Auxiliar10_pauta

December 3, 2020

1 Auxiliar 10 - Archivos y Diccionarios

1.0.1 P1 Lectura/Escritura de Archivos

El archivo notas.txt contiene una lista con los nombres de los estudiantes y sus respectivas notas. Cada linea del archivo corresponde a un alumno, donde primero se tiene el nombre, seguido por sus notas, separando cada valor con una coma.

Programe la función promedio, la cual a partir del archivo notas.txt genere un nuevo archivo llamado promedio_notas.txt el cual contenga los nombres y el promedio de notas de cada alumno.

Puede asumir que los datos serán entregados en un formato correcto.

[16]:
```
# promedio: None -> None
# Esta función lee el archivo notas.txt, el cual contiene una lista de␣
 ↪estudiantes con sus respesctivas notas,
# y escribe un archivo que contiene a los mismos alumnos con el promedio de sus␣
 ↪notas.
# Ej: La linea que tiene escrito Sherry,6.5,6.0,7.0 deberá escribir en el otro␣
 ↪archivo Sherry,6.5.
def promedio():
    archivo= open("notas.txt", "r")
    nuevas_lineas=[]
    for linea in archivo:
        linea=linea.strip()
        valores=linea.split(",")
        nombre=valores[0]
        notas=valores[1:]
        sum_notas=0
        for nota in notas:
          sum_notas=float(nota)+sum_notas
        promedio=round(sum_notas/len(notas),1)
        nueva_linea=nombre+" "+str(promedio)+"\n"
        nuevas_lineas.append(nueva_linea)
    archivo.close()
    nuevo_archivo=open("promedio_notas.txt", "w")
    for linea in nuevas_lineas:
      nuevo_archivo.write(linea)
    nuevo_archivo.close()
```

```
promedio()
```

1.0.2 P2. Diccionarios

Para comunicarnos secretamente con nuestros amigos, vamos a crear un programa que reemplace las letras del alfabeto por otra letra del mismo. Para ello debemos definir un alfabeto y su sustitución.

Por ejemplo:
alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
substitución = "QWERTYUIOPASDFGHJKLZXCVBNM"

Para los valores anteriores, tendremos que la letra A es reemplazada por la letra Q, la B por W, la C por la E y así sucesivamente.

Programaremos las siguientes funciones que nos permitirán cifrar y decifrar nuestros mensajes.

1.0.3 A)

Defina la función crearDiccionario, esta debe recibir dos strings, uno correspondiente al alfabeto y el otro a la substitución que utilizaremos.

A partir de estos valores debe devolver un diccionario que contenga todas las letras del alfabeto con sus respectivas subtituciones.

Por ejemplo:
alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
substitución = "QWERTYUIOPASDFGHJKLZXCVBNM"
crearDiccionario(alfabeto, substitucion) devolverá:
{'A': 'Q', 'B': 'W', 'C': 'E', 'D': 'R', 'E': 'T', 'F': 'Y', 'G': 'U', 'H': 'I', 'I': 'O', 'J': 'P', 'K': 'A', 'L': 'S', 'M': 'D', 'N': 'F', 'O': 'G', 'P': 'H', 'Q': 'J', 'R': 'K', 'S': 'L', 'T': 'Z', 'U': 'X', 'V': 'C', 'W': 'V', 'X': 'B', 'Y': 'N', 'Z': 'M'}

[17]:
```
# crearDiccionario: str str -> dict
# Crea un diccionario a partir del alfabeto y las sustitución que se le pasa␣
 ↪como parámetro,
# donde cada letra será una llave y su valor será su respectiva substitución.
# Ejemplo:
# alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# substitución = "QWERTYUIOPASDFGHJKLZXCVBNM"
# crearDiccionario(alfabeto, substitucion) devolverá:
#{'A': 'Q', 'B': 'W', 'C': 'E', 'D': 'R', 'E': 'T',
# 'F': 'Y', 'G': 'U', 'H': 'I', 'I': 'O', 'J': 'P',
# 'K': 'A', 'L': 'S', 'M': 'D', 'N': 'F', 'O': 'G',
# 'P': 'H', 'Q': 'J', 'R': 'K', 'S': 'L', 'T': 'Z',
# 'U': 'X', 'V': 'C', 'W': 'V', 'X': 'B', 'Y': 'N', 'Z': 'M'}

def crearDiccionario(alfabeto, substitucion):
  diccionario={}
  largo=len(alfabeto)
  for indice in range(largo):
    letra=alfabeto[indice]
```

```
    reemplazo=substitucion[indice]
    diccionario[letra]=reemplazo
  return diccionario

original = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
substitucion = "QWERTYUIOPASDFGHJKLZXCVBNM"
diccionario = {'A': 'Q', 'B': 'W', 'C': 'E', 'D': 'R', 'E': 'T',
 'F': 'Y', 'G': 'U', 'H': 'I', 'I': 'O', 'J': 'P',
 'K': 'A', 'L': 'S', 'M': 'D', 'N': 'F', 'O': 'G',
 'P': 'H', 'Q': 'J', 'R': 'K', 'S': 'L', 'T': 'Z',
 'U': 'X', 'V': 'C', 'W': 'V', 'X': 'B', 'Y': 'N', 'Z': 'M'}

assert crearDiccionario(original, substitucion)==diccionario
```

