Programmation orientée objet avec Python

Mars 2021

académie Versailles
GRETA
DES HAUTS-DE-SEINE
FORMEZ-VOUS À DEMAIN
RÉGION ACADÉMIQUE ÎLE-DE-FRANCE
LUDIKSCIENCES

Fichiers (chemin)

Objectif ouvrir un fichier sur le bureau :
Ex : C:\Utilisateur\Toto\Bureau\monfichier.txt

Exemple avec chemin absolu :

```
with open ('/Users/Toto/Desktop/monfichier.txt') as f:
    data = f.read()
…
```

ou

```
with open ('c:\\Users\\Toto\\Desktop\\monfichier.txt') as f:
    data = f.read()
…
```

Exemple en construisant le chemin sous forme de path :

```
import os

monfichier = os.path.join('c:\\Users\\Toto\\Desktop',
                          'monfichier.txt')
with open (monfichier) as f:
    data = f.read()
```

LUDIKSCIENCES

Os

Import du module os :

```
>>> import os
```

Répertoire courant :

```
>>> os.getcwd()
'C:\\Users\\Greta92'
```

Changer de répertoire :

```
>>> os.chdir('C:\\Windows\\System32')
>>> os.getcwd()
'C:\\Windows\\System32'
```

Structure chemin/fichier:

```
>>> chem = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(chem)
'calc.exe'
>>> os.path.dirname(chem)
'C:\\Windows\\System32'
```

Mais aussi :

os.listdir('C:\\Windows')
os.path.exists('C:\\Windows')
os.path.isdir('C:\\Windows')
os.path.isfile('C:\\Windows\\System32\\calc.exe')
…

```
>>> file = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(file)
('C:\\Windows\\System32', 'calc.exe')
```

LUDIKSCIENCES

Les mots-clés ajoutés

False	None	True	and	as	assert
break	class	continue	def	del	elif
else	except	finally	for	from	global
if	import	in	is	lambda	nonlocal
not	or	pass	raise	return	try
while	with	yield			

LUDIKSCIENCES

Call object by reference

Une fonction peut-elle modifier ses arguments ?

Les autres langages utilisent :
Call by value : on évalue les arguments, avant de passer les valeurs à la fonction. Elle ne peut pas changer les arguments. (Langage C)
Call by reference : les des arguments sont transmises. La fonction peut modifier les informations à ces adresseadressess (Fortran , VB, PHP, C++/C#)
Call by name et +…

Python est "Call by object reference" ou "Call by sharing"
Pour des types non modifiables (int, float, str, tuple , cela s'apparente au "Call by value")

- La fonction ne peut pas modifier de tels arguments

Pour les autres, au "call by reference"

- La fonction peut modifier la valeur de ces arguments

Attention des types non modifiables peuvent contenir des références à des objets modifiables et des effets de bord sont possibles malgré les apparences.

Classeur CallByObjectReference.ipynb

LUDIKSCIENCES

Résultat d'une fonction

pas de valeur
absence de return

```
None
```

une valeur

```
return x*math.sin(x)
…
print(f(3)
```

plusieurs valeurs
on récupère un tuple

```
return 2*pi*r, pi*r**2
…
périmètre, surface = cercle (1.5)
```

LUDIKSCIENCES

Arguments positionnels

Arguments positionnels
- même position dans la définition et dans l'appel

Souvent baptisés *args*

```
def f(x, y, z):
    return(math.sqrt(x*x+y*y+z*z))

print (f(a, b, c))
```

Valeurs par **défaut**
- pour les derniers arguments uniquement
  - recours aux arguments nommés en cas d'ambiguïté

```
def f(x, y=0, z=0):
    return(math.sqrt(x*x+y*y+z*z))



print (f(a, b), f(a, z=2))
```

