Clase 08 - Usando cadenas

May 25, 2025

1 Clase 08

1.1 Programación

1.1.1 Problema 1

Debido a intromisiones que han ocurrido en Canvas, se le ha pedido que implemente en Python un sistema para ingresar una clave para proteger su acceso. Esta debe tener entre 6 y 10 símbolos, debe contener al menos 5 letras, entre las cuales 3 deben ser 'A', 'B' o 'C', además, debe tener al menos un dígito. La clave no puede tener espacios. Si la clave está mal ingresada se debe repetir el ingreso.

Entrada:

La unica entrada es una cadena de símbolos.

Salida:

La unica salida del programa es uno de los siguientes mensajes clave ingresada correctamente clave no cumple los requisitos, ingrese nuevamente.

Ejemplo de entrada:

0_ACABAR87

Ejemplo de salida:

clave ingresada correctamente

[ ]:
```
"""
Valida si la clave cumple con los requisitos:
    - Longitud entre 6 y 10 caracteres
    - No contiene espacios
    - Al menos 5 letras
    - Al menos 3 letras de 'ABC'
    - Al menos 1 dígito
"""
def validar_clave(clave):
    if len(clave) < 6 or len(clave) > 10:
        return False
    if ' ' in clave:
        return False
    letras = sum(1 for c in clave if c.isalpha())
```

```
    digitos = sum(1 for c in clave if c.isdigit())
    letras_ABC = sum(1 for c in clave if c in 'ABC')

    if letras < 5 or letras_ABC < 3 or digitos < 1:
        return False
    return True
"""
Solicita la clave al usuario y usa la dunción vilidar_clave para verificarla
Si la clave no cumple con los requisitos, solicita nuevamente, hasta que se␣
 ↪ingrese una clave válida.
"""
def solicitar_clave():
    while True:
        clave = input("Ingrese la clave: ")
        if validar_clave(clave):
            print("clave ingresada correctamente")
            break
        else:
            print("clave no cumple los requisitos, ingrese nuevamente.")

solicitar_clave()
```

Explicación del código El código define una función llamada `validar_clave` que verifica si una clave (contraseña) cumple con ciertos requisitos de seguridad. Primero, la función comprueba que la longitud de la clave esté entre 6 y 10 caracteres; si no es así, retorna **False**. También verifica que la clave no contenga espacios, retornando **False** si encuentra alguno.

Luego, la función cuenta cuántas letras (`letras`), cuántos dígitos (`digitos`) y cuántas letras 'A', 'B' o 'C' (`letras_ABC`) contiene la clave. Para que la clave sea válida, debe tener al menos 5 letras, al menos 3 letras entre 'A', 'B' o 'C', y al menos 1 dígito. Si no cumple alguna de estas condiciones, retorna **False**. Si pasa todas las verificaciones, retorna **True**.

La función `solicitar_clave` solicita al usuario que ingrese una clave y utiliza `validar_clave` para comprobar si es válida. Si la clave cumple los requisitos, muestra un mensaje de éxito y termina el ciclo; si no, pide al usuario que ingrese una nueva clave hasta que sea válida. Esta estructura garantiza que solo se acepten claves que cumplan con todas las condiciones de seguridad especificadas.

Explicación de `letras`, `digitos` y `letras_ABC` Estas tres líneas de código cuentan diferentes tipos de caracteres dentro de la variable clave, que representa una cadena de texto.

La primera línea, `letras = sum(1 for c in clave if c.isalpha())`, recorre cada carácter c en la clave y suma 1 por cada carácter que sea una letra (mayúscula o minúscula), utilizando el método `.isalpha()`. Así, `letras` almacena el total de letras presentes en la clave.

La segunda línea, `digitos = sum(1 for c in clave if c.isdigit())`, hace algo similar pero suma 1 solo por cada carácter que sea un dígito numérico, usando el método `.isdigit()`. De esta forma, `digitos` contiene la cantidad total de números en la clave.

La tercera línea, `letras_ABC = sum(1 for c in clave if c in 'ABC')`, suma 1 por cada carácter que sea exactamente 'A', 'B' o 'C'. Así, `letras_ABC` almacena cuántas veces aparecen estas letras específicas en la clave.

Estas variables se usan después para validar si la clave cumple con los requisitos mínimos de letras, dígitos y letras específicas.

