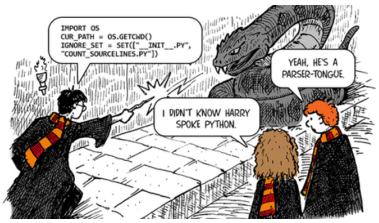
Devoir surveillé
2 h 30

Eric Berthomier
eric.berthomier@free.fr

13 mars 2016



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 Exercice 1 - Simulation d'un dé - 2 points

Simuler un jet de dé à 6 faces. Le joueur relance le dé, tant qu'il n'a pas appuyé (saisi - `input`) sur la touche F.

2.1 Corrigé

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

import random

fin=""

while ((fin != "F") and (fin != "f")) :
   de = random.randint(1,6)
   print ("Jet  : %d" % de)

   fin=input ("F pour finir\n")
```

3 Exercice 2 - Nombres premiers - 2 points

Indiquer si un nombre saisi est premier ou non. 0 et 1 ne sont pas des nombres premiers.

Rappel : un nombre est dit premier s'il n'est divisible que par lui-même et par 1.

3.1 Corrigé

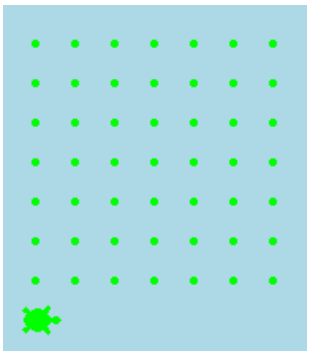
```
#!/usr/bin/python3
# -*- coding : UTF-8 -*-

# Saisie du nombre
nbre = int (input ("Votre nombre  : "))

if (nbre < 2) :
   print ("%d est n'est pas premier" % nbre)
else :
   valeur=2
   while ( ( nbre % valeur != 0 ) and ( valeur < nbre ))  :
      valeur += 1

   if ( valeur == nbre ) :
      print ("%d est premier" % nbre)
   else :
      print ("%d n'est pas premier" % nbre)
```

4 Exercice 3 - Utilisation de l'objet Turtle - 3 points

À l'aide du module Turtle et **en utilisant l'orienté objet de ce module**, écrire un programme qui dessine la figure géométrique ci-dessous et ceci en utilisant une double boucle.

4.1 Spécifications

- le carré fait 7 points sur 7 points
- l'espacement entre chaque point est de 25 pixels
- la fonction `dot()` permet de dessiner un point



4.2 Corrigé

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

import turtle

turtle.setup (800, 600)  # Taille du canevas

wn = turtle.Screen ()  # La fentre d'écran de Turtle
wn.bgcolor ("lightblue") # Couleur de fond de la fentre
wn.title ("Python Turtle")  # Définit un titre

angello= turtle.Turtle()  # Objet "Turtle"
angello.shape ("turtle")  # Une tortue au lieu d'un triangle
angello.color ("green")  # Tortue verte

dot_distance = 25
width = 7
height = 7

angello.penup()
```

```
for y in range(height) :
   for i in range(width) :
      angello.dot()
      angello.forward(dot_distance)
   angello.backward(dot_distance * width)
   angello.right(90)
   angello.forward(dot_distance)
   angello.left(90)

angello.penup()
wn.exitonclick()     # On attend un clic sur la croix
```

5 Exercice 4 - Horoscope - 3 points + 1 point bonus

Attention Cet exercice peut devenir chronophage.
À l'aide de la saisie de la date de naissance d'une personne, le programme affiche son signe du zodiaque. La date sera donnée sous la forme de 2 saisies : jours puis mois. On considèrera que l'utilisateur indique des dates réelles.

Il sera possible, **mais non obligatioire**, d'utiliser l'astuce suivante : `limite = mois * 100 + jour`.
Ainsi 13/08 se traduit par $0813 = 813$ qui est situé entre 722 et 822 donc Lion. [1]

Point bonus : utilisation de 2 listes et résolution par parcours d'une liste.

