— Labo —

7 Rappels python (1 UC)

Sommaire

Objectifs

- ▷ Test : *if*, ==
- ▷ Boucle : *for*, *while*, *range*
- ▷ Tableau : parcours, *len*
- ▷ Récursivité : appel, condition d'arrêt
- ▷ Appel de fonctions : simple, imbriqué

7.1 Boucle

Calcul d'une moyenne

Ecrire un programme qui permette à l'utilisateur de calculer la moyenne de ses notes. Ce programme commencera par demander combien l'utilisateur a eu de notes durant le semestre. Il lira ensuite ces notes une à une pour faire le calcul de la moyenne. Après avoir lu toutes les notes, on affichera la moyenne calculée.

En python, on peut utiliser la fonction ***input*** *pour récupérer une chaîne de caractères tapée au clavier.*

```
1 ma_chaine = input('Taper une chaine: ')
2 print 'La chaine est : ', ma_chaine
```

7.2 Tableau

Recherche dans un tableau

Ecrire une fonction python qui recherche un entier x dans un tableau d'entiers A de longueur N et qui imprime tous les indices i tel que $Ai = x$.

7.3 Appel de fonctions

Considérons la bibliothèque de fonctions suivantes :

```
1  from math import *
2
3  def somme(x1,x2):
4          return x1+x2
5
6  def soustraction(x1,x2):
7          return x1-x2
8
9  def multiplication(x1,x2):
10         return x1*x2
11
12 def division(x1,x2):
13         return x1/x2
14
15 def racine(x):
16         return sqrt(x)
17
18 def puissance(x,exposant):
19         return x**exposant
20
21 def factorielle(x):
22         return fact(x)
23
24 def valeurAbsolue(x):
25         return abs(x)
26
27 def cosinus(x):
28         return cos(x)
29
30 def sinus(x):
31         return sin(x)
32
33 def moyenne(tab):
34         moy = 0.0
35         for i in range(len(tab)):
36                 moy = moy + tab[i]
37         moy  = division(moy,len(tab))
38         return moy
39
40 def logarithme(x):
41         return log(x)
```

Considérons la formule suivante :
formule de l'ecart type :
$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ où $\bar{x}$ est la moyenne des n échantillons du tableau x

1. Ecrire une fonction qui calcule $(a-b)^2$ en utilisant au maximum la bibliothèque de fonctions. Le prototype de la fonction demandée est le suivant :

   ```
   def fonction1(a_real, b_real) -> result_real
   ```

2. Ecrire une fonction qui calcule $\sum_{i=1}^{n}(x_i - \bar{x})^2$ où $\bar{x}$ est la moyenne des n échantillons du tableau x, en utilisant au maximum la bibliothèque de fonctions et `fonction1`.
   Le prototype de la fonction demandée est le suivant :

   ```
   def fonction2(tab_real) -> result_real
   ```

3. On se propose d'écrire la fonction qui calcule l'ecart type sur un échantillon de n élèments, en utilisant au maximum la bibliothèque de fonctions et `fonction2`. Le prototype de la fonction demandée est le suivant :

   ```
   def ecartType(tab_real) -> result_real
   ```

7.4 Récursivité

Repeat

1. Ecrire une fonction `repete( n,str)` renvoyant la chaîne de caractères *str* répétée n fois : `repete(3, "bla")` donne `blablabla`.
2. Ecrire une fonction `pyramide(n, s)` qui écrit sur la première ligne, 1 fois la chaîne s, sur la deuxième, 2 fois la chaîne s, et ainsi de suite jusque la dernière ligne, où il y aura n fois la chaîne s. Ainsi `pyramide(5, "bla")` donnera
   ```
   bla
   blabla
   blablabla
   blablablabla
   blablablablabla
   ```
3. Quand on lance `pyramide(n, s)`, combien y a-t-il d'appels à `pyramide` ? Combien y a-t-il d'appels à `repete` ? Pour vous aider à répondre dessiner l'arbre des appels pour $n = 3$.

Exercice supplémentaire 7.1. *Recherche dans un tableau par dichotomie*

Ecrire la fonction `rechercheDicho(A,x)` *qui recherche un entier x dans un tableau ordonné d'entiers A de longueur N en utilisant une recherche dichotomique.*

Exercice supplémentaire 7.2. *Valeur la plus proche*

Décrire une fonction récursive qui, étant donné un entier X, détermine la valeur la plus proche de X dans un tableau d'entiers.

7.5 QCM

1. Quel appel est correct pour cette fonction ?

```
def fonction(a,b,c,d=1):
      print a,b,c,d
```

   (a) `fonction(10,5,3)`
   (b) `fonction(a,b,c,d)`
   (c) `fonction(10)`
   (d) `fonction`

2. Combien d'appels à `f` sont effectués ?

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

f(5)
```

   (a) 10
   (b) 5
   (c) 1
   (d) 6

3. Qu'affiche ce code ?

