MINESnancy

Programmation Objets

Pierre-Etienne Moreau
Guillaume Bonfante

Retour sur le TD précédent

Problem 22

score(‘COLIN’) = 3 + 15 + 12 + 9 + 14 = 53

Comment associer : ‘A’ à 1, ‘B’ à 2, etc. ?

Problem 22

```
def indice_lettre(x):
    lettres = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    i = 1
    for e in lettres:
        if e == x:
            return i
        else:
            i += 1
return 0
```

Complexité ?

Problem 22

```
def indice_lettre(x):
    ...
    return ...
```

Peut-on avoir une complexité en O(1) ?

Scrabble

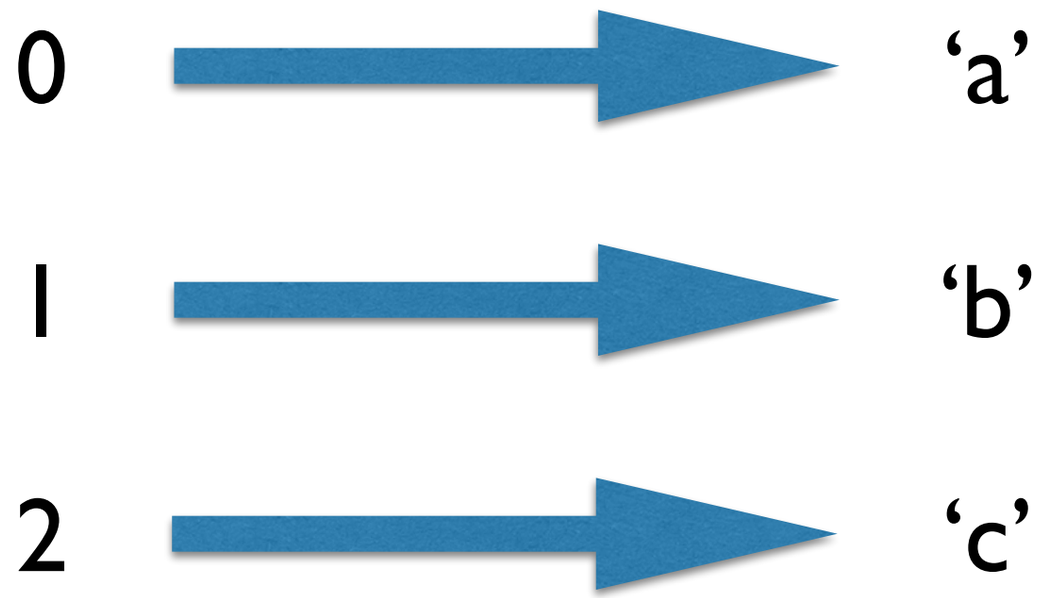
- A,E,I,L,N,O,R,S,T,U : 1 point
- D,G,M : 2 points
- B,C,P : 3 points
- F,H,V : 4 points
- J,Q : 8 points
- K,W,X,Y,Z : 10 points

```
def score(e):
    if e in 'aeilnorstu':
        return 1
    if e in 'dgm':
        return 2
    if e in 'bcp':
        return 3
    if e in 'fhv':
        return 4
    if e in 'jq':
        return 8
    if e in 'kwxyz':
        return 10
    return 0
```

Peut-on avoir une complexité en O(1) ?

