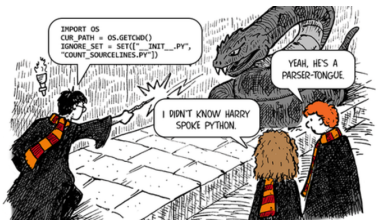
Devoir surveillé
2 h 00

Eric Berthomier
eric.berthomier@free.fr

17 octobre 2017



1 Notation

La correction de l'esemble des exercices prendra en compte l'ensemble de ces éléments.

Appréciation Générale	Orthographe, lisibilité, propreté
Duck Typing	Commentaires
Algorithmique	Programmation

2 Chiffrement de César (5 points)

Le chiffrement de césar consiste à décaler les lettres de l'alphabet d'un certain nombre de position pour un mot.

- **Faire saisir un texte, le mettre en majuscule et réaliser le chiffrement de César avec un décalage de 1.**
- Testez votre programme sur le message suivant : *AZ.* (BA.)
- Testez votre programme sur le message suivant : *Ceci est un message.* (DFDJ FTU VO NFTTBHF.)

2.1 Exemple

Avec un décalage de 1, E devient F, R devient S, I devient J, C devient D. Donc "ERIC" devient "FSJD"

2.2 Aide

- La mise en majuscule peut être effectué par la fonction `upper`.
- Les fonctions `ord` et `chr` permettent d'avoir le code ASCII d'un caractère et inversement.
- Les caractères majuscules [A-Z] ont un code ASCII entre 65 et 90
- Tout autre caractère que les caractères alphabétiques ne devra pas être pris en compte et rester tel que

2.3 Corrigé

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-

chaine = input ("Votre message : ")
chaine = chaine.upper ()

cesar = ""

for car in chaine:

   # Récupération du code ascii du caractère
   ascii = ord(car)

   if ((ascii >= 65 ) and ( ascii <= 90 )):
      # Ajout de 1 dans la code ascii
      if (ascii == 90):
         ascii = 65
      else:
         ascii +=1

   cesar+=chr(ascii)

print ("Chiffrement de César : %s" % (cesar))
```

3 Poulailler (15 points)

Un jeune couple de geeks souhaite se soulager des tracas quotidiens de leur petit poulailler. Aussi il décide de transformer leur poulailler en dompoul (poulailler domotique).

Le poulailler est équipé de
- un capteur de lumière
- un capteur permettant de connaître le nombre de poules dans le poulailler
- une porte automatique pilotable
- un capteur de poids permettant de connaître la quantité d'eau restant dans le réservoir d'eau
- un capteur de poids permettant de connaître la quantité de grain restant dans le réservoir de grains

L'ensemble des ordres domotiques devront être réalisé sous la forme d'affichage : `print ("Fermeture porte")` par exemple.

3.1 Exercice 1 - Nuit_Jour (2 points)

Écrire une fonction qui prend en entrée le nombre de lumens donné par le capteur et qui renvoie 0 pour Nuit et 1 pour Jour.
On considèrera qu'il fait jour si le nombre de lumens est supérieur à 50. Cette donnée devra être une variable globale.

3.2 Exercice 2 - Fermeture (3 points)

Écrire une fonction qui ferme la porte lorsque toutes les poules sont rentrées et qu'il fait nuit. Cette fonction prendra en paramètres le nombre de poules dans le poulailler, le nombre de poules total, et la luminosité.

3.3 Exercice 3 - Manger (3 points)

La portion de mélange grains / céréales à donner pour une poule est de 135 g.
Le réservoir possède une contenance de 5 kg de grains et le propriétaire rempli à sa guise.
Un capteur indique le poids en kg restant dans le réservoir.

Écrire une fonction qui permet de connaître le nombre de jour de rations disponibles dans le poulailler (nombre entier). Le nombre de poules pouvant varier (le renard est impitoyable mdr).

3.4 Exercice 4 - Modélisation (7 points)

Que ce soit un réservoir de grain ou d'eau la problématique reste la même et le fonctionnement identique.
Modéliser la classe Reservoir et implémenter 2 objets sur cette classe.
On prendra soin à identifier chaque réservoir. L'ensemble des unités utilisé est le gramme.
Cette classe devra prendre en compte :

1. reset : initialisation du réservoir à vide (0 kg) et de toutes les données qui lui sont associées
2. get_type : renvoie le type de réservoir
3. init_poids : mise à jour de l'information pour le poids contenu dans le réservoir
4. init_animaux : permet d'initialiser le nombre d'animaux
5. init_ration : détermine le grammage par jour par animal
6. get_nb_rations : retourne le nombre de rations disponible
7. alerte_rations : affichera : "Aucune Alerte pour <Identifiant du Réservoire>" ou "Alerte pour <Identifiant du Réservoire>" si le nombre de rations est à 1 ou à 0. Il renverra 0 si il n'y a pas d'alerte et 1 s'il y a une alerte.

Créer une instance de réservoir de type "Eau" et de type "Grains"

Réaliser la séquence suivante :

1. Je dispose de 2 réservoirs de 5kg un pour le grain et l'autre pour l'eau
2. Une poule boit environ 180 ml d'eau par jour et je dispose de 4 poules, afficher l'état du reservoir d'eau au fur et à mesure du temps (jour 1,2,3 ...). Pour chaque jour vous exécuterez le programme d'alerte.

-

3.5 Corrigé

