Ayudantía Programación

Ejercicios Certamen 3

Gonzalo Fernández

IWI-131 21/11/2017

Temas de la Ayudantía de Hoy

- Ejercicios.
- Ejercicios.
- Más ejercicios.

Pregunta de archivos

Considere el archivo `datos.txt`, indique en el cuadro de más abajo cómo queda el archivo `listado.txt` después de ejecutar el siguiente segmento de código:

```
def f1(a):
    c2 = a[1].split('-')[1]
    del a[1]
    return a, c2

archivo = open('datos.txt')
arch = open('listado.txt', 'w')
p = '{0}-{1}\n'
for linea in archivo:
    a,q = f1(linea.strip().split('/'))
    a1,a2 = map(int,a)
    s = p.format(str(a2-a1),q.upper())
    arch.write(s)
archivo.close()
arch.close()
```

datos.txt

```
2011/dic-usm-dic/2016
2009/ene-pucv-nov/2015
2009/ene-ubiobio-ago/2014
2009/ago-uai-dic/2015
2011/ago-uc-nov/2018
```

listado.txt

3. [40 %] La Universidad de Molterra tiene un avanzado sistema de control de sus estudiantes.

Para llevar control sobre las notas, cada alumno tiene un archivo cuyo nombre es su rol más la extensión `.txt`. La primera línea del archivo dice "Resumen academico", luego una línea en blanco y a continuación por cada nueva línea se lista cada uno de los cursos con nota. Los cursos deben estar ordenados según el semestre al que corresponden por malla y los cursos de un mismo semestre pueden estar en cualquier orden. Finalmente hay una línea que muestra el promedio de las notas en los cursos. A la derecha se muestra un **ejemplo** de archivo.
Para actualizar las notas un profesor indica el nombre de archivo donde puso las notas finales. Los 6 primeros caracteres del nombre del archivo indican la sigla del curso.

```
Resumen Academico

01 MAT021 56
01 IWG101 98
01 FIS100 64
01 QUI010 79
02 FIS110 60
02 IWI131 90
03 FIS120 40

PROMEDIO 69.57
```

Por ejemplo, para programación (código: IWI131) un nombre de archivo podría ser `IWI131201602.txt`. Cada línea del archivo contiene el rol de un alumno, el semestre al que corresponde el curso según su malla curricular y la nota que obtuvo. Los datos están separados por un espacio en blanco. Por ejemplo, una persona que está tomando el curso por malla en este semestre tendría una línea del tipo `201610001-k 02 90`, donde el `02` indica segundo semestre según su malla y el `90` es la nota que obtuvo.

Escriba un programa que pregunte al usuario (profesor) el nombre del archivo desde el cual leer notas, actualice el resumen académico de los respectivos alumnos en sus archivos y repita todo este proceso hasta que el nombre de archivo ingresado sea `0`. Notar que al actualizar un Resumen Académico el curso puede o no haber estado ingresado previamente, de estarlo, debe reemplazar la nota por la nueva. En cualquier caso, se debe actualizar el promedio general en la última línea del archivo.

Nota: puede utilizar la función `os.rename(nom_arch1, nom_arch2)`, la que renombra el archivo llamado `nom_arch1` con el nombre `nom_arch2`. Ambos parámetros son strings.

Restricción: Está prohibido el uso de cualquier función de ordenamiento usando alguna estructura de datos.

1. [20 %] Una consulta médica tiene un archivo `pacientes.txt` con los datos personales de sus pacientes. Cada línea del archivo tiene el rut, el nombre y la edad de un paciente, separados por un `':'`. Una línea del archivo se ve así: `12067539-7:Anastasia Lopez:32`.

Además, cada vez que alguien se atiende en la consulta, la visita es registrada en el archivo `atenciones.txt`, agregando una línea que tiene el rut del paciente, la fecha de la visita (en formato dia-mes-año) y el precio de la atención, también separados por `':'`. Una línea del archivo se ve así: `8015253-1:4-5-2010:69580`.

El médico de la consulta necesita saber los nombres de los pacientes que han sido atendidos en un día específico. A continuación se presentan las líneas de código que resuelven este problema, pero están desordenadas. Usted debe ordenarlas e indentarlas (dejar los espacios correspondientes de python) para que ambas funciones estén correctas.

