier 'g' (commence à 0)
    s.count('g')  --> 2
    s * 3       --> 'egg, baconegg, baconegg, bacon'
    '#' * 30    -->
        '##############################################'


    #Slicing
        egg, bacon
        0123456789
        s[0:3]      --> 'egg'       # 0 inclus : 3 exclus
        s[:3]       --> 'egg'
        s[5:10]     --> 'bacon' # 5 inclus : 10 exclus (max
            est 10)
        s[5:]       --> 'bacon'     # 5 inclus : fin
        s[:]                # Copie de s (Shalow copy)
    #Pas sur un clicing
        s[0:10:2]   --> 'eg ao' # Pas de 2
```

```
        s[::2]      --> 'eg ao'
        s[:8:3]     --> 'e,a'
    #Autres
        s[100]      --> Error
        s[5:100]    --> 'bacon'     # ça fonctionne
        s[50:100]   --> ''          # ça fonctionne mais
            retourne vide
    # Négatif
        #Ordre normal
        e   g   g   ,       b   a   c   o   n
        -10 -9  -8  -7  -6  -5  -4  -3  -2  -1
        s[-10:-7]   --> 'egg'
        s[:-3]      -->'egg, ba'
        #Ordre inverse
        s[::-1]     --> 'nocab , gge'
        s[2:0:-1]   --> 'gg'
        s[2::-1]    --> 'gge'

# String (compléments)
    # *** Est immuable, si opération --> nouvelle string
        affecté à variable ***

    # Méthode
    'spam'.upper()                  --> 'SPAM'
    'SPAM'.lower()
    'spam et fromage'.capitalize()  --> 'Spam et
        fromage'        # La première lettre en majuscule
    'SPAM'.reverse()                --> 'MAPS'
    'spam'.replace ('s', 'p')       --> 'ppam'
    '   une  chaine avec  des espaces   '.strip()
        --> 'une  chaine avec  des espaces'
    'introduction'.center(20)       --> '    introduction    '

    #Format
    'Je m'appelle {0} {1} et j'ai {2} ans.'.format("Paul",
        "Dupont", 21)    --> 'Je m'appelle Paul Dupont et j'ai
        21 ans.'
    adresse = "{no_rue}, {nom_rue}, {code_postal},
        {nom_ville}".format(no_rue=5, nom_rue="rue des
        Postes", code_postal=75003, nom_ville="Paris")


    # Changement par liste
    s = "le poulet, c'est bon"
    l = s.split()
    l       --> ['le', 'poulet,', "c'est", 'bon']
    l[3]    --> 'bon'
    l[3] = 'mauvais'
```

```
    s = " ".join(l)
    s       --> "le poulet, c'est mauvais"
    s = "-".join(l)
    s       --> "le-poulet,-c'est-mauvais"

    #Boucle For : String = Liste
    chaine = "Bonjour les ZER0S"        #Lettre est une
        variable créée par le for, ce n'est pas à vous de
        l'instancier.
    for lettre in chaine:
        if lettre in "AEIOUYaeiouy":
            print(lettre + ' est une voyelle')


#Liste
    #Liste ne stocke que référence vers objet
    # donc ne prend pas de place (juste adresse)

    # ********* Même opération car séquence *********

    # Range (c'est une liste)
        range(3)        --> [0, 1 , 2]
        range(1, 10, 2) --> [1, 3 , 5, 7, 9]

    L = list()  # liste vide
    l = []      # liste vide
    l = [4,' spam', True, 3.2]
    l[3]        --> 3.2
    l[0] = l[0] + 2
    l           -->  [6, 'spam', True, 3.2]
    l[1:2]      -->  'spam'

    # Insérer
        L = ['a', 'b', 'd', 'e']
        L.insert(2, 'c')    # On insère 'c' à l'indice 2
        L       --> ['a', 'b', 'c', 'd', 'e']
        l[1:2] = ['egg', 'spam']
        l       -->  [6,'egg', 'spam', True, 3.2]
        l[3:4] = ['egg', 'beans']
        l       -->  [6,'egg', 'spam', 'egg', 'beans', 3.2] #
            ca a enlevé TRUE
    # Contracter ou effacer
        l[1:3] = [] # liste vide
        l       -->  [6, 'egg', 'beans', 3.2]   # ca a enlevé
            TRUE