```
i=1
x=2
while x<10:
      print i
```

   (a) 2
   (b) 111111111
   (c) 11111111
   (d) 1111111
   (e) rien
   (f) 11111111111111111111111 ...

8 Concepts fondateurs (1 UC)

Sommaire

8.1 Exercices préparatoires

8.1.1 Classe ou objet ?

Différencier les classes des objets (instances de classe) dans les propositions suivantes : Footballeur, Zidane, Basket, Sport, Voiture, Clio, Rouge, Bleu, Homme, Couleur, Mégane, Manaudou.

8.1.2 Attributs de classe

Les étudiants sont caractérisés par leur numéro d'étudiant, leur nom, leur prénom, leur date de naissance et leur adresse.

Identifier les attributs de la classe `Etudiant`.

8.2 Classe

Un complexe z est composé d'une partie réelle x et d'une partie imaginaire y, tel que $z = x + i * y$. Nous proposons ici d'implémenter une classe `Complex`.

8.2.1 Définition

- ▷ Définir une classe `Complex`
- ▷ Commenter la classe `Complex`

En python, si la première ligne suivant l'instruction `class` est une chaîne de caractères, celle-ci sera considérée comme un commentaire et incorporée automatiquement dans un dispositif de documentation des classes qui fait partie intégrante de Python. Prenez donc l'habitude de toujours placer une chaîne décrivant la classe à cet endroit.

8.2.2 Constructeur

- ▷ Si cela n'a pas déjà été fait (pas bien), ajouter un constructeur
- ▷ Ajouter deux attributs `x` et `y`
- ▷ Ajouter au constructeur la possibilité de spécifier les valeurs de `x` et `y`

8.3 Objet

8.3.1 Instanciation

Nous venons de définir une classe `Complex`. Nous pouvons dès à présent nous en servir pour créer des objets de ce type, par instanciation.

- ▷ Créer un objet `c1`, instance de la classe `Complex`.

8.3.2 Identité

- ▷ Effectuer les manipulations avec notre nouvel objet `c1` suivantes :

```
>>> print c1.__doc__
>>> print c1
```

Que remarquez vous ?

8.3.3 Comportement

- ▷ Affecter la valeur 3 à l'attribut `x` et 4 à l'attribut `y` de l'objet `c1`
- ▷ Les fonctions peuvent utiliser des objets comme paramètres.
  Définir une fonction `affiche_complex` qui affiche les valeurs du complex passé en paramétre.
  Testez la avec l'objet `c1`.
- ▷ Créer une méthode (fonction dans la classe) `show` qui affiche les attributs de la classe `Complex`.
  Testez la avec l'objet `c1`.

8.3.4 Manipulation de plusieurs objets

- ▷ Créer une fonction `compareComplex` qui renvoie vrai si chaque attribut de deux objets sont identiques.
- ▷ Créer un objet `c2` de type `Complex`, tester `compareComplex`

8.3.5 Manipulation de plusieurs objets Python

- ▷ Affecter la valeur 3 à l'attribut `x` et 4 à l'attribut `y` de l'objet `c2`.
- ▷ Ajouter une méthode `clone` qui recopie un objet de type `Complex`.
- ▷ Effectuer les manipulations suivantes :

```
>>> print (c1 == c2)
>>> c1=c2; print (c1 == c2)
>>> c3=c1.clone(); print (c1 == c3)
```

  Que remarquez vous ?

Exercice supplémentaire 8.1. *Classe Livre*

1. *Ecrire une classe* `Livre` *avec les attributs suivants :*
   - ▷ `titre` *: Le titre du livre,*
   - ▷ `auteur` *: L'auteur du livre,*
   - ▷ `prix` *: Le prix du livre,*
   - ▷ `annee` *: L'année du livre.*
2. *La classe* `Livre` *doit pouvoir être construites avec les possibilités suivantes :*

- ▷ `Livre()`,
- ▷ `Livre(titre)`,
- ▷ `Livre(titre, auteur)`,
- ▷ `Livre(titre, auteur, prix)`,
- ▷ `Livre(titre, auteur, prix, annee)`

3. *Instancier deux livres de votre choix*
4. *Créer une fonction qui renvoie vrai si le titre de l'instance passée en paramètre correspond au titre demandé.*

Exercice supplémentaire 8.2. *Classe* `Time`

- ▷ *Définir une nouvelle classe* `Time`*, qui nous permettra d'effectuer toute une série d'opérations sur des instants, des durées, etc.*
- ▷ *Ajouter les attributs* `pays`*,* `heure`*,* `minute` *et* `seconde`*.*
- ▷ *Créer un objet* `instant` *qui contient une date particulière.*
- ▷ *Ecrivez une fonction* `affiche_heure()` *qui permet de visualiser le contenu d'un objet de classe* `Time` *sous la forme conventionnelle "heure :minute :seconde".*
- ▷ *Appliquée à l'objet* `instant` *la fonction* `affiche_heure()`
- ▷ *Encapsuler la fonction* `affiche_heure()` *dans la classe* `Time` *(méthode* `affiche`*)*
- ▷ *Instancier un objet* `maintenant`
- ▷ *Tester la méthode*

Une fonction qui est ainsi encapsulée dans une classe s'appelle une méthode.

Exercice supplémentaire 8.3. *Pour obtenir les nombres premiers compris entre 2 et un certain entier N on va construire une liste chaînée d'objets appelés des MangeNombres, chacun comportant deux variables d'instance : un nombre premier et une référence sur le MangeNombres suivant de la liste.*

Le comportement d'un MangeNombres se réduit à l'opération "manger un nombre". Le MangeNombres mp associé au nombre p mange les multiples de p : si on lui donne à manger un nombre q qui n'est pas multiple de p, mp le donne à manger à son MangeNombres suivant, s'il existe. Si mp n'a pas de suivant, celui-ci est crée, associé à q. La création d'un MangeNombres mp provoque l'affichage de p. Définissez la classe MangeNombres et écrivez un programme affichant les nombres premiers entre 2 et N.

Exercice supplémentaire 8.4. *classe Ensemble*

1. *Créez une classe Ensemble pour représenter des sous-ensembles de l'ensemble des entiers compris entre 0 et N - 1, ou N est une constante donnée, avec les méthodes suivantes :*
   - ▷ *cardinal() : nombre d'éléments de l'ensemble,*
   - ▷ *contient(i) : test d'appartenance,*
   - ▷ *ajout(i) : ajout d'un élément à l'ensemble,*
   - ▷ *toString() : transformation de l'ensemble en une chaîne de caractères de la forme {5 ; 6 ; 10 ; 15 ; 20 ; 21 ; 22 ; 30}*

   Les éléments d'un ensemble seront mémorisés dans une variable d'instance de type tableau. Pour essayer la classe Ensemble, écrivez un programme déterminant les nombres différents contenus dans une suite de nombres lue sur l'entrée standard.
2. *Modifiez la classe précédente afin de permettre la représentation d'ensembles d'entiers quelconques, c'est- à-dire non nécessairement compris entre 0 et une constante N connue à l'avance. Faites en sorte que l'interface de cette classe soit identique à celle de la version précédente qu'il n'y ait rien à modifier dans les programmes qui utilisent la première version de cette classe.*

8.4 QCM

1. Comment créer une **méthode** `show()` qui affiche l'attribut `x` de la classe `B` ? _____

(a)
```
class B:
    def __init__(self):
          self.x = 3
    def show(b1):
          print b1.x
```

(b)
```
class B:
    def __init__(self):
          self.x = 3