Bélier	21 mars au 19 avril	Taureau	20 avril au 21 mai
Gémeaux	21 mai au 21 juin	Cancer	21 juin au 22 juillet
Lion	22 juillet au 22 août	Vierge	23 août au 22 septembre
Balance	23 septembre au 22 octobre	Scorpion	23 octobre au 22 novembre
Sagittaire	23 novembre au 21 décembre	Capricorne	22 décembre au 19 janvier
Verseau	20 janvier au 19 février	Poissons	20 février au 20 mars

5.1 Corrigé

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

# Seules les dates de début de signe sont nécessaires, on ajoute une date fictive pour le Capricorne qui se situe à
      cheval sur l'année
tab_dates_zodiaque = [120, 220, 321, 420, 521, 622, 723, 822, 922, 1022, 1122, 1222, 1300]
tab_zodiaque = ["Capricorne","Verseau","Poissons","Bélier","Taureau","Gémeaux","Cancer","Lion","Vierge","Balance","
      Scorpion","Sagittaire","Capricorne"]

jj = input ("Jour de naissance  : ")
mm = input ("Mois de naissance  : ")

# Transformation en nombre
nbre_zod = int(mm) * 100 + int(jj)

# Affichage pour debugging
print ("Nombre  : ", nbre_zod)

# Parcours de la liste
i=0
while (tab_dates_zodiaque [i] <= nbre_zod) :
   i+=1

# Le signe du zodiaque est celui qui précède
print ("Votre signe zodiacal est %s " % tab_zodiaque[i])
```

1. Ne pas mettre 0xxx dans vos tableaux

6 Problème - La boulangerie - 10 points + 1 point bonus

6.1 Notation

Du fait même de la difficulté à comprendre la notion de la récursivité, les points associés aux exercices correspondants ne sont pas adaptés à la difficulté.

6.2 Calcul des proportions - 2 points

Pour faire un pâton de 360 g (nécéssaire à la fabrication d'une baguette traditionnelle), il est nécessaire de mélanger les ingrédients suivants :

- 50 % de farine
- 55 % **de la masse de farine** en eau
- 50 % **de la masse de farine** en fermento (eau / farine / levure)
- 3 % **de la masse de farine** en sel
- 4 ‰ **de la masse de farine** en levure

Votre programme demande au boulanger de saisir le nombre de baguettes traditionnelles qu'il désire préparer et lui retourne la masse de chaque ingrédient en kilogramme pour l'eau, la farine, le fermento et en grammes pour les autres éléments.
Vous utiliserez `%.2f` pour afficher 2 chiffres après la virgule.

6.3 Contrôle de saisie (valeurs numériques uniquement) - 1 point

Vérifier la saisie des informations données par le boulanger (uniquement des chiffres).

6.4 Corrigé

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

# Nombre de baguettes
try :
   nb_baguettes = int(input ("Nombre de baguettes  : "))
except ValueError  :
   print("Oops ! Ceci n'est pas un nombre. Essayez de nouveau...")
   exit (1)

# Poids des baguettes (en grammes)
pds_baguettes = nb_baguettes * 360

# Calcul
pds_farine = pds_baguettes / 2
pds_eau = pds_farine * 55 / 100
pds_fermento = pds_farine * 50 / 100
pds_sel = pds_farine * 3 / 100
pds_levure = pds_farine * 4 / 1000

# Poids de pâte
pds_pate = pds_farine + pds_eau + pds_fermento + pds_sel + pds_levure
print ("Poids de pâte  : %d g" % pds_pate)

# Affichage des informations
print ("Farine  : %.2f kg" % (pds_farine / 1000))
print ("Eau  : %.2f kg" % (pds_eau / 1000))
print ("Fermento  : %.2f kg" % (pds_fermento / 1000))
print ("Sel  : %.2f g" % pds_sel)
print ("Levure  : %.2f g" % pds_levure)
```

6.5 Gestion du silo à farine : écriture de fonctions - 5 points

On considère un fichier texte contenant le nombre de kilo de farine restant dans le silo (un entier). On considèrera sa valeur à 1 tonne soit 1000 kilos.