    # append            # Econome en mémoire car pas de copie
        l.append('34')
```

```
        l       -->  [6, 'egg', 'beans', 3.2, '34']
    # extend ()
        l.extend([3, 5, 9])          # Or  L += [3, 5, 9]
        l       -->  [6, 'egg', 'beans', 3.2, '34', 3, 5,
            9]
    # Supprimer
        del l[0]         # On supprime le premier élément de la
            liste
        l.remove('a')    # Attention ne supprime que la
            premiere occurrence !
        l = [i for i in l if i != 'a']
    # Pop           #Retourne le dernier élément de la liste
        et l'enlève de la liste
        l.pop   -->  9
        l       -->  [6, 'egg', 'beans', 3.2, '34', 3, 5]
    # ENUMERATE
        L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
        for i, elt in enumerate(L):
            print("{}  {}".format(i, elt))
            --> "0  a" ...
        for elt in enumerate(ma_liste):
            print(elt)
            --> (0, 'a') ...
    #Autre manière
        autre_liste = [[1, 'a'], [4, 'd'], [7, 'g'], [26, 'z']]
        for nb, lettre in autre_liste:
            print("{} {}".format(nb, lettre))
            --> "1 a" ...

    #Fonction ordre
        l.sort()
        sorted(l)
        l.reverse()
        # Exemple :  prendre le 3ème plus grand mots
            sorted(s, key=len, reverse=True)[2]


    #Liste de liste
        L = [1, 3.5, "une chaine", []]



#Tuple
    # ********* Même opération car séquence *********
    # Comme liste mais immuable (pas de modification après
        création)

    t = ()  # liste vide
```

```
    type(t)         -->  <type 'tuple'>

    t = (4, )           # Seulement si un seul élément
    t = (4, 'spam')

    #On peut omettre les ()
    t = 4, 'spam'

    # Convertir en liste pour le modifier
    l = list(t)
    l.append('x')
    t = tuple(l)
    t       -->  (4, 'spam', 'x')

# IF
    if 'x' in l:
        print 'Bravo'
    elif 'n' in l:
        print 'pas mal'
    else:
        'shit'

# Boucle
    # FOR
    for x in range(1,11):
        print x , x**2

    # WHILE
    #Pour interrompre la boucle, vous pouvez taper CTRL + C
    L = range(10)
    while L:
        L.pop(0)
        print L
        --> 0
        --> [1,2,3,4,5,6,7,8,9]
        --> 1
        --> [2,3,4,5,6,7,8,9]
    #On sort au bout de 10

    # BREAK
    while True:     #ne s'arrête jamais
        s = raw_input('Question ?')
        # s = input('Question ?')
        if 'non' in s:
            break
            #Sort de la boucle while si l'utilisateur met 'non'

    # CONTINUE
```

```
    #Le mot-clé continue permet de… continuer une boucle, en
        repartant directement à la ligne du while ou for
    i = 1
    while i < 20:
        if i % 3 == 0:
            i += 4
            print i
            continue # On retourne au while sans exécuter les
                autres lignes
        print i
        i += 1
                -- > 1  2   7   8   13...


# Fonction
    l = [1, 3, 8]
    l2 = [2, 5, 10]

    def fct_carre(l):
        for x in l:
            print x, x**2
        # Si aucun Return, ça retourne <None>
        # Sinon :
        Return 'fin'

    -->fct_carre(l)
    -->fct_carre(l2)

    #Description
    def table(nb, max=10):
        """Fonction affichant la table de multiplication par nb
        de 1*nb à max*nb (max >= 0)"""
        # La chaine qui permet de définir la fonction est une
            docString
        # Si on tape help(table), elle s'affichera