def show():
    print x
```

(c)
```
class B:
    def __init__(self):
          self.x = 3
    def show(self):
          print self.x
```

(d)
```
class B:
    def __init__(self):
          self.x = 3

def show(b1):
    print b1.x
```

2. Que renvoie la fonction `compareD` ?

```
class D:
    def __init__(self,xinit=10):
         self.x = xinit

def compareD(d1,d2):
       return d1.x == d2.x

d10 = D(2)
d20 = D()
print compareD(d10,d20)
```

(a) Faux

(b) Vrai

(c) Une erreur

(d) Rien

3. Combien de fois *new C* va t'il être affiché ?

```
class C:
  def __init__(self):
       print "new  C"

i = 0
while i < 10:
     c1 = C()
     i = i + 1
```

(a) 0 fois

(b) 2 fois

(c) 10 fois

(d) 1 fois

4. Comment créer une instance de `A` ?

```
class A:
  def __init__ (self):
          print "toto"
```

(a) `a1 = A()`

(b) `a1 = A(10)`

(c) `a1 = A`

(d) `A() = a1`

9 Encapsulation (1 UC)

Sommaire

Dans les exercices qui suivent, utilisez les règles d'encapsulation vues en cours (contrôle d'accés, mutateurs, observateurs ...).

9.1 Domino

Définissez une classe `Domino` qui permette d'instancier des objets simulant les pièces d'un jeu de dominos.

1. Le constructeur de cette classe initialisera les valeurs des points présents sur les deux faces A et B du domino (valeurs par défaut = 0).
2. Deux autres méthodes seront définies :
   - ▷ une méthode `affiche_points()` qui affiche les points présents sur les deux faces
   - ▷ une méthode `valeur()` qui renvoie la somme des points présents sur les 2 faces.

Exemples d'utilisation de cette classe :

```
>>> d1 = Domino(2,6)
>>> d2 = Domino(4,3)
>>> d1.affiche_points()
 face A : 2 face B : 6
>>> d2.affiche_points()
 face A : 4 face B : 3
>>> print "total des points :", d1.valeur() + d2.valeur()
 15
>>> liste_dominos = []
>>> for i in range(7):
        liste_dominos.append(Domino(6, i))
>>> for j in  liste_dominos:
        j.affiche_points()
```

9.2 CompteBancaire

Définissez une classe `CompteBancaire`, qui permette d'instancier des objets tels que `compte1`, `compte2`, etc.

1. Le constructeur de cette classe initialisera deux attributs `nom` et `solde`, avec les valeurs par défaut `'Dupont'` et `1000`.
2. Trois autres méthodes seront définies :
   ▷ `depot(somme)` permettra d'ajouter une certaine somme au solde
   ▷ `retrait(somme)` permettra de retirer une certaine somme du solde
   ▷ `affiche()` permettra d'afficher le nom du titulaire et le solde de son compte.

Exemples d'utilisation de cette classe :

```
>>> compte1 = CompteBancaire('Duchmol', 800)
>>> compte1.depot(350)
>>> compte1.retrait(200)
>>> compte1.affiche()
Le solde du compte bancaire de Duchmol est de 950 euros.
>>> compte2 = CompteBancaire()
>>> compte2.depot(25)
>>> compte2.affiche()
Le solde du compte bancaire de Dupont est de 1025 euros.
```

9.3 Voiture

Définissez une classe `Voiture` qui permette d'instancier des objets reproduisant le comportement de voitures automobiles.

1. Le constructeur de cette classe initialisera les attributs d'instance suivants, avec les valeurs par défaut indiquées :
   `marque = 'Ford', couleur = 'rouge', pilote = 'personne', vitesse = 0.`
   Lorsque l'on instanciera un nouvel objet `Voiture()`, on pourra choisir sa marque et sa couleur, mais pas sa vitesse, ni le nom de son conducteur.
2. Les méthodes suivantes seront définies :
   ▷ `choix_conducteur(nom)` permettra de désigner (ou de changer) le nom du conducteur

- ▷ `accelerer(taux, duree)` permettra de faire varier la vitesse de la voiture.
  La variation de vitesse obtenue sera égale au produit : `taux` * `duree`. Par exemple, si la voiture accélère au taux de $1,3m/s^2$ pendant 20 secondes, son gain de vitesse doit être égal à 26 m/s. Des taux négatifs seront acceptés (ce qui permettra de décélérer). La variation de vitesse ne sera pas autorisée si le conducteur est 'personne'.
- ▷ `affiche_tout()` permettra de faire apparaître les propriétés présentes de la voiture, c'est-à dire sa marque, sa couleur, le nom de son conducteur, sa vitesse.

Exemples d'utilisation de cette classe :