LUDIKSCIENCES

Arguments positionnels en nombre variable

*x signifie que x est un tuple

**args*

```
def f(*x):
    t=0
    for u in x:
        t+=u*u
    return(math.sqrt(t))
…
print (f(a, b, c))
```

Arguments nommés

Keyword Args
souvent baptisé *kwargs*

Valeurs par défaut explicite
Position non significative

```
def f(x=0, y=0, z=0):
    return(math.sqrt(x*x+y*y+z*z))
…
print (f(x=a, y=b, z=c))
print (f(y=b, z=c, x=a))
```

LUDIKSCIENCES

Arguments nommés en nombre variable

**d est un dictionnaire dans lequel on trouve les arguments

***kwargs*

```
def g(**d):
    x=d['x']
    y=d['y']
    z=d['z']
    return(math.sqrt(x*x+y*y+z*z))
…
print (g(x=a, z=b, y=c))
```

LUDIKSCIENCES

Arguments par défaut non modifiables

Les valeurs par défaut sont non modifiables

plus de souplesse avec None

l'algorithme de la fonction détermine une valeur par défaut calculée

on teste si x est l'objet None

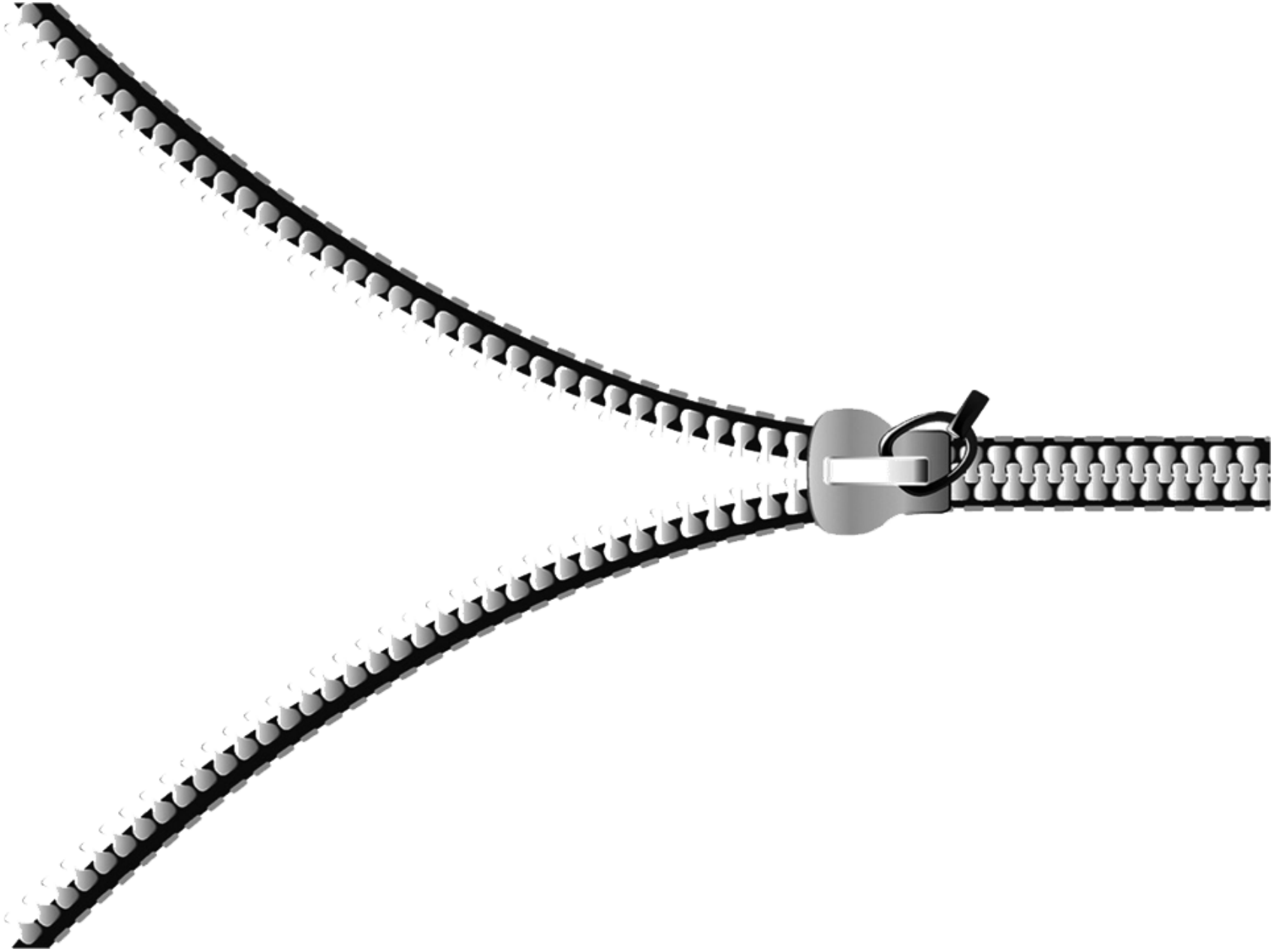
```
def f(x = None):
    …
    if x is None:
        x= -b /(2*a)
    …
```