    # Peut retourner plusieurs valeur
    def f(x, y, z):
        return x+y, z
    a, b = f(100, 5, 3)     -->  a = 105, b = 3


    # Nbe d'argument variable
        def fonction_inconnue(*parametres):
            # parametres sera un tuple
        def fonction_inconnue(nom, prenom, *commentaires):
            # parametres seront 2 var et un tuple
```

```
# Dictionnaire
    # ----- Voire Table de hash -----
    # Definition: Collection non ordonnée de couple Clé-Valeur

    # Création
        d = dict()
        d = {}
        # A la main
        d = {'marc' : 39, 'alice' : 30, 'eric' : 38}
        # par liste de Tuple
        l = [('marc',39),('alice',30),('eric',38)]
        d = dict(l)

    # Opération
        len(d)      --> 3
        'marc' in d     --> True
        'louis' in d    --> False
        'louis' not in d    --> True
        d['marc']   --> 39
        del['marc']
        d           -->  {'alice' : 30, 'eric' : 38}
    # La méthode pop supprime la clé mais elle renvoie la
        valeur supprimée
        d.pop('alice')      --> 'alice'
        d           -->  {'eric' : 38}

    # Liste
        d.keys()    --> ['alice','eric']
        d.values()  --> [30, 38]
        d.items()   --> [('alice',30),('eric',38)]

    # Parcours
        for cle in d:   #or
        for cle in d.keys():
        for valeur in d.values():
        for cle, valeur in d.items():

    # On se sert parfois des dictionnaires pour stocker des
        fonctions.
        def fete():     ...
        def oiseau():   ...
        fonctions = {}
        fonctions["fete"] = fete
        fonctions["oiseau"] = oiseau
        fonctions["oiseau"]             --> <function oiseau
            at 0x00BA5198>
        # on essaye de l'appeler
```

```
        fonctions["oiseau"](10)


# Set (ensemble)
    # Definition : Ensemble non ordonné d'objet unique (objet
        immuable)

    # Création
        # A la main
        s = {1,2,3,'spam'}
        # Par liste
        l = [1,2,3,3,'spam']
        s = set(l)
        s       --> set([1,2,3,'spam'])  #Objet doit être unique
    # Opération
        s.add('egg')
        s       --> set([1,2,3,'egg','spam'])
        s.remove(2)
        s       --> set([1,3,'egg','spam'])
        s.update([5,6])
        s       --> set([1,3,5,6,'egg','spam'])
    # Différence
        s2 = {1,5,'ham'}
        s - s2  --> set([3,6,'egg','spam'])             #
            Différence
        s | s2  --> set([1,3,5,6,'egg','spam','ham'])   # Union
        s & s2  --> set([1,5])                          #
            Intersection

    # FrozenSet (Set immuable)
        fs = frozenset(s)
        fs.add(3)  --> Error


# Référence partagée
    # -- au moins 2 variables référence un même objet --
    a = [1,2]
    b = a
    a[0] = 'spam'
    b[0]                --> 'spam' # et non pas 1

    # Shalow Copy
    # -- Faire que b reste le même --
    b = a[:]
    a[0] = 'spam'
    b[0]    --> 1
        #Exception
        a = [1,[2]]
```

```
        b = a[:]
        a [1][0] = 'Spam'
        b [1][0]    --> 'spam' # et non pas 1
        #Comment éviter cela
        import Copy
        a = [1,[2]]
        b = copy.deepcopy(a)
        a [1][0] = 'Spam'
        b [1][0]    --> 2

# Module (Fichier .py)
    import math
    dir(math)   --> toutes les fonctions du module math
    help(math)  --> toute l'aide
    help(math.tan)  --> Retrun the tan of x...
    math.log(10)    --> 2.302...