La función `buscar_nombre(rut)` recibe un `rut` y entrega el nombre de la persona con ese rut (asuma que el rut buscado existe en el archivo correspondiente). La función `pacientes_dia(dia, mes, anio)` recibe la fecha y entrega una lista de los nombres de los pacientes que se atendieron ese día.

```
for line in arch:
arch.close()
def buscar_nombre(rut):
arch = open('pacientes.txt')
return n
if rut == r:
r, n, e = line.strip().split(':')
```

```
fecha = map(str,(dia, mes, anio))
arch = open('atenciones.txt')
if fecha == f:
def pacientes_dia(dia, mes, anio):
lista_n = list()
fecha = '-'.join(fecha)
arch.close()
return lista_n
for line in arch:
r, f, c = line.strip().split(':')
lista_n.append(nombre)
nombre  = buscar_nombre(r)
lista_n = list(set(lista_n))
```

1. [20 %] El banco Pythonbank maneja información de sus clientes en el archivo `clientes.txt`, con una estructura del tipo `rut#nombre#direccion`. Además cuenta con el archivo `movimientos.txt`, con una estructura del tipo `rut,producto,movimiento,aaaa-mm-dd`, donde se almacenan los cargos (movimientos negativos) y abonos (movimientos positivos) que se realizan en los productos de los clientes. A partir de estos archivos se desea generar un nuevo archivo `productos.txt`, con una estructura del tipo `rut/nombre/cantproductos/saldo`, donde se resumirá la información presente en los archivos antes mencionados.

A continuación se presentan las líneas de código que resuelven este problema, pero están desordenadas. Usted debe ordenarlas e indentarlas (dejar los espacios correspondientes de python) para que ambas funciones estén correctas.

La función `prod_cli`, a partir del archivo de movimientos, retorna una tupla con la cantidad de productos y el saldo final asociado a un rut dado. La función `archivo_prod` crea el archivo `productos.txt`, recibiendo como parámetro el nombre de los archivos antes descritos.

```
if p not in c:
mov_banco.close()
sumaprod += 1
saldo = saldo + int(s)
r,p,s,_ = li.strip().split(',')
c = set()
saldo = 0
def prod_cli(rut, movimiento):
c.add(p)
if r == rut:
return sumaprod, saldo
sumaprod = 0
for li in mov_banco:
mov_banco = open(movimiento)
```

```
cli_banco.close()
r, n, _ = li
cli_banco = open(cli)
prod.write(f.format(r, n, cant, s))
cant, s = prod_cli(r, movimiento)
for li in cli_banco:
li = li.strip().split('#')
prod.close()
prod = open('productos.txt', 'w')
f = '{0}/{1}/{2}/{3}\n'
def archivo_prod(cli, movimiento):
```

1. [25 %] La web Linkedpy analiza los procesos de postulación de recién titulados a empresas. Para ello tiene el archivo `titulados.txt`, donde cada línea tiene a los titulados en el formato `nombre;rut`, y el archivo `postulaciones.txt`, donde cada línea tiene un rut-titulado, el puesto y la empresa a la que cada titulado postula en el formato `rut#puesto#empresa`.

A partir de estos archivos se desea generar un archivo por empresa, los cuales deben tener los titulados que postularon a algún puesto en la empresa con el formato `rut;nombre;puesto`.

A continuación se presentan las líneas de código que resuelven este problema, pero que están desordenadas. Usted debe ordenarlas e indentarlas (dejar los espacios correspondientes de python) para que ambas funciones estén correctas.

La primera función retorna una lista con todas las empresas en el archivo con postulaciones (recibido como parámetro). Y la segunda función resuelve el problema antes descrito, recibiendo como parámetro el nombre del archivo con titulados y el nombre del archivo con postulaciones.

```
def empresas(post):
emp.append(e)
arch_P.close()
for li in arch_P:
r, p, e = li.strip().split('#')
if e not in emp:
arch_P = open(post)
emp = list()
return emp
```

```
arch_E = open(e + '.txt', 'w')
for pos in arch_P:
arch_T.close()
def registros(tit, post):
arch_E.write(li.format(r, n, p))
emp = empresas(post)
arch_T = open(tit)
n, r2 = titu.strip().split(';')
arch_P.close()
for e in emp:
arch_E.close()
if e2 == e:
r, p, e2 = pos.strip().split('#')
if r2 == r:
li = '{0};{1};{2}\n'
arch_P = open(post)
return None
for titu in arch_T:
```

1. [25 %] El Dpto de promoción de la USM necesita identificar los alumnos que participaron en los ensayos PSU del 2014 y que están matriculados este 2015, para lo cual posee dos archivos de texto, `matriculados.txt` y `ensayos.txt`, cada uno con la estructura descrita más abajo

`matriculados.txt`

```
Rut#Nombre#apellido#mail#carrera
```

`ensayos.txt`

```
Rut,nombre,ensayo,aaaa-mm-dd
```

A partir de estos archivos se desea generar uno nuevo, el cual debe tener los alumnos que participaron en algún ensayo PSU y que actualmente se encuentra matriculado en la USM. Los datos que le interesa tener son:

`Rut#nombre#apellido#mail#carrera#cantidad_ensayos`

A continuación se presentan las líneas del código que resuelven este problema, pero que están desordenadas. Usted debe ordenarlas e indentarlas (dejar los espacios correspondientes de python) para que ambas funciones estén correctas.