LUDIKSCIENCES

Signature des fonctions

les arguments positionnels
avant
les arguments nommés

```
f(*args, **kwargs)
```



LUDIKSCIENCES

Zip

zip réunit deux objets itérables

```
a = [1, 2, 3]
neveux = ['Riri', 'Fifi', 'Loulou]
zipped = zip(a, neveux)
```

dans cet exemple, un couple de listes est zippé en une liste de couples.

Attention zip renvoie un objet zip (un itérateur)
Le convertir en liste

```
zipped = list(zipped)
```

Il n'y a pas de unzip. C'est zip qui est utilisé
* est l'opérateur qui déballe les arguments (*splat*, *unpacking argument lists*)

```
l1, l2 = zip(*zipped)
```

Classeur Jupyter Zip.ipynb

LUDIKSCIENCES

Argument fonctionnel

un argument à une fonction peut être une fonction

Noter l'usage de l'*underscore* dans le nom de la fonction

La fonction s'appelle map_ parce que Python connaît une fonction map qui fait presque la même chose

```
import math

def f(x):
    return x*x

def g(x):
    return math.sin(x)

def map_(func, l):
    t=[]
    for e in l:
        t.append(func(e))
    return t

print(map_(f, list(range(3))))
print(map_(g, list(range(3))))
```

LUDIKSCIENCES

Variables globales

La portée des noms (*scope*) peut être
- globale
  - le fichier en cours
    - globals()
- locale
  - la fonction en cours
    - locals()

Priorités
- local < global < prédéfini

Une fonction ne peut pas modifier une variable globale à moins de l'avoir déclarée comme "global"

```
import math
# compter les appels à f
n=0

def f(x):
    global n
    #compter l'appel
    n+=1
    return x*math.sin(x)



def main():
   print(f(0), f(math.pi), f(5), sep"\t")
   print(n)

main()
```

Classeur Jupyter GlobalLocal.ipynb

LUDIKSCIENCES

Fonctions imbriquées

la définition d'une fonction peut être locale à une autre fonction

- Sa définition est incluse dans celle de la fonction qui l'utilise
- Elle n'est pas définie en dehors

Définir des fonctions auxiliaires sans polluer le scope global

- Organisation du code
- Eviter des effets de bords non désirés
- Faciliter la maintenance

Remarque : dans l'exemple ci-contre n reste défini au niveau global

- Comment introduire n dans main ?
  - Essayer!!!

```
import math
# compter les appels à f
n=0
def main():

    def f(x):
        global n
        #compter l'appel
        n+=1
        return x*math.sin(x)



    print(f(0), f(math.pi), f(5),
          sep="\t")
    print(n)

main()
print(f(0)) # NameError : name 'f' is
              not defined
```