    #moocs papier (1.7)
    # On peut appeler un module depuis un autre, contenu dans
        le même répertoire en précisant le nom du fichier
        (sans l'extension .py)
    import os
    from multipli import *

    # Si l'on met tout le code dans le même module Comment
        séparer les fonctions à importer dans d'éventuels
        autres modules
    # et les commande qui ne doivent être actionné que si l'on
        clic sur ce module
    if __name__ == "__main__":
        table(4)                        #(table est dans multipli)
        os.system("pause")
    # Voilà. À présent, si vous faites un double-clic
        directement sur le fichier multipli.py, vous allez voir
    # la table de multiplication par 4. En revanche, si vous
        l'importez, le code de test ne s'exécutera pas.
    # Si la variable __name__ est égale à __main__
        Cela veut dire que le fichier appelé est le fichier
        exécuté



# Les packages
# Un package sert à regrouper plusieurs modules

    # Importer des packages
    import nom_bibliotheque
    # Pointe vers le sous-package evenements
```

```
    nom_bibliotheque.evenements
    # Pointe vers le module clavier
    nom_bibliotheque.evenements.clavier

    # Importer un seul module
    from nom_bibliotheque.objets import bouton

    # Créer ses propres packages
    # En Python, vous trouverez souvent le fichier
        d'initialisation de package __init__.py dans un
        répertoire destiné à devenir un package.


# 1.8. Les exceptions
annee = input()
try: # On essaye de convertir l'année en entier
    annee = int(annee)
except:
    print("Erreur lors de la conversion de l'année.")
    annee = 2000
    # Cette méthode est assez grossière. Elle essaye une
        instruction et intercepte n'importe quelle exception
    # C'est une mauvaise habitude à prendre !
    # Python peut lever des exceptions qui ne signifient pas
        nécessairement qu'il y a eu une erreur.

# Voici une manière plus élégante et moins dangereuse :
try:
    resultat = numerateur / denominateur
except NameError:
    print("La variable numerateur ou denominateur n'a pas été
        définie.")
except TypeError:
    print("La variable num ou denum possède un type
        incompatible avec la division.")
except ZeroDivisionError:
    print("La variable denominateur est égale à 0.")

# Finally permet d'exécuter du code après un bloc try
try:
    resultat = numerateur / denominateur
except ...
finally:
    # Instruction(s) exécutée(s) qu'il y ait eu des erreurs ou
        non


# Pass:  tester un bloc d'instructions… mais ne rien faire si
```

```
    erreur
try:
    resultat = numerateur / denominateur
except:
    pass

# Les Assertions
    # Les assertions sont un moyen simple de s'assurer, avant
        de continuer, qu'une condition est respectée.
    # En général, on les utilise dans des blocs try … except.
assert var == 5     # Si le test renvoie True, l'exécution se
    poursuit normalement. Sinon, une exception AssertionError
    est levée.
assert var == 8
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    AssertionError

#À quoi cela sert-il, concrètement ? Exemple :
annee = input("Saisissez une année supérieure à 0 :")
try:
    annee = int(annee) # Conversion de l'année
    assert annee > 0
except ValueError:
    print("Vous n'avez pas saisi un nombre.")
except AssertionError:
    print("L'année saisie est inférieure ou égale à 0.")



# Revenir sur if
    # Est considéré comme faux :
    False 0 [] {} () '' None
    # Tester si c'est un chiffre
    s = '123'
    if s.isdigit():
        p = int(s) + 10     # Ca fonctionne

# ITERATEUR
    # tout les built-in has an iterator (you can make a boucle
        FOR on it)
    s = {1, 2, 3, 'a'}
    for i in s: ...

    #Iter
    it = s.__iter__()
    it.next     --> 'a'
    it.next     --> 1
```

```
    it.next     --> 2
    it.next     --> 3
    it.next     --> ERREUR StopIteration
    # Un itérateur ne peut parcourir un objet qu'une seule fois
    # Objet a 2 méthodes
        __iter__()
        next()
    # Donc
    L = [1,2,3]
    it = L.__iter__()
    it2 = L.__iter__()
        it is it2   --> False


# Liste de lecture 4
# FICHIER
    #Ecrire 'w'
    f = open (r'C:\tmp\spam.txt', 'w')
    # r = raw string
    # w : write (va créer le fichier ou le vider
    for i in range(100):
        line = '{} {}\n'.format(i, i**2)
        #\n : retour chariot
        f.write(line)
    f.close()       #Save et close