Explicación de `.isalpha` El método `.isalpha` es una función incorporada en Python que se utiliza para verificar si todos los caracteres de una cadena son letras del alfabeto (ya sean mayúsculas o minúsculas). Cuando se llama como `cadena.isalpha()`, retorna **True** si la cadena contiene únicamente letras y no está vacía; de lo contrario, retorna **False**. Es útil para validar entradas donde solo se permiten letras, excluyendo números, espacios y símbolos especiales. Por ejemplo, *`"Hola".isalpha()`* devuelve **True**, mientras que *`"Hola123".isalpha()`* devuelve **False**.

Explicación de `.isdigi`

El método `.isdigit` es una función incorporada en Python que se utiliza para verificar si todos los

Explicación by Copilot

1.1.2 Problema 2

Se dice que quiénes pueden leer el siguiente texto:

> *C12R70 D14 D3 V3R4N0 357484 3N L4 PL4Y4 O853RV4ND0 D05 CH1C45 8R1NC4ND0 3N L4 4R3N4, 357484N 7R484J4ND0 MUCH0 C0N57RUY3ND0 UN C4571LL0 D3 4R3N4 C0N 70RR35, P454D1Z05 0CUL705 Y PU3N735*

tienen capacidades especiales en su cerebro. Pero Ud. no cuenta con dichas capacidades especiales pero sabe programar en Python, por lo que escribirá código que le permita traducir este tipo de textos.

Entrada:

La´unica entrada es una cadena de símbolos de longitud m´axima 1000 símbolos compuesta de letras, dígitos, espacios, comas y/o puntos.

Salida: La´unica salida del programa es la cadena ingresada en la que los dígitos han sido reemplazados por letras segúnn la siguiente tabla:

Dígito	Letra
1	*I*
3	*E*
4	*A*
5	*S*
7	*T*
0	*O*

Ejemplo de entrada

: C12R70 D14 D3 V3R4N0 357484 3N L4 PL4Y4

Ejemplo de salida

: CIERTO DIA DE VERANO EN LA PLAYA

[ ]:
```
"""""
La función traduccion toma un texto y reemplaza ciertos caracteres por letras␣
 ↪específicas, según la tabla.
"""
def traduccion(texto):
    traduccion = {
        '1': 'I',
        '3': 'E',
        '4': 'A',
        '5': 'S',
        '7': 'T',
        '0': 'O'
    }
    resultado = []
    for caracter in texto:
        if caracter in traduccion:
            resultado.append(traduccion[caracter])
        else:
            resultado.append(caracter)
    return ''.join(resultado)

# Se pide al usuario un texto para traducir y se imprime el resultado con la␣
 ↪función traduccion.
entrada = input("Ingrese el texto a traducir: ")
salida = traduccion(entrada)
print(salida)

"""
Observar que la función traduccion reemplaza los caracteres según la tabla␣
 ↪dada, sin embargo, no se especifican ciertas letras.
Por ello, no se logra la traducción completa de la frase.
Puede ser que la tabla está incompleta o el ejercicio está mal estructurado.
"""
```

Explicación del código La función `traduccion` permite transformar un texto que contiene ciertos dígitos en una versión donde esos dígitos son reemplazados por letras específicas, según una tabla de conversión. El diccionario `traduccion` define las correspondencias: por ejemplo, el dígito *'1'* se reemplaza por *'I'*, el *'3'* por *'E'*, el *'4'* por *'A'*, y así sucesivamente para los dígitos *'5'*, *'7'* y *'0'*.

El proceso consiste en recorrer cada carácter del texto de entrada. Si el carácter es uno de los dígitos definidos en el diccionario, se agrega la letra correspondiente a la lista `resultado`. Si no, se agrega el carácter original sin cambios. Al finalizar el recorrido, la función une todos los elementos de la lista en una sola cadena usando `''.join(resultado)` y la retorna.

En la parte principal del código, se solicita al usuario que ingrese un texto, se traduce usando la función y se imprime el resultado. Así, el programa facilita la lectura de textos escritos con números en lugar de letras, reemplazando automáticamente los dígitos por las letras indicadas.

Explicación de `.join` El método `.join` es una función incorporada en Python que se utiliza para unir (concatenar) los elementos de un iterable (como una lista o una tupla) en una sola cadena de texto. Se llama sobre una cadena que actúa como separador y recibe como argumento el iterable cuyos elementos se desean unir. Por ejemplo, `'-'.join(['a', 'b', 'c'])` produce la cadena `'a-b-c'`. Si se usa una cadena vacía como separador, como en `''.join(lista)`, todos los elementos se concatenan sin ningún carácter entre ellos. Este método es muy útil para convertir listas de caracteres o palabras en una sola cadena.