Classeur Jupyter GlobalLocal.ipynb

LUDIKSCIENCES

nonlocal

Pour indiquer qu'un symbole est défini dans un niveau englobant

- mais pas nécessairement le niveau global
- on utilise nonlocal

```
import math
def main():
    # compter les appels à f
    n=0
    def f(x):
        nonlocal n
        #compter l'appel
        n+=1
        return x*math.sin(x)

    print(f(0), f(math.pi), f(5),
          sep="\t")
    print(n)

main()
```

Classeur Jupyter GlobalLocal.ipynb

LUDIKSCIENCES

Espace de nommage

namespace
gérés par des dictionnaires

```
builtins
```

les fonctions et constantes prédéfinies par Python

```
__builtins__
dir(__builtins__)
```

lister les noms locaux, globaux, connus en un point du code

```
locals()
globals()
```

le préfixe du nom est le nom du module

```
math.pi
random.randint()
```

LUDIKSCIENCES

Module

un module est un fichier .py qui contient des définitions
- de fonctions
- de variables

```
mon_module.py
```

pour l'importer
- même si on l'importe plusieurs fois
- dans un même projet, les variables ne sont initialisées que la 1ère fois

```
import mon_module
```

pour en faire l'inventaire

```
dir(mon_module)
```

si le module a été modifié et qu'il est nécessaire de le réimporter

```
reload mon_module
```

LUDIKSCIENCES

LUDIKSCIENCES

Package

Un package est un ensemble de modules *.py
En général, réunis dans un sous-dossier

Jusqu'à la version 3.3, il fallait faire figurer dans les sous-dossier, un source spécial

__init.py__

souvent vide

permet de limiter les symboles qui seront exportés

tous les autres restent internes

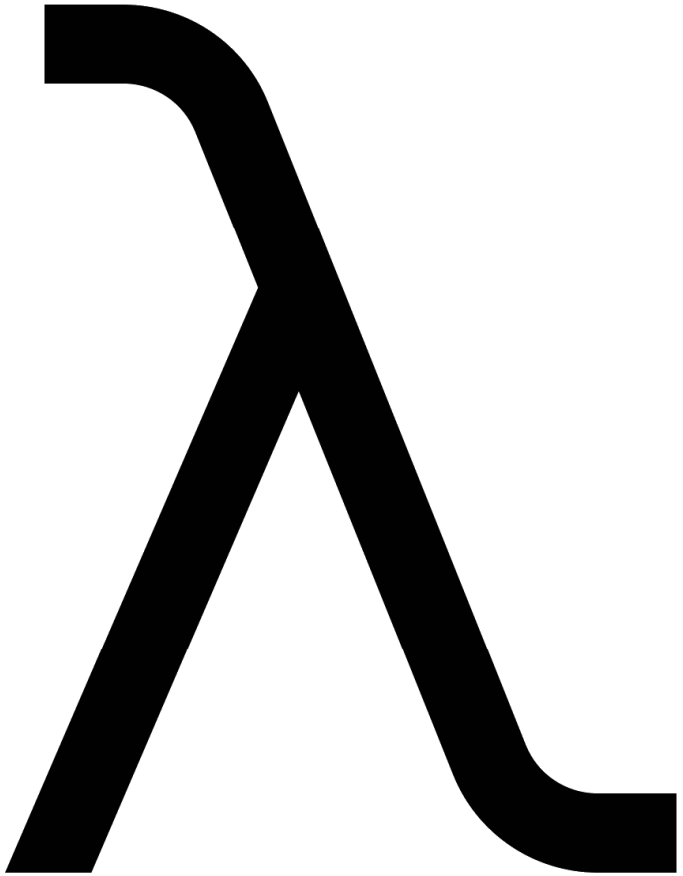
Depuis la version 3.3 (PEP 420) Python dispose de packages implicites et ce fichier __init__.py n'est plus nécessaire. Il suffit de répartir les sources dans une hiérarchie de dossiers pour faire un package.
Organiser le code

```
import mon_package #comme un module
```

```
all= ["ma_fonction"]
```

https://www.python.org/dev/peps/pep-0420/

LUDIKSCIENCES



LUDIKSCIENCES

Lambda

λ ou en majuscule : Λ

λ –calcul – thèse de A. Church (1930)