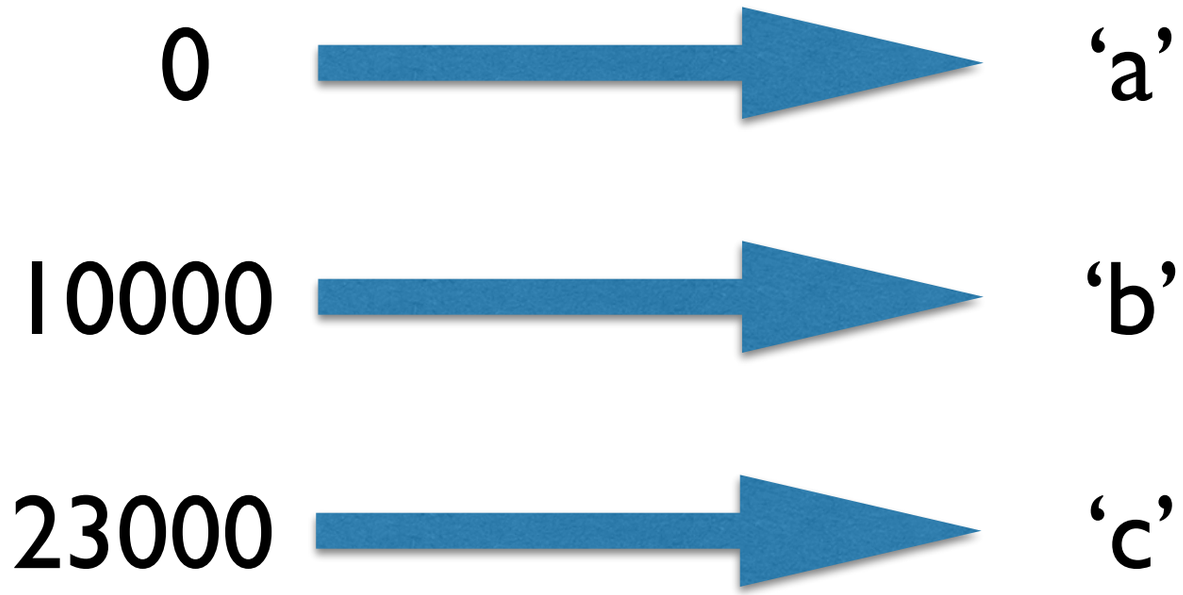
Dictionnaire

- **la liste** `u=['a','b','c']` **est une** association
- **qui associe :**

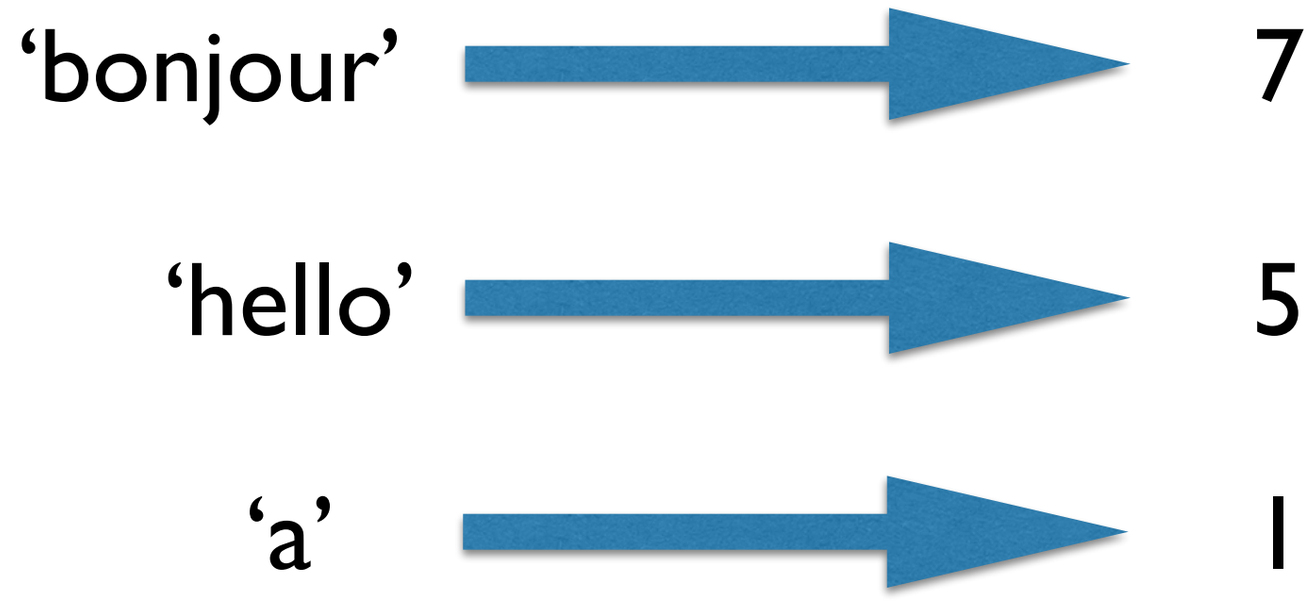


u[2] **vaut** 'c'

Comment créer l'association ?



Comment créer l'association ?



Dictionnaire

- construction : `d = {'bonjour':7, 'hello':5, 'a':1}`
- opérations :
  - `d['hello']`
  - `len( d )`
  - `'b' in d`
  - `d['b'] = 1`

dictionnaire : ensemble **de** clés, valeurs

Attention : les clés **doivent être** immuables
(entier, chaine, tuple)

Problem 22

```
d = { 'A':1, 'B':2, …, 'Z':26 }

def indice_lettre(x):

    return d[x]
```

Programmation Orientée Objet

Parmi les valeurs manipulées par un langage, certaines **sont des** objets

Objet

- **Correspond à un** concept
- **Possède une** structure interne
- **Répond à des** messages **(propose des** méthodes**)**

En Python

- **Toutes les** valeurs **sont des** objets
- `"hello"` **est un objet**
- `(2,4,8)` **est un objet**
- `3` **est un objet**

Méthodes

- Chaque objet propose un ensemble de méthodes
- Une méthode est une fonction applicable sur un objet
- **Exemple :** `'ab'.replace('a','c')`
- `replace` **est le nom de la méthode**
- `'ab'` **est appelé** receveur

Nouveau type d'objets

- L'utilisateur peut définir de nouvelles catégories d'objets
- **Cela s'appelle une** classe
- **Une classe définit**
  - **la** structure interne **des objets**
  - **les** méthodes **qu'offrent les objets**

Exemple : Counter