Explicación de `.append`

El método `.append` es una función incorporada en Python que se utiliza para agregar un elemento

Explicación by Copilot

1.1.3 Problema 3

Construya un programa en Python que lea un mensaje compuesto por letras (mayúsculas y minúsculas), signos de puntuación (“,”,“;” y “.”) y espacios (“ ”). El tamaño máximo del mensaje es 1000 símbolos. Luego, implemente las siguientes operaciones:

- Que cuente las palabras. Las palabras son conjuntos de una o más letras (minúsculas y/o mayúsculas) que están separadas por uno o más espacios o signos de puntuación.
- Que cuente la cantidad de vocales (mayúsculas y minúsculas).

Entrada:

La´unica entrada es una cadena de símbolos de longitud máxima 1000 símbolos compuesta de letras (mayúsculas y/o minúsculas), espacios, símbolos “,”, “.” y/o “;”.

Salida

: La´unica salida del programa está compuesta por 6 números, la cantidad de palabras y la cantidad de cada una de las vocales.

Ejemplo de entrada:

La bella y graciosa moza, marchose a lavar la ropa. La mojó en un arroyuelo.

Ejemplo de salida:

15 palabras, 13 vocales ’a’, 3 vocales ’e’, 1 vocal ’i’, 7 vocales ’o’ y 2 vocales ’u’.

[ ]:
```python
# Función que cuenta la cantidad de vocales en un mensaje.

def contar_vocales(mensaje):
    # Inicializamos el contador de vocales
    contador_vocales = {
        'a': 0,
        'e': 0,
```

```
        'i': 0,
        'o': 0,
        'u': 0
    }
    # Convertimos el mensaje a minúsculas para contar las vocales sin importar␣
 ↪su caso
    mensaje = mensaje.lower()
    # Contamos las vocales
    for letra in mensaje:
        if letra in contador_vocales:
            contador_vocales[letra] += 1
    return contador_vocales

# Función que cuenta la cantidad de palabras en un mensaje.

def contar_palabras(mensaje):
    # Reemplazamos los signos de puntuación por espacios
    for signo in [',', ';', '.']:
        mensaje = mensaje.replace(signo, ' ')
    # Contamos las palabras separadas por espacios
    palabras = mensaje.split()
    return len(palabras)

# Función que solicita un mensaje al usuario y cuenta las palabras y vocales.

def solicitar_mensaje():
    # Leemos el mensaje
    mensaje = input("Ingrese el mensaje: ")
    # Contamos las palabras
    cantidad_palabras = contar_palabras(mensaje)
    # Contamos las vocales
    contador_vocales = contar_vocales(mensaje)
    # Imprimimos los resultados
    print(f"{cantidad_palabras} palabras, {contador_vocales['a']} vocales 'a',␣
 ↪{contador_vocales['e']} vocales 'e', {contador_vocales['i']} vocal 'i',␣
 ↪{contador_vocales['o']} vocales 'o' y {contador_vocales['u']} vocales 'u'.")

solicitar_mensaje()
```

Explicación del código Este fragmento de código contiene tres funciones que trabajan juntas para analizar un mensaje de texto ingresado por el usuario. La función `contar_vocales` recibe una cadena y cuenta cuántas veces aparece cada vocal (‘a’, ‘e’, ‘i’, ‘o’, ‘u’), sin importar si están en mayúsculas o minúsculas. Para lograrlo, primero convierte todo el mensaje a minúsculas y luego recorre cada carácter, incrementando el contador correspondiente si encuentra una vocal.

La función `contar_palabras` se encarga de contar la cantidad de palabras en el mensaje. Para evitar que los signos de puntuación interfieran en el conteo, reemplaza las comas, puntos y puntos

y coma por espacios. Después, utiliza el método `split()` para separar el mensaje en palabras y retorna la cantidad total.

Por último, la función `solicitar_mensaje` pide al usuario que ingrese un mensaje, utiliza las dos funciones anteriores para obtener la cantidad de palabras y el conteo de cada vocal, y finalmente imprime los resultados en un formato claro y detallado. Así, el usuario puede ver fácilmente cuántas palabras y cuántas veces aparece cada vocal en el texto ingresado.