```
>>> a1 = Voiture('Peugeot', 'bleue')
>>> a2 = Voiture(couleur = 'verte')
>>> a3 = Voiture('Mercedes')
>>> a1.choix_conducteur('Romeo')
>>> a2.choix_conducteur('Juliette')
>>> a2.accelerer(1.8, 12)
>>> a3.accelerer(1.9, 11)
Cette voiture n'a pas de conducteur !
>>> a2.affiche_tout()
Ford verte pilotee par Juliette, vitesse = 21.6 m/s.
>>> a3.affiche_tout()
Mercedes rouge pilotee par personne, vitesse = 0 m/s.
```

10 Collaboration et héritage (1 UC)

Sommaire

10.1 Collaboration

10.1.1 Exercices préparatoires

Pour définir un rectangle, nous spécifions sa largeur, sa hauteur et précisons la position du coin supérieur gauche. Une position est définie par un point (coordonnées x et y).
Quelle(s) classe(s) et quel(s) attribut(s) peuvent être définis pour représenter ces notions ?
A quoi pourra servir le constructeur de chaque classe ?

10.1.2 Collaboration entre objets

Objets composés d'objets

- ▷ Définir la classe `Point` contenant les attributs `x` et `y` (coordonnées)
- ▷ Définir la classe `Rectangle`
- ▷ Instancier un objet `rectangle1` de largeur 50, de hauteur 35, et dont le coin supérieur gauche se situe au coordonnée (12,27)

Constructeur / Destructeur

- ▷ Rafiner au constructeur de la classe `Point` et de la classe `Rectangle` les possibilités que vous jugerez utiles (valeurs par défaut ...)
- ▷ Préciser le destructeur de la classe `Point` et de la classe `Rectangle`

▷ Tester le destructeur de la classe `Rectangle`

Objets comme valeurs de retour d'une fonction Nous avons vu plus haut que les fonctions peuvent utiliser des objets comme paramètres. Elles peuvent également transmettre une instance comme valeur de retour.

▷ Définir la fonction `trouveCentre()` qui est appelée avec un argument de type `Rectangle` et qui renvoie un objet `Point`, lequel contiendra les coordonnées du centre du rectangle.
▷ Tester cette fonction en utilisant l'objet `rectangle1` défini en question 10.1.2.

Modifier un objet Modifier la taille d'un rectangle (sans modifier sa position), en réassignant ses attributs hauteur (hauteur actuelle +20) et largeur (largeur actuelle -5).

10.2 Héritage

10.2.1 Exercices préparatoires

Gestion d'heures complémentaires Chaque enseignant de l'université effectue un certain nombre d'heures d'enseignement dans une année. Suivant le statut de l'enseignant, un certain nombre de ces heures peut-être considéré comme complémentaire. Les heures complémentaires sont payées séparément à l'enseignant. Les volumes horaires sont exprimés en heures entières et le prix d'une heure complémentaire est de 10 Euros.
Le nom et le nombre d'heures total d'un enseignant sont fixés à sa création, puis seul le nom peut être librement consulté (méthode `nom()`).
D'autre part on veut pouvoir librement consulter un enseignant sur son volume d'heures complémentaires (méthode `hc()`) et sur la rétribution correspondante (méthode `retribution()`).
Il y a deux types d'enseignants :

▷ les intervenants extérieurs : toutes les heures effectuées sont complémentaires,
▷ les enseignants de la fac : seules les heures assurées au delà d'une charge statutaire de 192h sont complémentaires.

Questions :

1. Sans écrire de code, modéliser les enseignants : quelles sont les classes ? où sont implémentées les méthodes ? lesquelles sont nouvelles, redéfinies ?

2. Comment modifier le modèle pour y introduire les étudiants de troisième cycle qui assurent des enseignements : toutes les heures effectuées sont complémentaires mais dans la limite de 96 heures.

10.2.2 Héritage simple et multiple

- ▷ Définir une classe `Mammifere`, qui contiendra un ensemble de caractéristiques propres à ce type d'animal (nom, âge, nombre de dents ...).
- ▷ A partir de cette classe, nous pourrons alors dériver une classe `Primate`, une classe `Rongeur`, et une classe `Carnivore`, qui hériteront toutes les caractéristiques de la classe `Mammifere`, en y ajoutant leurs spécificités.
- ▷ Au départ de la classe `Carnivore`, dériver une classe `Belette`, une classe `Loup` et une classe `Chien`, qui hériteront encore une fois toutes les caractéristiques de la classe parente avant d'y ajouter les leurs.
- ▷ Définir une classe `Herbivore` dérivé de la classe `Primate`
- ▷ Définir une classe `Omnivore`, qui hérite de la classe `Carnivore` et de la classe `Herbivore`

10.2.3 Polymorphisme

- ▷ Ecrivez une classe `Forme` qui contiendra une méthode `calculAire`.
- ▷ Faites-en hériter la classes `Carre` contenant un attribut `cote`
- ▷ idem pour la classe `Cercle` contenant un attribut `rayon`.
- ▷ Rédéfinir la méthode `calculAire` pour les classes `Carre` et `Cercle`
- ▷ Définir une fonction `AireTotal` qui à partir d'un tableau de `Forme` calcul l'aire totale

Exercice supplémentaire 10.1. *écrire le code python de l'exercice de préparation sur la gestion d'heures complémenatires*

10.3 QCM

1. Qu'affiche ce code ?