    #Lire 'r'
    #parcourir les lignes car un fichier a un itérateur par
        ligne
        f = open (r'C:\tmp\spam.txt', 'r')
        f2 = open (r'C:\tmp\spam2.txt', 'w')
        for l in f:
            f2.write(l.replace(' ',', '))
        f.close()
        f2.close()
    #Read
        f = open (r'C:\tmp\spam.txt', 'r')
        contenu = f.read()
        f.close()

    #Append 'a' : écriture mais sans écraser l'ancien fichier
        et mettre à la fin
    f = open (r'C:\tmp\spam.txt', 'a')

    #Changer le répertoir courant
    import os
    os.chdir("C:/tests python") # Assigner le CWD ( = «
        Current Working Directory » )
```

```
    os.getcwd()                 # Retrouver le CWD
        #Chemin relatif
        # si on se trouve dans C:, notre chemin relatif sera
            "test\fic.txt"
        # "..\rep1\fic1.txt" sera le chemin si on est dans
            "rep2"
        f = open (r'/spam.txt', 'r')

    #Fermer automatiquement le fichier au cas ou on oublie
        Close()
    with open('C:\tmp\spam.txt', 'r') as f:
        # Opérations sur le fichier
        contenu = f.read()
        # Si une exception se produit, le fichier sera tout de
            même fermé à la fin du bloc.
        # Il est inutile, par conséquent, de fermer le fichier
            à la fin du bloc with

    #'b' binary     #va servir pour sauver des objets
    with open('C:\tmp\spam.txt', 'wb') as fichier:

    # Pickle : Enregistrer Objets dans Fichiers (Pickler)
        import pickle as p
        with open('donnees', 'wb') as fichier:
            mon_pickler = p.Pickler(fichier)
            # enregistrement
            mon_pickler.dump(objet1)
            mon_pickler.dump(objet2)
    # Unpickler : récupérer les objets dans les fichiers
        with open('donnees', 'rb') as fichier:
            mon_depickler = p.Unpickler(fichier)
            # Lecture des objets contenus dans le fichier...
                récupère dans l'ordre
            objet1 = mon_depickler.load()
            objet2 = mon_depickler.load()



# Fonction LAMBDA
    f = lambda x: x**2 - 3
    f(1)    --> -2      #Aucun interêt

    #On peut mettre dans une liste
    L = [lambda x : x**2 - 3, lambda x : x**3 - 5]
    # Puis faire :
    def f(x):
        print x, x(3)       # x est donc une fonction
    f(L[0])     -->     <fonction <lambda> at 0x02b684> 1
```

```
        # x : Adresse de la fonction, x(3) = 1

    # On aurait pu faire direct
    f(lambda x : x**2 - 3)

#MAP
    L = range(10)
    map(lambda x : x**2, L)
        --> [0,1,4,9,16,25,36,49,64,81]

#FILTER
    filter(lambda x : x % 2 ==0, L]
        --> [0,2,4,6,8]

# Compréhension de liste
# Puissance des itérateurs avec syntaxe simple
    # Comme MAP
    [x**2 for x in range(10)]
        --> [0,1,4,9,16,25,36,49,64,81]

    # Comme FILTER
    [x**2 for x in range(10) if x % 7 ==0]
        --> [0,49]


    #Compréhension de SET (idem)
    {i**3 for i range(10)}
        --> set([0,1,8,64,512,343,216,729,27,125])

    #Compréhension de Dictionnaire (idem)
    {i: i **2 for i in range(10)}
        --> {0:0, 1:1, 2:4, 3:9, 4:16, 5:25, 6:36, 7:49, 8:64,
            9:81}
    # Changer des dico
    d = {123 : 'marc', 145 : 'eric', 655 : 'jean'}
    d[123]      --> 'marc'
    # Le dico n'est pas fait pour accéder au clé par valeur
    # Sans faire de manière itérative
    # --> renverser dictionnaire à l'envers
    d2 = {d(k) : k for k in d}
    d2['eric']      --> 145