1.0.4 B)

Programe la función cifrarTexto, esta recibirá un diccionario y un string correspondiente al mensaje que queremos cifrar.La función deberá devolver el mensaje cifrado a partir del diccionario que recibió. Además, si alguna de las letras del mensaje no se encuentra en el diccionario, deberá agregarlas sin substituir su valor.

Por ejemplo, si utilizamos el diccionario del ejemplo anterior y el texto "HOLA UWU" debemos recibir como respuesta "IGSQ XVX", además, en este caso deberíamos agregar " " al diccionario.

[18]:
```
# cifrarTexto: dict str -> str
# Devuelve el texto cifrado a partir del diccionario entregado como parámetro.
# Si algún valor no se encuentra en el diccionario, se agrega sin substituir su␣
 ↪valor.
# Por ejemplo, para el diccionario del ejercicio anterior y el texto "HOLA UWU".
# se debe recibir como respuesta IGSQ XVX.
def cifrarTexto(diccionario, texto):
  solucion=""
  for letra in texto:
    if not letra in diccionario:
      diccionario[letra]=letra
    solucion=solucion+diccionario[letra]
  return solucion

diccionario1=crearDiccionario(original, substitucion)
assert cifrarTexto(diccionario1, "HOLA UWU") == "IGSQ XVX"
diccionario2=crearDiccionario(substitucion, original)
assert cifrarTexto(diccionario2, "IGSQ XVX") == "HOLA UWU"
```

1.0.5 P3. Iterar en un diccionario

Programe una función llamada maxReunión, esta debe busca el horario en el que más personas pueden participar de una reunión (asuma que sólo habrá un máximo). Esta función no recibe parámetros y debe retornar el valor del la maáxima cantidad de participantes. Además, debe mostrar en la pantalla el día, el horario y los participantes de la reunión con mayor asistencia.

Para lograr lo descrito, se tendrá la variable global horarios, la cual está definida a continuación. Esta contiene un diccionario con los horarios en que los participantes de la reunión confirmaron su disponibilidad.

[19]:
```
horarios={
  "lunes": {
    "mañana": ["Juan","Andrés"],
    "tarde": ["Pepito","Juan"],
    "noche":["Anita", "Pedrito"]
  },
  "martes": {
    "mañana": ["Andrea","María"],
    "tarde": [],
    "noche":["Andrea", "Pedrito", "María"]
  },
  "miércoles":{
    "mañana": ["Pepito","Juan","Andrea","María"],
    "tarde": [],
    "noche":[]
  },
  "jueves": {
    "mañana": ["Juan","Andrés"],
    "tarde": ["Pepito","Juan","María"],
    "noche":["Pepito", "María"]
}
}
```

[20]:
```
# maxReunion: None -> int
# Busca el horario en que más personas confirmaron su disponibilidad para␣
 ↪participar en un reunión.
# Ej:
# maxReunion() debe retornar 4, e imprimir en pantalla que el día miércoles en␣
 ↪la mañana pueden asistir
# Pepito, Juan, Andrea y María.

def maxReunión():
  global horarios
  max=0
  dia_max=""
  hora_max=""
  for dia in horarios:
    for hora in horarios[dia]:
      cantidad_asistentes=len(horarios[dia][hora])
      if cantidad_asistentes>max:
        max=cantidad_asistentes
        dia_max=dia
        hora_max=hora
  print("El día "+ dia_max+" en la "+ hora_max+" pueden asistir:")
```

```
  for invitado in horarios[dia_max][hora_max]:
    print(invitado)
  return max

assert maxReunión() == 4
```

```
El día miércoles en la mañana pueden asistir:
Pepito
Juan
Andrea
María
```