```
class A:
            def __init__(self):
                  print "Creation d'un objet A"
class B(A):
            def __init__(self):
                   A.__init__(self)
                   print "Creation d'un objet B"
class C(B):
            def __init__(self):
                   B.__init__(self)
                   print "Creation d'un objet C"
c1 = C()
```

   (a) Creation d'un objet A , Creation d'un objet B , Creation d'un objet C

   (b) Creation d'un objet C , Creation d'un objet B , Creation d'un objet A

   (c) Creation d'un objet C , Creation d'un objet A , Creation d'un objet B

   (d) Creation d'un objet C

2. Qu'affiche ce code ?

```
class A:
              def __init__(self, xInit = 1):
                           self.x = xInit
class B(A):
             def __init__(self, xInit=0, yInit=0)
                           A.__init__(self, xInit)
                           self.y = yInit
v1 = B()
v2 = B(5, 4)
print v1.x, v1.y, v2.x, v2.y
```

   (a) 0 0 5 4

   (b) 1 0 0 0

   (c) Une erreur

   (d) 0 0 0 0

   (e) 5 4 5 4

3. Qu'affiche ce code ?

```
class A:
            def __init__(self, x):
                        self.x = x
class B(A):
            def __init__(self):
                        A.__init__(self)
b = B(5)
print b.x
```

(a) x

(b) 0

(c) Une erreur

(d) 5

4. Qu'affiche ce code ?

```
class A:
            def __init__(self, xInit = 0.0):
                        self.__x = xInit
a = A(5.0)
print a.__x
```

(a) 5.0

(b) __x

(c) Une erreur

(d) 0.0

5. Qu'affiche ce code ?