# 5.2/3 Importantion des modules
    #Trouve le module dans le même dossier
    #sinon dans le dossier PYTHONPATH
```

```
    #sinon Librairie standard SYS.PATH
    #Execute le fichier (fonction créer seulement à l'appel
        des fonctions)
    #SYS
        import sys
        sys.path
    #OS
        import os
    #Uniquement une valeur ou fonction
        from os import x
        # Avantage
        from math import cos()
        cos(10)     # Fonctionne direct dans faire math.cos(10)

    # Quand le module est dans un autre dossier
        import sys

            sys.path.append("F:\LYXOR-ETF-BIU\ETF_DW\Python\Cobr
            a\Fonction\\")
        import fct_Date



# 5.4 Class, Instance & Méthodes
    class c:
        x=1
    #Instance
        I = c()
    #Espace de nommage
        c.__dict__
            --> {'x':1, '__module__':'__main__', '__doc__':
                None}
        I.__dict__
            --> {}
    c.x     --> 1
    I.x     --> 1   #Car va aller chercher dans class

    c.y = 10    # création
    c.x = 5     # update
    I.x     --> 5
    I.y     --> 10  #même si instance défini avant création de Y

    I.x = 'a'
    I.x     --> 'a'
    c.x     --> 5

    #Fonction = methode dans une classe
    class c:
```

```
        x=1
        def f(self, a):
            print self.x
            self.x = a
    I = c()
    I.f(5)      --> 1       #Raccourci de : C.f(I, 5)
    I.f(10)     --> 5

# 5.5 Heritage
    class C1: pass
    class C2: pass
    class C(C1, C2):
        def f(self,a):
            self.x = 10
    #C1 et C1 sont des super class pour C et si qque chose est
        absent dans C, ça va aller chercher dans C1 ou C2
    I1 = C ()
    I2 = C ()
    #Arbre d'héritage : I1 et I2 vont aller chercher dans C
        puis dans C1 et C2 si besoin

    #Exemple
    class C:
        def set_x(self, x):
            self.x = x
        def get_x(self):
            print self.x
    #Sous classe de C
    class sousC(C):
        def get_x(self):
            print 'x est : ', self.x
    sousC.__bases__ --> Retourne les Super classe : C ici
    #Instance
    c = C()
    sc = sousC()
    c.set_x(10)     # Va aller dans C
    sc.set_x(20)    # Va aller dans sousC puis dans C
    c.get_x         --> 10
    sc.get_x        --> x est : 20


# 5.6 Surcharge




##########################################################
```

```
### MOOCS écrit sur les classes ###
#####################################################

    # 3.1 Class
        # Constructeur : méthode de notre objet se chargeant
            de créer nos attributs
        class Personne:
            '''Classe définissant une personne caractérisée
                par : son nom, son prénom, son âge son lieu de
                résidence'''

            def __init__(self): # Notre méthode constructeur
                self.nom = "Dupont"

        # Quand on appelle la classe
        bernard = Personne()
        bernard          -->     <__main__.Personne object at
            0x00B42570>
        bernard.nom      -->     'Dupont'
        bernard.nom  = 'Tupin'
        bernard.nom      -->     'Tupin'

        # Avec paramètres d'entrées
        class Personne:
            '''Classe définissant une personne caractérisée
                par : son nom, son prénom, son âge son lieu de
                résidence'''

            def __init__(self, nom, prenom):
                self.nom = nom
                self.prenom = prenom
                self.age = 33
                self.lieu_residence = "Paris"


        # Attributs de classe : Attributs identique pour
            toutes instances
        class Personne:
            '''Classe définissant une personne caractérisée
                par : son nom, son prénom, son âge son lieu de
                résidence'''

            objets_crees = 0 # Le compteur vaut 0 au départ
            def __init__(self):
                '''À chaque fois qu'on crée un objet, on
                    incrémente le compteur'''
                Compteur.objets_crees += 1
                # On ne peut pas accéder à l'attribut depuis
```