Explicación de `.split` El método `.split` en Python se utiliza para dividir una cadena de texto en una lista de subcadenas, usando un separador específico. Por defecto, si no se indica ningún argumento, separa la cadena en cada espacio en blanco, eliminando los espacios adicionales. Por ejemplo,`"uno dos tres".split()` devuelve `['uno', 'dos', 'tres']`. También se puede especificar otro carácter como separador, por ejemplo, `"a,b,c".split(',')` produce `['a', 'b', 'c']`. Este método es muy útil para extraer palabras o elementos individuales de una cadena de texto.

Explicación de `.lower`

El método `.lower`es una función incorporada en Python que se utiliza para convertir todos los cara

Explicación by Copilot

1.1.4 Problema 4

Para los Aliados, enviar mensajes codificados fue muy importante durante la Segunda Guerra Mundial. Unos matemáticos que trabajaban en el equipo criptográfico de los aliados tenía una ingeniosa idea, enviar una contraseña escondida dentro del propio mensaje codificado. El punto interesante era que el receptor del mensaje solo tenía que saber el tamaño de la contraseña y luego buscar la contraseña dentro del texto recibido.

Se puede encontrar una contraseña de tamaño N buscando en el texto la subcadena de tamaño N que aparece con más frecuencia. Después de encontrar la contraseña, se eliminan del texto todas las subcadenas que coinciden con la contraseña. Se le pide que escriba un programa Python que, dado el tamaño de la contraseña y el mensaje codificado, determina la contraseña siguiendo la estrategia dada anteriormente.

Para ilustrar su tarea, considere el siguiente ejemplo en el que el tamaño de la contraseña es tres ($N = 3$) y el mensaje de texto es *baababacb*. La contraseña sería, entonces; *aba* porque esta es la subcadena con el tamaño 3 que aparece más a menudo en todo el texto (aparece 2 veces) mientras que las otras 6 diferentes subcadenas aparecen solo una vez (*baa*, *aab*, *bab*, *bac*, *acb*).

Entradas:

La entrada está compuesta por un número entero $N(0 < N \leq 10)$, correspondiente al tamaño de la clave, seguido del texto codificado. Para simplificar, suponga que el texto incluye sólo letras minúsculas.

Salidas:

El programa debe desplegar la clave.

Ejemplo de Entrada:

baababacd

Ejemplo de Salida:

aba

[ ]:
```python
def encontrar_clave(n, texto):
    frecuencias = {}

    # Contar las subcadenas de longitud n
    for i in range(len(texto) - n + 1):
        subcadena = texto[i:i+n]
        if subcadena in frecuencias:
            frecuencias[subcadena] += 1
        else:
            frecuencias[subcadena] = 1

    # Buscar la subcadena con la mayor frecuencia
    max_frecuencia = 0
    clave = ""
    for subcadena in frecuencias:
        if frecuencias[subcadena] > max_frecuencia:
            max_frecuencia = frecuencias[subcadena]
            clave = subcadena

    return clave
# Leer la entrada
print("Ingrese el número de caracteres de la clave y el mensaje:")
entrada = input("Ingrese mensaje: ").strip().split()
n = int(entrada[0])
texto = entrada[1]

# Mostrar la clave
print("La contraseña secreta es:", encontrar_clave(n, texto))
```

1.1.5 Explicación del código

Este código resuelve el problema de encontrar la subcadena más frecuente de longitud n dentro de un texto, lo cual se utiliza para descubrir una "contraseña secreta" escondida en un mensaje codificado. La función `encontrar_clave` recibe dos argumentos: el tamaño de la subcadena (`n`) y el texto donde se buscará.

Primero, se crea un diccionario llamado `frecuencias` para contar cuántas veces aparece cada subcadena de longitud n en el texto. El ciclo `for` recorre el texto, extrayendo todas las posibles subcadenas de tamaño n y actualizando su frecuencia en el diccionario. Si la subcadena ya existe en el diccionario, su contador se incrementa; si no, se inicializa en 1.

Luego, el código busca cuál de todas las subcadenas encontradas tiene la mayor frecuencia. Para esto, recorre el diccionario y compara la frecuencia de cada subcadena con el valor máximo encontrado hasta el momento. Si encuentra una subcadena con mayor frecuencia, actualiza tanto el valor máximo como la clave correspondiente.

Finalmente, el programa solicita al usuario que ingrese el tamaño de la clave y el mensaje codifica

Explicación by Copilot.