```
class A:
......
            def show(self):
                        print "Je suis un A"
class B(A):
......
            def show(self):
                        print "Je suis un B"
a = B()
a.show()
```

(a) Je suis un B

(b) Je suis un B, Je suis un A

(c) Je suis un A

(d) Une erreur

(e) Je suis un A, Je suis un B

11 Pour aller plus loin

Sommaire

11.1 Abstraction (1 UC)

11.1.1 Exercices préparatoires

Vie parisienne Toute sa vie, voici la journée d'un parisien :

- ▷ métro
- ▷ boulot
- ▷ dodo

Dans le métro, tous les parisiens ont le même comportement.
Dans leur lit, tous les parisiens ont le même comportement.
Le boulot différe pour chacun d'entre eux (parisien commerçant, parisien cadre ...) .
Sans écrire de code, comment représenter ces informations ?

11.1.2 Classe abstraite / méthode abstraite

L'objectif est de définir une classe abstraite destinée à gérer un tableau trié d'éléments et comportant une méthode abstraite `plusGrand(self,a,b)`. Cette méthode devra comparer deux éléments. Pour gérer un tableau trié d'objets d'un certain type, il faudra étendre la classe abstraite en une classe définissant la méthode `plusGrand(self,a,b)` pour le type d'objets en question.

On construira :

1. Une classe abstraite `TableauTrieAbstrait` gérant un tableau d'éléments qui reste toujours trié par ordre croissant par rapport à la relation définie par une méthode abstraite `plusGrand`. Cette classe devra contenir au moins
   - ▷ une méthode abstraite `plusGrand(self,a,b)`

- ▷ une méthode `inserer(self,element)` pour insérer une instance d'élément dans le tableau (il faut faire attention à l'ordre pour ne pas l'insérer n'importe oú)

2. Une classe `TableauTrieEntiers` qui étend la classe `TableauTrieAbstrait` ; cette classe est destinée à gérer un tableau trié d'entier. Il faut essentiellement y définir la méthode `plusGrand(self,a,b)` pour des entiers.
3. Une classe, `TableauTrieChaines` qui étend la classe `TableauTrieAbstrait` ; cette classe est destinée à gérer un tableau trié de chaîne de caractères. Il faut essentiellement y définir la méthode `plusGrand(self,a,b)` en se basant sur le nombre de caractères.

Exercice supplémentaire 11.1. *Vie parisienne ... suite*
Implémenter les classes de l'exercice préparatoire sur la vie parisienne.
Ajouter les informations suivantes :

- ▷ *la jeunesse de tous les parisiens est la même (cris de bébés, croissance, études)*
- ▷ *à la fin de sa vie, le parisien fait le bilan de sa vie (le commercial compte son pécule, le chercheur regarde combien il avait trouvé de réponses et de problèmes)*
- ▷ *la mort de tous les parisiens est la même (un dernier souffle)*

Exercice supplémentaire 11.2. *Boulangerie*

Un simulateur fait "vivre" trois personnages économiques que sont :

1. *Le Paysan*
2. *Le Meunier*
3. *Le Boulanger*

Les classes à construire sont :

1. *Le Paysan*
   - ▷ *Mange du pain pour vivre*
   - ▷ *Produit du blé*
   - ▷ *Livre du blé à un menier*
   - ▷ *Reçoit du pain*
2. *Le Meunier*
   - ▷ *Mange du pain pour vivre*
   - ▷ *Reçoit du blé*

- ▷ *Produit de la farine en consommant du blé*
- ▷ *Livre de la farine à un boulanger*
- ▷ *Reçoit du pain*

3. *Le Boulanger*
   - ▷ *Mange du pain (de son propre stock) pour vivre*
   - ▷ *Reçoit de la farine*
   - ▷ *Fabrique du pain en consommant de la farine*
   - ▷ *Livre du pain aux meuniers et aux paysans (en fait, à tous ceux qui peuvent en recevoir)*

- ▷ *Etablir les trois "identités de classe" des personnages, en décrivant les attributs nécessaires.*
- ▷ *Peut-on trouver un principe "plus abstrait" qui permette de factoriser certains éléments ?*
  $\rightarrow$ *Etablir cette classe.*
- ▷ *Identifier les comportements communs et les résoudre dans cette classe.*
- ▷ *Modifier et terminer les classes personnages pour tenir compte de cette factorisation.*
- ▷ *Evaluer une manière "intéressante" de procéder aux échanges de matière.*
- ▷ *Ecrire une petite classe "enveloppante" qui mette en scéne trois personnages et les fassent "vivre" dans une boucle en mode "tour par tour".*

▷ `cellules` est un conteneur de cellules.

La méthode `update()` du terrain lance cycliquement la méthode `update()` du soleil puis les méthodes `update()` de toutes les cellules.

En plus de la méthode `update()`, il y a les méthodes suivantes :
▷ `celluleDessus()`, `celluleDessous()`, `celluleGauche()`, `celluleDroite()` renvoient les cellules se situant respectivement au dessus, au dessous, à gauche et à droite d'une cellule donnée. Ceci permettra aux cellules de calculer leur température. Ces méthodes prennent en compte le fait que le terrain reboucle sur lui même en haut et en bas (par exemple, la cellule de gauche d'une cellule au bord correspond à la cellule au bout à droite de la même ligne).

12.1.3 Détails sur les cellules

Une cellule possède les attributs suivants :
▷ `terrain` est le terrain qui contient la cellule
▷ `temperature` est la température de la cellule
▷ `coefDiffusion` est le coefficient de diffusion de cette température (permettant de caractériser par exemple le fait que l'eau se réchauffe moins vite que la terre).
▷ `x et y` représentent la position des cellules sur le terrain. Il s'agit du numéro de colonne et de ligne.
▷ `coefPerte` représente le fait que la température à tendance à descendre à cause de l'atmosphère.
▷ `couleur` représente la couleur de la cellule.

La méthode `update()` des cellules calcule la température grâce à la formule suivante :

$$T_x = max(TMIN, \left\{ \{[\sum_{x_i \in Voisins} \frac{T(x_i) - T_x}{nbVoisins}] * coefD_x\} + T_x - \{coefP_x * (-TMIN + T_x)\} \right\})$$

$coefD_x$ et $coefP_x$ étant respectivement les coefficients de diffusion et de perte de la cellule x. Les voisines sont ici au nombre de quatre et correspondent aux cellules de droite, de gauche du dessous et du dessus.

Cette méthode a également en charge de calculer la `couleur` de la cellule, allant de 0 (froid) à 255 (chaud) (méthode `normalize`).

12.1.4 Détails sur le soleil

La méthode `update()` du soleil réchauffe la cellule avec une température max de `TSoleil`, par le biais de la formule suivante :

$$T_x = T_x + coefD_x * (TSoleil - T_x)$$

où T_x est la température de la cellule sur laquelle se trouve le soleil.

Le soleil a des coordonnées `x et y` sur le `terrain` (contenant le soleil).

12.1.5 Détails sur le simulateur

Le simulateur posséde une méthode `run` qui :

1. instancie un soleil avec une température de 500
2. lui donne une position aux coordonnnées (50,50)
3. instancie un terrain de taille 100*100
4. lui donne l'instance du soleil
5. effectue une boucle infinie qui demande la mise à jour du terrain

12.2 Travail demandé

Proposer une implémentation de chacune des classes précédentes respectant le sujet. Instancier ensuite un simulateur. Vous respecterez les principes objets, plus particulièrement sur la visibilité des attributs.

12.2.1 La classe `Cellule`

12.2.2 La classe `Terrain`

12.2.3 La classe `Soleil`

12.2.4 La classe `Simulateur`

Instancier ensuite un simulateur et lancer la méthode `run`

13 Correction : GAIA

```
1  # -*- coding: utf-8 -
2
3
4  ######################################################
5  #### auteur : C.Buche
6  #### date   : 23/03/09
7  ######################################################
8
9  import math
10
11 class Terrain:
12         pass
13
14 ######################################################
15 class Cellule :
16         """ Une Cellule possede des caracteristiques physiques (temperature, coeff. de diffusion, coeff. de perte)      """
17
18                 # ------------ constructeurs et destructeurs ------------
19         def __init__(self,terrain=Terrain(),x=0,y=0,temperature=0,coefDiffusion=0.7,coefPerte=0.001,couleur=0):
20                 print "creation d'une Cellule"
21                 self.__x = x
22                 self.__y = y
23                 self.__temperature = temperature
24                 self.__coefDiffusion = coefDiffusion
25                 self.__coefPerte = coefPerte
26                 self.__terrain = terrain
27                 self.__couleur = couleur
28
29         def __del__(self):
30                 print "destruction d'une Cellule"
31
32                 # ------------ accesseurs et mutateurs ------------
33         def getX(self):
34                 return self.__x
35
36         def getY(self):
37                 return self.__y
38
39         def setPosition(self,x,y):
40                 self.__x = x
41                 self.__y = y
42
```