```
            méthode d'une classe à Classe
        # Accessibilité des variables LG pour les
            classes (Local, Global)
        # On doit préciser '''compteur.attribut'''


# Méthode
class TableauNoir:

def __init__(self):
    self.surface = ""
def ecrire(self, message_a_ecrire):
    if self.surface != "":
        self.surface += "\n"
    self.surface += message_a_ecrire


# Méthodes de classe : ne prend pas en premier
    paramètre SELF (l'instance de l'objet) mais CLS
    (la classe de l'objet)
class Compteur:
    '''Cette classe possède un attribut de classe qui
        s'incrémente à chaque fois que l'on crée un
        objet de ce type'''
    objets_crees = 0 # Le compteur vaut 0 au départ
    def __init__(self):
        '''À chaque fois qu'on crée un objet, on
            incrémente le compteur'''
        Compteur.objets_crees += 1
    def combien(cls):
        '''Méthode de classe affichant combien
            d'objets ont été créés'''
        print("Jusqu'à présent, {} objets ont été
            créés.".format(cls.objets_crees))
    combien = classmethod(combien)


# Méthodes statiques : elles ne prennent aucun premier
    paramètre, ni SELF ni CLS
#   - indépendant de toute donnée, aussi bien contenue
    dans l'instance de l'objet que dans la classe
class Test:
    '''Une classe de test tout simplement'''
    def afficher():
        '''Fonction chargée d'afficher quelque chose'''
        print("On affiche la même chose.")
        print("peu importe les données de l'objet ou
            de la classe.")
```

```
        afficher = staticmethod(afficher)

    # DIR : Voir les attributs et les méthodes d'une classe
    class Test:
        def __init__(self):
            self.mon_attribut = "ok"
        def afficher_attribut(self):
            print("Mon attribut est
                {0}.".format(self.mon_attribut))

    un_test = Test()
    dir(un_test)
        --> ['__class__', '__delattr__', '__dict__',
            '__doc__', '__eq__', '__format__', '__ge__'
            , '__getattribute__', '__gt__', '__hash__',
                '__init__', '__le__', '__lt__','__module__'
            , '__ne__', '__new__', '__reduce__',
                '__reduce_ex__', '__repr__', '__setattr__'
            , '__sizeof__', '__str__', '__subclasshook__',
                '__weakref__'
            , 'afficher_attribut', 'mon_attribut']
        # Méthodes spéciales de Python
            __dict__    # Attribut qui est un
                dictionnaire, contient clés = noms des
                attributs, valeurs = valeurs des attributs
            un_test.__dict__    --> {'mon_attribut': 'ok'}


# 3.2 Encapsulation
    # Propriété : Accesseurs et Mutateurs
    mon_objet.mon_attribut      #va devenir
        mon_objet.get_mon_attribut()
        mon_objet.set_mon_attribut(valeur)

    class Personne:
        def __init__(self, nom, prenom):
            '''Constructeur de notre classe'''
            self._lieu_residence = "Paris" # Notez le
                souligné _ devant le nom // interdit
                l'accès depuis l'extérieur
        def _get_lieu_residence(self):
            '''Méthode qui sera appelée quand on
                souhaitera accéder en lecture à l'attribut
                "lieu_residence"'''
            return self._lieu_residence
        def _set_lieu_residence(self, nouvelle_residence):
            '''Méthode appelée quand on souhaite modifier
                le lieu de résidence'''
```

```
            self._lieu_residence = nouvelle_residence
        # On va dire à Python que notre attribut
            lieu_residence pointe vers une propriété
        lieu_residence = property(_get_lieu_residence,
            _set_lieu_residence)

    # Quand on veut accéder à objet.lieu_residence
    # Python tombe sur une propriété redirigeant vers la
        méthode _get_lieu_residence
    # Quand on beut modificer objet.lieu_residence -->
        Redirection vers _set_lieu_residence

    jean = Personne("Micado", "Jean")
    jean.age        --> 33
    jean.lieu_residence     --> 'Paris'
        #On accède à l'attribut lieu_residence !
            _get_lieu_residence
    jean.lieu_residence = "Berlin"                  #
        _set_lieu_residence
    jean.lieu_residence     --> 'Berlin'
        #On accède à l'attribut lieu_residence !
            _get_lieu_residence


# 3.3 Méthode spéciale
    __init__
    __del__

    def __del__(self):
        '''Méthode appelée quand l'objet est supprimé'''
        print("C'est la fin ! On me supprime !")