- Quels sont les services offerts par un compteur ?
- Quelle information est stockée dans le compteur ?

Définition de la classe

```
class Counter:
    def __init__(self):
        self.value = 0

c = Counter()
print(c)
```

Docstring

```
class Counter:
 def __init__(self):
     '''Create a Counter'''
     self.value = 0


c = Counter()
print(c)
```

Définition de la classe

```
class Counter:

  def __init__(self):

    self.value = 0


def increment(self):

  self.value += 1


def reinit(self):

  self.value = 0
```

```
def toString(self):

  return f"my value is: {self.value}"


# utilisation de la classe:

if __name__ == '__main__':

    c = Counter()

    print(c.toString())

    c.increment()

    print(c.toString())
```

Objet

- un objet a un type
- un objet a une adresse
- **un objet est une** instance **d'une classe**
- **un objet possède des** variables d'instance
- **un objet possède des méthodes**

Classe

- une classe permet de construire des objets
- une classe définit un nouveau type
- une classe définit la structure interne des objets
- une classe définit les méthodes des objets

Conception objet

- 1 concept = 1 classe
- Exemple : jeu de rôle avec des Monstres et des Compteurs
- 2 concepts : 2 classes
- la classe Monstre
- la classe Compteur

Notre super jeu

- un monstre peut compter
- un monstre peut donner son nom
- un monstre peut donner son age
- un monstre vieillit d'un an tous les 10 appuis sur le compteur

Quelles sont les méthodes offertes par un Monstre ?