```
43          def getTemperature(self):
44                  return self.__temperature
45
46          def setTemperature(self,t):
47                  self.__temperature = t
48
49          def setTerrain(self,t):
50                  self.__terrain = t
51
52          def getTemperature(self):
53                  return self.__temperature
54
55          def getCoefDiffusion(self):
56                  return self.__coefDiffusion
57
58          def getCouleur(self):
59                  return self.__couleur
60
61                  # ------------ methodes ------------
62          def __normaliser(self,x): ## exemple de fonction de normalisation
63                  pente = 0.5
64                  centre =  self.__terrain.getTemperatureMoyenne()
65                  y = 255. / (1 + math.exp(1-(x-centre)*pente) )
66                  y = int(y)
67                  return y
68
69          def update(self):
70                  ##  calcul de la nouvelle temperature
71                  min = self.__terrain.getTMIN();
72                  dTempDiff =  self.__terrain.celluleDessous(self).getTemperature() - self.__temperature;
73                  dTempDiff += self.__terrain.celluleDessus(self).getTemperature() - self.__temperature;
74                  dTempDiff += self.__terrain.celluleGauche(self).getTemperature() - self.__temperature;
75                  dTempDiff += self.__terrain.celluleDroite(self).getTemperature() - self.__temperature;
76                  dTempDiff *= self.__coefDiffusion;
77
78                  dTempPerte = self.__coefPerte * (self.__temperature - min);
79
80                  self.__temperature = dTempDiff + self.__temperature - dTempPerte;
81
82                  if(self.__temperature < min ):
83                          self.__temperature = min
84
85                  ## tranformation temperature vers couleur
86                  self.__couleur = self.__normaliser(self.__temperature)
87
```

```
88                  return self.__temperature;
89
90  ######################################################
91  class Terrain :
92          """ Le terrain est compose de cellules  """
93
94                  # ------------ constructeurs et destructeurs ------------
95          def __init__(self,nbcelluleX=0,nbcelluleY=0,tMIN=10,tMOY=100):
96                  print "creation d'un Terrain"
97                  self.__nbcelluleX = nbcelluleX
98                  self.__nbcelluleY = nbcelluleY
99                  self.__tMIN=tMIN
100                 self.__temperatureMoyenne=tMOY
101                 self.__cellules = []
102                 self.__soleil = None
103                 for i in range(self.__nbcelluleX):
104                         tab = []
105                         for j in range(self.__nbcelluleY):
106                                 tab.append(None)
107                         self.__cellules.append(tab)
108
109                 for i in range(self.__nbcelluleX):
110                         for j in range(self.__nbcelluleY):
111                                 self.__cellules[i][j] = Cellule()
112                                 self.__cellules[i][j].setPosition(i,j)
113                                 self.__cellules[i][j].setTerrain(self)
114
115         def __del__(self):
116                 print "destruction d'un Terrain"
117                 for i in range(self.__nbcelluleX):
118                         for j in range(self.__nbcelluleY):
119                                 del self.__cellules[i][j]
120
121                 # ------------ accesseurs et mutateurs ------------
122         def setSoleil(self,s):
123                 self.__soleil = s
124
125         def getTMIN(self):
126                 return self.__tMIN
127
128         def getTemperatureMoyenne(self):
129                 return self.__temperatureMoyenne
130
131
132         def getnbcelluleY(self):
```

```
133             return self.__nbcelluleY
134
135         def getnbcelluleX(self):
136             return self.__nbcelluleX
137
138             # ------------ methodes ------------
139         def update(self):
140             self.__soleil.update()
141             for i in range(self.__nbcelluleX):
142                 for j in range(self.__nbcelluleY):
143                     self.__cellules[i][j].update()
144
145         def getCellule(self, x, y):
146             if x < 0 :
147                 x = self.__nbcelluleX - 1
148             if x >= self.__nbcelluleX:
149                 x = 0
150             if y < 0:
151                 y = self.__nbcelluleY - 1
152             if y >= self.__nbcelluleY:
153                 y = 0
154             return self.__cellules[x][y]
155
156
157         def celluleDessous(self,cellule):
158             x = cellule.getX()
159             y = cellule.getY()
160             return self.getCellule(x, y + 1)
161
162         def celluleDessus(self,cellule):
163             x = cellule.getX()
164             y = cellule.getY()
165             return self.getCellule(x, y - 1)
166
167         def celluleGauche(self,cellule):
168             x = cellule.getX()
169             y = cellule.getY()
170             return self.getCellule(x-1, y)
171
172         def celluleDroite(self,cellule):
173             x = cellule.getX()
174             y = cellule.getY()
175             return self.getCellule(x+1, y)
176
177
```

```
178 ######################################################
179 class Soleil :
180         """ Le Soleil rechauffe la cellule """
181 
182                 # -------------- constructeurs et destructeurs --------------
183         def __init__(self,temperature=0):
184                 print "creation d'un Soleil"
185                 self.__x = 0
186                 self.__y = 0
187                 self.__temperature = temperature
188 
189         def __del__(self):
190                 print "destruction d'un Soleil"
191 
192                 # -------------- accesseurs et mutateurs --------------
193         def setPosition(self,x,y):
194                 self.__x = x
195                 self.__y = y
196 
197         def setTerrain(self,t):
198                 self.__terrain = t
199 
200         def getTerrain(self):
201                 return self.__terrain
202 
203         def getX(self):
204                 return self.__x
205 
206         def getY(self):
207                 return self.__y
208 
209                 # -------------- methodes --------------
210         def update(self):
211                 cellule = self.__terrain.getCellule(self.__x,self.__y);
212                 Tx = cellule.getTemperature();
213                 coefDx = cellule.getCoefDiffusion();
214                 cellule.setTemperature(Tx+coefDx*(self.__temperature-Tx))
215 
216 
217 ######################################################
218 class Simulateur :
219         """ le simulateur execute regulierement la methode update() du terrain  """
220 
221                 # -------------- constructeurs et destructeurs --------------
222         def __init__(self):
```