6.5.1 lire_etat_silo - fichier - 1,5 points

Écrire une fonction qui lit l'état du silo et le retourne.

6.5.2 ecrire_etat_silo - fichier - 1,5 points

Écrire une fonction qui écrit l'état du silo (l'état sera passé en paramètre (entier)).

6.5.3 enleve_farine - 1 point

Écrire une fonction qui enlève n kilos (paramètre) de farine du silo et met à jour son état.

6.5.4 ajoute_farine - 1 point

Écrire une fonction qui ajoute n kilos (paramètre) de farine du silo et met à jour son état.

6.6 Corrigé

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

def lit_etat() :

   # Ouverture du fichier en mode lecture
   f = open ("silo.txt", "r")

   valeur = f.readline()

   # Fermeture du fichier
   f.close()

   return(int(valeur))

def ecrit_etat(valeur) :

   # Ouverture du fichier en mode lecture
   f = open ("silo.txt", "w")

   f.write(str(valeur))

   # Fermeture du fichier
   f.close()

def enleve_farine(vEnleve) :

   etat = lit_etat()

   if (( etat - vEnleve ) < 0) :
      print ("Impossible de réaliser l'opération")
   else :
      etat -= vEnleve
      ecrit_etat(etat)

def ajoute_farine(vAjoute) :

   etat = lit_etat()

   if (( etat + vAjoute ) > 1000) :
      print ("Impossible de réaliser l'opération")
   else :
      etat += vAjoute
      ecrit_etat(etat)

ecrit_etat (1000)
enleve_farine (500)
print ("Etat du silo  : %d" % lit_etat())
ajoute_farine (500)
print ("Etat du silo  : %d" % lit_etat())
```

6.7 Modéliser l'objet Silo à Farine - 2 points + 1 point bonus

Créer la classe CSilo reprenant les 4 fonctions précédentes plus un constructeur (point bonus).

1. constructeur (intialisation à 1000 kg si le fichier n'existe pas)
2. lire_etat_silo

3. ecrire_etat_silo
4. enlever_farine
5. ajouter_farine

Exécuter le cheminement suivant :

1. Afficher l'état
2. Enlever 500 kg de farine
3. Afficher l'état
4. Ajouter 250 kg de farine
5. Affichier l'état

6.8 Corrigé

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

import os.path

class CSilo :

   # constructeur
   def __init__(self, nom_fichier) :

      # Definition du nom de fichier
      self.nom_fichier=nom_fichier

      #~ Si le fichier silo.txt n'existe pas, le créé avec la valeur 1000
      if not (os.path.isfile(self.nom_fichier)) :
         self.ecrit_etat(1000)

   def lit_etat(self) :

      # Ouverture du fichier en mode lecture
      f = open (self.nom_fichier, "r")

      valeur = f.readline()

      # Fermeture du fichier
      f.close()

      return(int(valeur))

   def ecrit_etat(self,valeur) :

      # Ouverture du fichier en mode lecture
      f = open (self.nom_fichier, "w")

      f.write(str(valeur))

      # Fermeture du fichier
      f.close()

   def enleve_farine(self,vEnleve) :

      etat = self.lit_etat()

      if (( etat - vEnleve ) < 0) :
         print ("Impossible de réaliser l'opération")
      else :
         etat -= vEnleve
         self.ecrit_etat(etat)

   def ajoute_farine(self,vAjoute) :

      etat = self.lit_etat()

      if (( etat + vAjoute ) > 1000) :
         print ("Impossible de réaliser l'opération")
```

```
      else :
         etat += vAjoute
         self.ecrit_etat(etat)
```

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

import silo

mon_silo=silo.CSilo("boulangerie.txt")

print ("Etat du silo  : %d" % mon_silo.lit_etat())

mon_silo.enleve_farine (500)
print ("Etat du silo  : %d" % mon_silo.lit_etat())

mon_silo.ajoute_farine (250)
print ("Etat du silo  : %d" % mon_silo.lit_etat())
```

7 Information

La note maximale est de $22/20$, toute note supérieure à 20 donne droit à un croissant et un café.