```python
#!/usr/bin/python3
# -*- coding: UTF-8 -*-

class Reservoir ():

   # Le type du réservoir est une chaîne
   # "eau" pour un réservoir d'eau
   # "grain" pour un réservoir à grains
   def __init__(self,id,type,debug=0):
      self.type=type
      self.poids = 0
      self.animaux = 0
      self.ration = 0
      self.identifiant = id
      self.debug=debug
      return

   # Renvoie l'identifiant du réservoir
   def qui (self):
      return (self.identifiant)

   # Renvoie le type de réservoir
   def get_type (self):
      return (self.type)

   # Réinitialise le réservoir
   def reset (self):
      self.poids = 0
      self.animaux = 0
      self.ration = 0

   # Initialiser le poids du réservoir
   def init_poids (self, poids):
      self.poids = poids
```

```
   # Définit le nombre d'animaux
   def init_animaux (self, animaux):
      self.animaux = animaux

   # Définit la ration d'un animal
   def init_ration (self, ration):
      self.ration = ration

   # Renvoie le poids du réservoir
   def get_poids (self):
      return (self.poids)

   # Renvoie le nombre de rations disponibles
   def get_nb_rations (self):
      return (self.poids / self.animaux)

   # Envoie un alerte si le nombre de rations est trop petit
   def alerte_rations (self):
      reste = int (self.get_nb_rations ())
      if ( reste <= 1 ) :
         print ("Alerte sur %s - %s" % (self.qui (), self.get_type ()))
         return (1)
      else:
         print ("Aucune alerte")
         return (0)

   # Retourne l'état du réservoir
   def get_etat (self):
      return self.poids

   # Consomme 1 ration par jour par individu
   def consommer (self):
      self.poids = self.poids - (self.animaux * self.ration)
```

```
#!/usr/bin/python3
# -*- coding: UTF-8 -*-

import CReservoir

reservoireau = CReservoir.Reservoir ("R1","eau")
print ("Le type de réservoir est : " + reservoireau.get_type())

reservoirgrain = CReservoir.Reservoir ("R2","grain")
print ("Le type de réservoir est : " + reservoirgrain.get_type())

# Le réservoir contient 5kg d'eau
reservoireau.init_poids (5000)

#~ # Le réservoir a été prévu pour 4 poules
reservoireau.init_animaux (4)

# La consommation d'eau d'une poule est de 180 ml
reservoireau.init_ration (180)

for i in range (12):
   print ("Jour %d - %d g" % ((i+1) , reservoireau.get_poids()))
   reservoireau.consommer()
   reservoireau.alerte_rations()
```