```
223                 print "creation d'un Simulateur"
224
225         def __del__(self):
226                 print "destruction d'un Simulateur"
227
228                 # ------------ methodes ------------
229         def run(self):
230                 s = Soleil(500)
231                 s.setPosition(50,50)
232                 t = Terrain(20,20,10,100)
233                 t.setSoleil(s)
234                 s.setTerrain(t)
235                 while(True):
236                         t.update()
237
238
239 ##################### Programme principal ########
240 # s = Simulateur()
241 # s.run()
242 ###################################################
243
244
245 ############## Partie graphique #################
246 from Tkinter import *
247
248 class TerrainGraphique(Frame,Terrain):
249         """ Classe representant le terrain en 2D """
250         def __init__(self, master,nbcelluleX=0,nbcelluleY=0,tMIN=10,tMOY=100):
251                 Frame.__init__(self, master)
252                 master.title("GAIA")
253                 self.pack()
254                 self.damier=Canvas(self,bg="black",height=nbcelluleY*10,width=nbcelluleX*10)
255                 self.damier.pack()
256
257                 Terrain.__init__(self,nbcelluleX,nbcelluleY,tMIN,tMOY)
258
259                 self.__rectangles = []
260                 for i in range(self.getnbcelluleX()):
261                         tab = []
262                         for j in range(self.getnbcelluleY()):
263                                 tab.append(None)
264                         self.__rectangles.append(tab)
265
266                 for i in range(self.getnbcelluleX()):
267                         for j in range(self.getnbcelluleY()):
```

```
268                                 self.__rectangles[i][j] = self.damier.create_rectangle(i*10,j*10,i*10+9,j*10+9)
269
270
271                 self.pal=['white','gray', 'green','yellow','gold','orange','pink','red']
272
273                 self.timer = 50
274
275         def setTimer(self,timer):
276                 self.timer = timer
277
278         def update(self):
279                 Terrain.update(self)
280
281                 for i in range(self.getnbcelluleX()):
282                         for j in range(self.getnbcelluleY()):
283                                 couleur = self.getCellule(i,j).getCouleur()
284                                 couleur = (len(self.pal)* couleur) / 255. -1
285                                 couleur = abs(int(couleur))
286                                 couleur = self.pal[couleur]
287                                 self.damier.itemconfigure(self.__rectangles[i][j],fill=couleur)
    ## changer les couleurs
288
289                 fen1.after(self.timer, self.update)
    ## rappel de update
290
291
292 ###################################################
293 flag = 0
294 def start_it():
295     "demarrage de l'animation"
296     global flag
297     flag = flag +1
298     if flag ==1:
299         t.update()
300
301 class SimulateurGraphique :
302         """ le simulateur execute la methode update() du terrain sur click """
303
304                 # ------------ constructeurs et destructeurs ------------
305         def __init__(self,timer=50):
306                 print "creation d'un Simulateur Graphique"
307                 self.timer = timer
308
309
310         def __del__(self):
```

```
311                 print "destruction d'un Simulateur Graphique"
312
313                 # ------------ methodes ------------
314         def run(self,s = Soleil(500)):
315                 s.setPosition(2,2)
316
317                 global t
318                 global fen1
319                 fen1 = Tk()
320
321                 t = TerrainGraphique(fen1,100,100,10,100)
322                 t.setTimer(self.timer)
323                 t.setSoleil(s)
324                 s.setTerrain(t)
325
326                 Button(fen1,text='Quitter',command=fen1.quit).pack(side=RIGHT)
327                 Button(fen1,text='Demarrer',command=start_it).pack(side=LEFT)
328                 fen1.mainloop()
329
330
331
332 ######################################################
333 class SoleilMouvant(Soleil) :
334         """ Le Soleil mouvant rechauffe la cellule """
335
336                 # ------------ constructeurs et destructeurs ------------
337         def __init__(self,temperature=0):
338                 Soleil.__init__(self,temperature)
339
340         def __del__(self):
341                 print "destruction d'un Soleil mouvant"
342
343
344                 # ------------ methodes ------------
345         def update(self):
346                 Soleil.update(self)
347                 t = self.getTerrain()
348                 sizeX = t.getnbcelluleX()
349                 sizeY = t.getnbcelluleY()
350
351                 x = self.getX() +1
352                 y =  ((math.cos((  (x - sizeX/2.) )/sizeX) * sizeY))*1.3 - (sizeY/1.5)
353                 y = int(y)
354                 #y = self.getY()
355                 if x > sizeX : x = 0
```

```
356                 if y > sizeY : y = 0
357                 self.setPosition(x,y)
358
359
360
361
362
363 ##################  Programme principal ########
364
365 timer = 5
366 s = SimulateurGraphique(timer)
367 soleil1 = SoleilMouvant(5000)
368 s.run(soleil1)
```

Références

[Booch, 1992] Booch, G. (1992). *Conception orientée objets et applications*. Addison-Wesley.

[Brooks, 1987] Brooks, F. (1987). No silver bullet - essence and accidents of software engineering. *IEEE Computer*, 20(4) :10–19.

[Peter, 1986] Peter, L. (1986). *The peter pyramid*. New York, NY :William Morrow.