Taller de Programación

Examen

3 de Diciembre de 2019

Instrucciones:

- Lea atentamente el enunciado de cada uno de los problemas.
- Elija solo **TRES** de los CUATRO problemas del examen.
- Para cada problema cree un archivo.py distinto. El nombre del archivo debe ser el número del problema (uno.py, dos.py o tres.py)
- Comprima los problemas en un solo archivo **ZIP** y subalo a la sección Evaluación en http://canvas.udd.cl. Solo tiene una oportunidad para subir sus respuestas.
- Recuerde que usaremos un software de detección de plagio para detectar copia.

(2pts) Problema 1. Trabajando con Fechas

Su primera tarea en su nuevo trabajo como Data Scientist es crear un código que trabaje con las fechas de las próximas reuniones a realizarse en su empresa, creando funciones que cumplan los siguientes requisitos:

- Programe la funcion `fecha_tupla` que recibe una fecha en el formato `'dd-mm-aaaa'` y devuelve la fecha de tipo tupla `('dd','mm','aaaa')`.
- Programe la funcion `fecha_string` que recibe una fecha de tipo tupla `('dd','mm','aaaa')` y retorna la fecha en el formato `'dd-mm-aaaa'`.
- Programe la funcion `comparar_fechas(fecha1,fecha2)` que recibe dos fechas de tipo tupla `('dd','mm','aa')` y retorna `2` si `fecha1` > `fecha2`, `1` si `fecha1` < `fecha2` y `0` si `fecha1` = `fecha2`.

Hint: Una manera eficiente de comparar fechas es primero comparar los años, luego si estos son iguales entonces comparar los meses y si estos tambien son iguales entonces comparar los días.

Todo lo anterior debe estar plasmado en un programa que solicite al usuario la función que desea ejecutar, **por ejemplo**:

```
¿Qué desea hacer?
1. Transformar fecha a tupla
2. Comparar fechas
Seleccione una opción: _
```

Posible solución

```
def fecha_tupla(fecha):
    dd=fecha[0:2]
    mm=fecha[3:5]
    aa=fecha[6:10]
    return (dd,mm,aa)

def fecha_string(fecha):
    dd=fecha[0]
    mm=fecha[1]
    aa=fecha[2]
    return dd + '-' + mm + '-' + aa

 def comparar_fechas(fecha1,fecha2):
    ano1=fecha1[2]
    ano2=fecha2[2]

```

```
    mes1=fecha1[1]
    mes2=fecha2[1]

    dia1=fecha1[0]
    dia2=fecha2[0]

    if ano1>ano2:
        return 2
    elif ano1<ano2:
        return 1
    else:
        if mes1> mes2:
            return 2
        elif mes1< mes2:
            return 1
        elif mes1==mes2:
            if dia1>dia2:
                return 2
            elif dia1<dia2:
                return 1
            else:
                return 0

s = int(input('Que desea hacer?, seleccione 1 o 2'))
if s == 1:
    f = input('Ingrese fecha ')
    print(fecha_tupla(f))
if s == 2:
    f1=input('Ingrese fecha 1 ')
    f2=input('Ingrese fecha 2 ')
    f11=fecha_tupla(f1)
    f22=fecha_tupla(f2)
    print(comparar_fechas(f11,f22))
```

Puntaje:

- 0.5 pts. Programar la funcion fecha_tupla correctamente
- 0.5 pts. Programar la funcion fecha_string correctamente
- 0.7 pts. Programar la funcion comparar_fecha correctamente
- 0.3 pts. Incluir menu

(2pts) Problema 2. Diccionario de pobreza regional

Usando el siguiente diccionario relacionado a la pobreza regional de nuestro país:

```
pobreza_reg={'Tarapaca' : 7.1, 'Antofagasta' : 5.4, 'Atacama' : 6.9, 'Coquimbo' : 13.8, 'Valparaíso' : 12.0, 'Libertador Bernardo OHiggins' : 13.7, 'Maule' : 18.7, 'Biobío' : 17.6, 'La Araucanía' : 23.6, 'Los Lagos' : 16.1, 'Aysen' : 6.5, 'Magallanes y La Antártica Chilena' : 4.4, 'Región Metropolitana de Santiago' : 7.1, 'Los Ríos' : 16.8, 'Arica y Parinacota' : 9.7}
```

cuyas `llaves (keys)` corresponden a los nombres las regiones de Chile, y cuyos `valores (values)` corresponden a la información del porcentaje de pobreza de ingreso en la respectiva región *(Casen 2015)*, imprima el **nombre la región** con **mayor población en situación de pobreza** de ingresos y su respectivo porcentaje.

Posible solución

Nota: podía asumir que solo hay una región con máximo (dos o más regiones podían tener el mismo porcentaje).

```
max_pobreza = max(pobreza_reg.values())

for region,porcentaje in pobreza_reg.items():
    if porcentaje == max_pobreza:
        reg_max_pobreza = region

print('La región de Chile con mayor porcentaje de pobreza de ingresos es:', reg_max_pobreza, 'y su porcentaje es de:', max_pobreza, 'porciento')
```

Puntaje:

- 0.4 Encuentra el valor máximo de los porcentajes de pobreza (usando la función max() o un ciclo que recorra los valores (values) del diccionario).
- 0.8 Recorre los items del diccionario (keys y values) para buscar el nombre de la región con mayor porcentaje de pobreza.
- 0.5 Aplica correctamente la condición para encontrar el nombre de la región.
- 0.3 Imprime correctamente el nombre de la región con mayor porcentaje de pobreza, y el respectivo porcentaje.