"tout est fonction"

Définir une fonction anonyme

absence du nom !

présence de la liste d'arguments

expression qui donne la valeur de retour

```
lambda x: x*x    # est une fonction
```

en partie droite d'une affectation

def f(x):… est préféré

```
f = lambda x: x*x
print (f(12))
```

Remplacer un argument fonctionnel

```
map(lambda x: x*x, list(range(3)))
```

Classeurs Jupyter Dir et Map

LUDIKSCIENCES

Map Filter Reduce

Map

appliquer une même fonction à tous les éléments d'une séquence
en construisant un itérateur sur une nouvelle séquence

Ex: transformer une liste de nombres en liste de str

Filter

appliquer une fonction de sélection à tous les éléments d'une séquence
l'itérateur obtenu donne les seuls éléments pour lesquels la fonction a retourné une valeur Vrai.

Reduce

Combiner 2 à 2 les éléments d'une séquence en partant de la gauche

Classeurs Jupyter Dir et Map

LUDIKSCIENCES

Yield

Comme return

```
def g():
    …
    yield x
    suite
```

g retourne alors un générateur

```
gen = g()
```

chaque appel successif se fait par

```
next(gen)
```

on récupère alors le résultat transmis par yield

Contrairement à return, l'appel suivant reprend à l'instruction qui suivait le yield

Classeurs Jupyter Générateurs.ipynb

LUDIKSCIENCES

Expression génératrice

Liste de 1000 entiers en compréhension va en mémoire !

```
[ i in range (10000) ]
```

Expression génératrice

```
( i in range (10000) )
```

Noter les parenthèses

Les valeurs sont fabriquées lorsqu'il y en a besoin.

Evaluation "paresseuse"

lazy evaluation

LUDIKSCIENCES

Décorateur

Signalé par @
S'applique à la fonction dont la définition suit

```
@decorateur1
def func(x):
    …
```

Un décorateur est une fonction
- Un seul argument, fonctionnel
- Une seule valeur retournée, une fonction

```
def decorateur(f):
    def g (*args, **kwargs):
        …

    return g
```

Dans d'autres langages : "annotation"