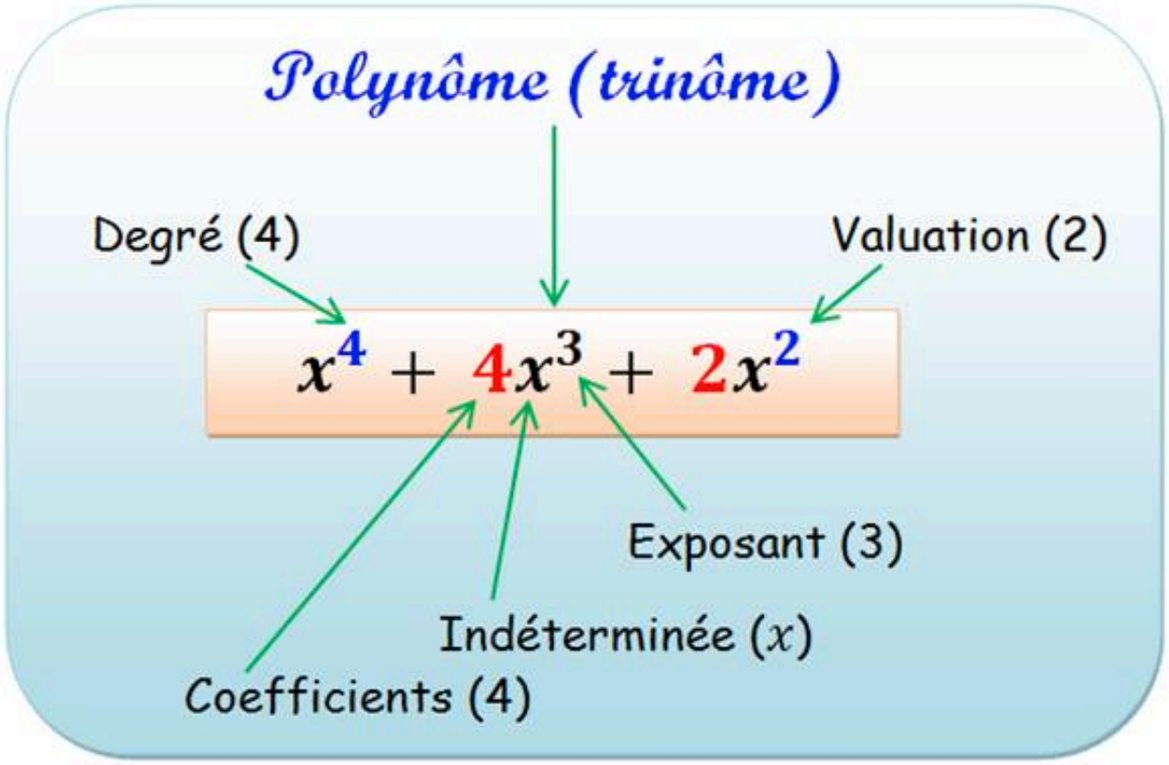
Quelles sont les attributs d'un Monstre ?

Let’s do it

Résumé

- `__init__(self)` pour construire l'objet
- `__str__(self)` pour représenter l'objet par une chaine
- `o.nom` pour accéder à un champ
- `o.nom()` pour appeler une méthode

En TD



Représenter des Polynômes
Calculer avec des Polynômes

Tests unitaires avec assert

```
class Counter:
   ...

if __name__ == '__main__':
   p = Counter()
   assert(p.value == 0)
```

Tests unitaires avec unittest

counter.py

```
class Counter:

  …
```

test_counter.py

```
import unittest
from counter import Counter
class TestCounter(unittest.TestCase):

  def test_create_counter(self):
    p = Counter()
    self.assertEqual(p.value, 0)

  def test_increment(self):
    p = Counter()
    p.increment()
    self.assertEqual(p.value,1)
    p.reinit()
    self.assertEqual(p.value,0)

if __name__ == '__main__':
  unittest.main()
```

Intérêts de `unittest`

- Tous les tests sont exécutés
- On voit rapidement quel test échoue
- Les messages d'erreur sont plus clairs (on peut voir le résultat attendu)