(2pts) Problema 3. Progresión de Diego

Los números de la progresión de Diego son los favoritos de los amantes de la simetría, pues son aquellos que se leen de la misma manera tanto de izquierda a derecha, como de derecha a izquierda. **Por ejemplo:** 22, 33, 2882, 5005, 292, 2882, etc.

Se le solicita crear un programa que imprima todos los números pertenecientes a la progresión de Diego contenidos en el intervalo cerrado de `[1, 3000]`.

```
for i in range(1, 3001):
    j = str(i)
    jr = j[::-1]
    if j == jr:
        print(i)
```

Puntaje:

- 0.3 pts. hacer range(0, 3001)
- 0.5 pts. convertir numero a str o list
- 0.5 pts. calcular reverso de str o list
- 0.2 pts. chequear que es palíndromo
- 0.5 pts. print de número palíndromo

(2pts) Problema 4. Notas de presentación a Examen

Usted es la nueva profesora de Programación en la Universidad Libre de Las Condes, y desea crear una función que permita agregar la información de cada alumno de forma **ordenada** según su **nota de presentación** a examen. Su función debe evaluar y ordenar los registros en un archivo, cada vez que se agrega un nuevo contenido. Como parámetro de entrada, la función recibe una lista con la siguiente información: `nombre`, `apellido`, `rut`, y `nota_presentacion`.

```
def actualizar_archivo(lista):
    #escriba aquí su código
    return "Registro Actualizado!"
lista = ['Moisés', 'Bravo', '11111111-8', 5.2]
print(actualizar_archivo(lista))
```

Nota: El archivo debe guardarse separado por `;`. Recuerde que para escribir datos en un archivo debe abrirlo en modo escritura, es decir, con modo `'w'`. Para escribir en el archivo debe usar la función `write(texto)`, donde texto es una variable de tipo `str`. Por ejemplo, el siguiente código crea el archivo `notas.txt` e inserta tres líneas (notar que cada string contiene el caractér de salto de línea `\n`:

```
f = open('notas.txt','w')
f.write('primera linea\n')
f.write('segunda linea\n')
f.write('tercera linea\n')
f.close()
```

Posible solución a) Idea: matener una lista en el scope global, cada vez que se ejecuta la función agregar el estudiante a la lista y ordenarla, y finalmente guardar el archivo.

```
from operator import itemgetter

# aqui guardamos a los estudiantes
estudiantes = []

def actualizar_archivo(lista):
    # agregamos a la lista y ordenamos
    estudiantes.append(lista)
    estudiantes.sort(key=itemgetter(3))

    # aqui guardamos
    f = open('notas.txt','w')
    for al in estudiantes:
        #crear string manualmente
```

```
        linea = '{};{};{};{}\n'.format(al[0],al[1],al[2],al[3])
        f.write(linea)
    f.close()
    return "Registro Actualizado!"

lista = ['Moisés', 'Bravo', '11111111-8', 5.2]
print(actualizar_archivo(lista))
lista = ['Carolina', 'Garcia', '122222-8', 6.1]
print(actualizar_archivo(lista))
lista = ['Diego', 'Caro', '122211-8', 7]
print(actualizar_archivo(lista))
```

Posible solución b) Asumiendo que el archivo `notas.txt` existe, podemos leer la lista de alumnos desde el archivo y ponerlos en una lista. Ordenamos la lista, y luego re-escribimos la lista ordenanda el en archivo.

```
from operator import itemgetter

def actualizar_archivo(lista):
    # estudiantes es una lista con el estudiante
    estudiantes = [lista]

    # leemos los estudiantes (asumimos que el archivo notas.txt existe)
    f = open('notas.txt','r')
    for linea in f:
        est = linea.strip().split(';')
        est[3] = float(est[3])
        estudiantes.append(est)
    f.close()

    # ordenamos por nota
    estudiantes.sort(key=itemgetter(3))

    # guardamos las notas ordenadas
    f = open('notas.txt','w')
    for al in estudiantes:
        #crear string manualmente
        linea = '{};{};{};{}\n'.format(al[0],al[1],al[2],al[3])
        f.write(linea)
    f.close()
    return "Registro Actualizado!"

lista = ['Moisés', 'Bravo', '11111111-8', 5.2]
print(actualizar_archivo(lista))
lista = ['Carolina', 'Garcia', '122222-8', 6.1]
```

```
print(actualizar_archivo(lista))
lista = ['Diego', 'Caro', '122211-8', 7]
print(actualizar_archivo(lista))
```

Puntaje:

- 0.5 pts: mantener la lista de estudiantes global, o leerla desde el archivo
- 0.5 pts: ordenar la lista usando la nota
- 0.5 pts: abrir el archivo en modo escritura
- 0.5 pts: guardar linea siguiendo el formato separado por `;` y agregando un salto de línea `\n` por cada estudiante.