Ne pas confondre avec le *design pattern* du décorateur

Classeurs décorateur.ipynb

LUDIKSCIENCES

Décorateurs prédéfinis

@property
@x.setter, getter, deleter

Définir un attribut avec les méthodes pour le modifier, le supprimer, l'obtenir

getter, setter, deleter sont appelés des mutateurs

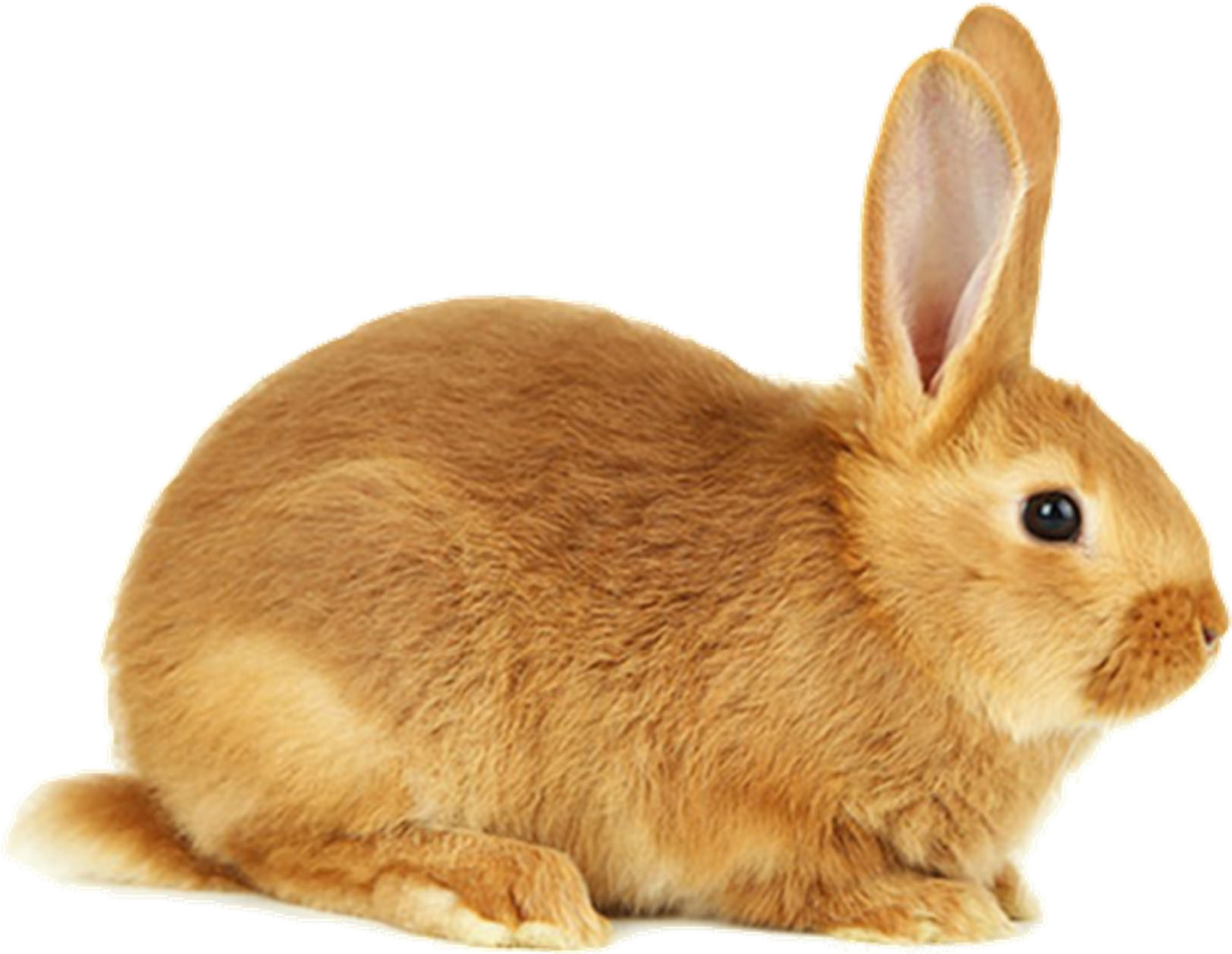
Ce dispositif est justifié si les mutateurs font quelque chose de plus que = ou del

La propriété est "cachée" par __ et reste manipulable par les mutateurs prévus par le concepteur.