Tener en cuenta que la función *num_ensayos* debe retornar la cantidad de ensayos en la que participó la persona (rut) pasado como parámetro.

La función *crear_archivo* recibe 2 parámetros, el nombre del archivo con los alumnos matriculados y el nombre del archivo con los alumnos que participaron en algún ensayo. La función debe crear el archivo `Reporte.txt`, con los alumnos que están matriculados y que participaron en al menos 1 ensayo.

`num_ensayos`

```
arch.close()
def num_ensayos(rut, archivo):
if rut == l[0]:
arch = open(archivo)
l = linea.strip().split(",")
for linea in arch:
cont+=1
return cont
cont = 0
```

`crear_archivo`

```
new.close()
new = open("Reporte.txt","a")
def crear_archivo(matri, ensa):
x = num_ensayos(l[0],ensa)
for linea in arch:
arch.close()
if x>0:
s = linea.strip()
new.write(s+","+str(x)+"\n")
arch = open(matri)
l = linea.strip().split("#")
```

2. [40 %] La aplicación PYelper le permite conocer los mejores restaurantes en función de lo que otros usuarios han opinado. La información está almacenada en 2 archivos. En el archivo `estrellas.txt` se indica cuantas estrellas (de 0 a 5) le ha dado un usuario a cierto restaurante siguiendo la estructura: `usuario:restaurante:estrellas`. En el archivo `restaurantes.txt` están listados todos los restaurantes que han sido calificados y a qué comuna pertenencen en el siguiente formato: `restaurante:comuna`.

Los siguientes archivos son un **ejemplo** de lo anterior:

`estrellas.txt`

```
comensal2016:py_resto_bar:4
comensal2016:restoPy:3
gloton23:py_resto_bar:3
comensal2016:los_pysales:5
```

`restaurantes.txt`

```
py_resto_bar:comunaA
restoPy:comunaB
los_pysales:comunaA
```

Al respecto se le solicita:

a) Escribir la función `enComuna(archivo, comuna)` que retorna una lista con todos los restaurantes que están en la `comuna` descritos en el `archivo`.

```
>>> enComuna('restaurantes.txt', 'comunaA')
['py_resto_bar', 'los_pysales']
```

b) Escribir la función `promedioRestaurante(archivo, restaurante)` que retorna el promedio de estrellas que tiene el `restaurante` descrito en el `archivo`. Asuma que cada `restaurante` ha sido calificado al menos una vez.

```
>>> promedioRestaurante('estrellas.txt', 'py_resto_bar')
3.5
```

c) Escribir la función `elMejorDeLaComuna(archivoA, archivoB, comuna)` que retorna el nombre del restaurante con mejor promedio de la `comuna`, considerando que en el `archivoA` están las evaluaciones de los restaurantes y en el `archivoB` las comunas en las que están los restaurantes. Si hay más de un restaurante con la mejor evaluación retornar cualquiera, y si no hay restaurantes en la `comuna` retornar un string vacío (`''`).

```
>>> elMejorDeLaComuna('estrellas.txt', 'restaurantes.txt', 'comunaA')
'los_pysales'
```

3. [40 %] Twitter ha demostrado ser un buen termómetro del sentir ciudadano. En algunos casos, puede incluso llegar a predecir el resultado de una elección política. Considere que se tiene un archivo con la información de los candidatos a presidente de Pythonia en el formato `apellido;nombre;coalicion`. Una línea del archivo se ve así: `Netero;Isaac;Nueva asociacion`.

Además se tiene un archivo con todas las publicaciones (*tweets*) en Twitter en el formato `usuario;tweet;fecha`, donde `usuario` es quiene posteó el *tweet*, `tweet` el texto del *tweet* y `fecha` es cuando se publicó el *tweet* en formato `AAAA-MM-DD`. Asuma que no hay `';'` dentro del texto o en el nombre de usuario. Una línea del archivo se ve así: `@killua;que voto @presiNetero o freecss;2015-08-17`.

Escriba una función `separa_tweets(arch_tweets, arch_candi)` donde `arch_tweets` corresponde al archivo de tweets y `arch_candi` al archivo de candidatos. La función debe generar un archivo por cada candidado. El nombre del archivo debe ser `apellido.txt`, donde `apellido` es el apellido del candidato en minúscula. El archivo, para cada candidato, debe incluir aquellos *tweets* (con sus tres campos), en los que el texto del *tweet* **incluya** al apellido del candidato, **sin importar si está en mayúsculas o minúsculas**. Un *tweet* puede hacer referencia a más de un candidato, en cuyo caso dicho *tweet* debe ser escrito en cada uno de los archivos de los candidatos mencionados. El archivo debe tener los *tweets* ordenados por fecha de emisión, desde el más reciente al más antiguo en el formato `usuario;tweet;fecha`.