        #La destruction ? Quand un objet se détruit-il ?
        del (del mon_objet)     #ou fin de fonction ou
            programme
        #Cela ne sert à rien la plupart du temps sauf si
            on veut récupérer une info avant destruction

    __repr__
        def __repr__(self):
            '''Quand on entre notre objet dans
                l'interpréteur'''
            return "Personne: nom({}), prénom({}),
                âge({})".format(self.nom, self.prenom,
                self.age)

        #Change
        p1 = Personne("Micado", "Jean")
```

```
    p1      --> <__main__.XXX object at 0x00B46A70>
    # en
    p1      --> Personne: nom(Micado), prénom(Jean),
        âge(33)

__getattr__
    # Permet de renvoyer un message si un attribut
        n'existe pas
    class Protege:
        def __init__(self):
            self.alibaba = 1
        def __getattr__(self, nom):
            print("Alerte ! Il n'y a pas d'attribut {}
                ici !".format(nom))

    pro = Protege()
    pro.alibaba     --> 1
    pro.element     --> Alerte ! Il ny a pas dattribut
        element ici !

__setattr__
__delattr__

# Méthodes mathématiques
class Duree:
    '''Classe contenant des durées sous la forme d'un
        nombre de minutes et de secondes'''
    def __init__(self, min=0, sec=0):
        self.min = min      # Nombre de minutes
        self.sec = sec      # Nombre de secondes
    def __str__(self):
        return "{0:02}:{1:02}".format(self.min,
            self.sec)

d1 = Duree(3, 5)
print(d1)       --> 03:05

# Comment additionner des durée
def __add__(self, objet_a_ajouter):
    '''L'objet à ajouter est un entier, le nombre de
        secondes'''
    nouvelle_duree = Duree()
    # On va copier self dans l'objet créé pour avoir
        la même durée
    nouvelle_duree.min = self.min
    nouvelle_duree.sec = self.sec
    # On ajoute la durée
    nouvelle_duree.sec += objet_a_ajouter
```

```
            # Si le nombre de secondes >= 60
            if nouvelle_duree.sec >= 60:
                nouvelle_duree.min += nouvelle_duree.sec // 60
                nouvelle_duree.sec = nouvelle_duree.sec % 60
            # On renvoie la nouvelle durée
            return nouvelle_duree
##################################################################
```

```
#-----------------------------------------------------------
# Exo & Fonction classique
#-----------------------------------------------------------

# Année bissextile
    annee = input("Saisissez une année : ")
    annee = int(annee)
    if annee % 400 == 0 or (annee % 4 == 0 and annee % 100 !=
        0):
                print("L'année saisie est bissextile.")
    else:
                print("L'année saisie n'est pas bissextile.")
    os.system("pause")
    # On met le programme en pause pour éviter qu'il ne se
        referme (Windows)


# Arrondir un float et changer point par virgule
    afficher_flottant(3.99999999999998)     -->     '3,999'

    def afficher_flottant(flottant):
        if type(flottant) is not float:
                    raise TypeError("Le paramètre attendu doit
                        être un flottant")
        flottant = str(flottant)
        partie_entiere, partie_flottante = flottant.split(".")
        return ",".join([partie_entiere, partie_flottante[:3]])
```

```python
#Déterminer si un  nombre est premier
    def PrimeTime(num):
        L = [x for x in range(2,num) if num % x == 0]
        if len(L) == 0:
            return 'true'
        else:
            return 'false'
    # keep this function call here
    print PrimeTime(raw_input())
```