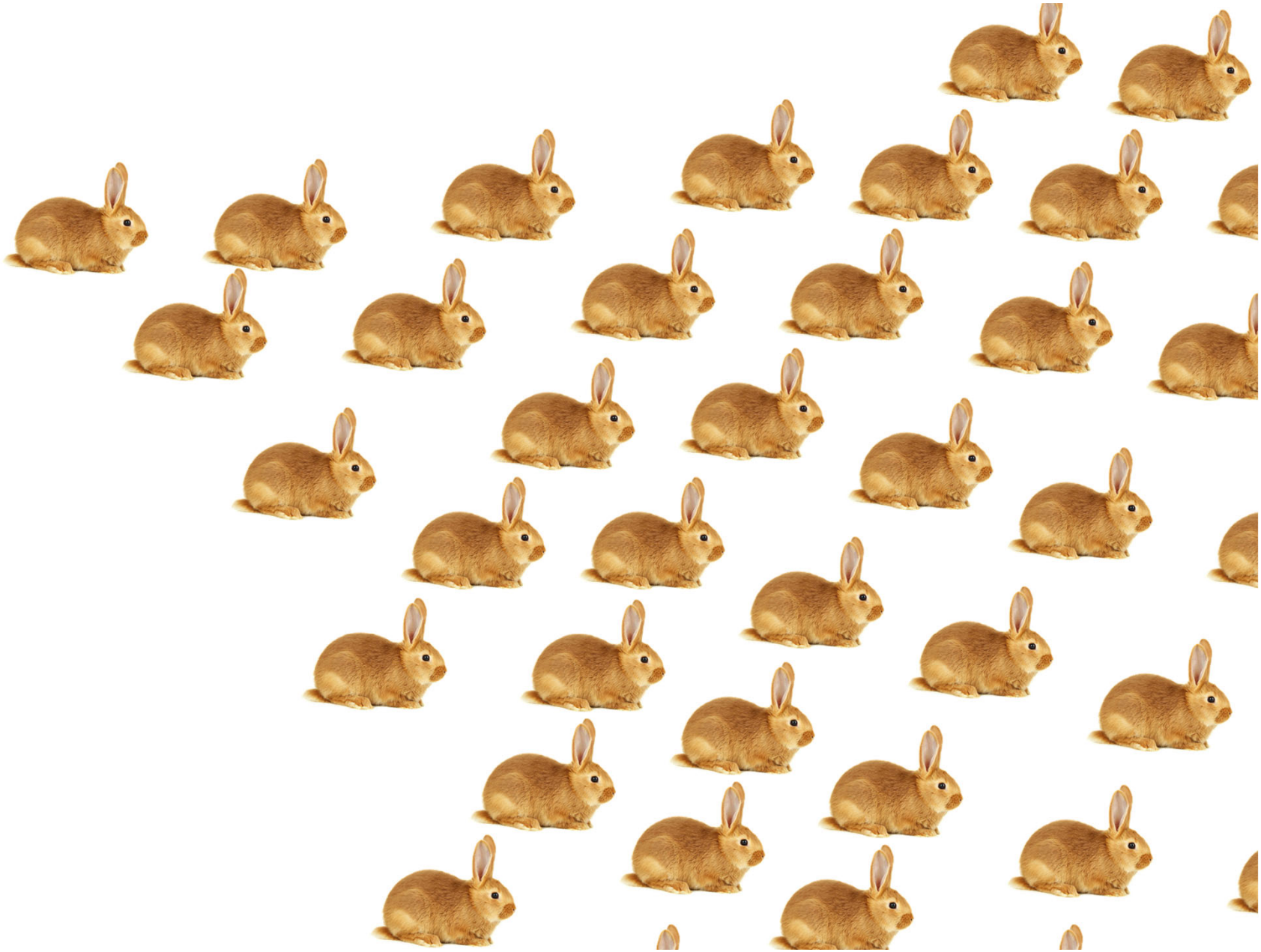
```
class UneClasse:
    def __init__(self,val):
        self.propriete = val

x = uneClasse(12)
print (x.propriete)
```

```
class UneClasse:
    def __init__(self,val):
        self.propriete = val

    @property
    def propriete(self):
        return self.__propriete

    @propriete.setter
    def propriete(self, val):
        self.__propriete = val

x = uneClasse(12)
print (x.propriete)
```

LUDIKSCIENCES



LUDIKSCIENCES



LUDIKSCIENCES

Exercice Fibonacci

Nombres de Fibonacci
0, 1, 1, 2, 3, 5, …
le suivant est la somme des 2 précédents
Modèle de l'évolution d'une population de lapins

Lister les 10 premiers nombres de Fibonacci à l'aide
- algorithme récursif fib_r(n)
- algorithme itératif fib_i(n)
- générateur fib_g()

Instrumenter vos algorithmes pour compter les opérations effectuées

Classeur Jupyter Fibonacci.ipynb

LUDIKSCIENCES