



Laboratorio 4 - Strings y Ciclos For

Objetivos

Parte 1: Strings

Introducción

Actividad 1: Índices positivos y negativos

Actividad 2: Substrings y el operador rebanador (*slice*)

Actividad 3: Operaciones y funciones comunes sobre strings

Actividad 4: Métodos `upper()` y `lower()`

Actividad 6: Comparar strings

Parte 2: Recorriendo Strings

Actividad 1: Recorriendo con un Ciclo For

Actividad 2: Recorriendo con un Ciclo While

Actividad 3: Ejercitemos

Parte 3: Desafíos

Desafío 1: Durante la época de las clases online ...

Desafío 2: **Telégrafo**

Desafío 3: KiwiAirlines

Desafío 4: El arte de mezclar colores

Objetivos

1. Resolver problemas que requieran del uso de *strings*, incluidos los que necesitan comparar *strings* lexicográficamente.
2. Procesar *strings* – aplicando iteración con `while` y `for` sobre los caracteres que los forman– para buscar patrones y/o construir otros *strings*.

Parte 1: Strings

Introducción

En este laboratorio aprenderemos a trabajar con *strings* (cadenas de texto) en Python. Hasta ahora hemos usado strings para mostrar mensajes o recibir

datos con `input()`, pero no habíamos explorado cómo están formados ni cómo procesarlos.

Un string es una **secuencia de caracteres** encerrados entre comillas simples (`' '`) o dobles (`" "`), y se comporta como una **colección ordenada**: podemos acceder a sus letras, recorrerlos con ciclos, compararlos, transformarlos, y mucho más.

Algunos ejemplos de strings:

```
mensaje = "Hola"
nombre = 'Ada Lovelace'
vacio = ""
```

Actividad 1: Índices positivos y negativos

Cada carácter de un string tiene una **posición**, llamada índice. Por ejemplo, el string `"Monty Python"` tiene 12 caracteres (los espacios en blanco cuentan como un caracter), y el **índice** de cada uno de los caracteres es este:

0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n



En Python, si leemos los índices de los caracteres de un string de **izquierda a derecha**, siempre comienzan en **0**. Es decir, el primer caracter tiene el índice cero, el segundo caracter el índice 1, y así sucesivamente.

Podemos acceder a las letras - o caracteres - individuales de un string usando corchetes `[]` e indicando el **índice** del caracter. Por ejemplo, si guardamos nuestro string en una variable llamada `texto`:

```
texto = "Monty Python"
print(texto[0]) # Imprimirá el caracter del índice 0, es decir, 'M'
print(texto[3]) # Imprimirá el caracter del índice 3, es decir, 't'
```

También podemos usar **índices negativos** para contar desde el final de un string hacia su inicio, es decir, de **derecha a izquierda**. En este caso,

comenzamos desde el -1 en vez de 0:

M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
texto = "Monty Python"
print(texto[-1]) # Imprimirá el caracter del índice -1, es decir, 'n'
print(texto[-7]) # Imprimirá el caracter del índice -7, es decir, el espacio ' '
print(texto[-11]) # Imprimirá el caracter del índice -11, es decir, 'o'
```

Veamos algunos ejercicios con índices. **Primero selecciona la respuesta sin ayuda, y luego ejecuta los programas en Python para ver si tu respuesta fue la correcta:**

1. ¿Qué imprime este código?

```
nombre = "astronauta"
print(nombre[5])
```

- a. 'n'
- b. 'a'
- c. 'u'
- d. 't'

2. ¿Cuál es el índice de la letra 'r' en "murciélago" ?

- a. 5
- b. 4
- c. 3
- d. 2

3. ¿Qué imprime este código con índices negativos?

```
palabra = "galaxia"
print(palabra[-3])
```

- a. 'i'

- b. 'a'
- c. 'x'
- d. 'l'

4. Completa el código para que se imprima la última letra:

```
fruta = "sandía"  
print(fruta[____])
```

💡 Debería imprimir: **a**

- ¿Hay otro índice que sirva para imprimir esa última letra?
5. Escribe un programa que imprima la tercera letra de un string ingresado por el usuario. Asume que los strings que se ingresen siempre tendrán por lo menos 3 letras.

Actividad 2: Substrings y el operador rebanador (*slice*)

Adicionalmente, podemos utilizar los corchetes para obtener parte de un string, es decir, generar *substrings*. A esto le llamamos utilizar el operador *slice* (rebanar en inglés).

El operador de slice se utiliza ahora con dos índices: `[inicio : fin]` entrega un string con todos los caracteres desde el índice **inicio**, hasta el **índice fin-1**. Veamos cómo funciona:

```
texto = "Monty Python"  
print(texto[6:10]) # Imprimirá un substring desde el índice 6 al 9, es decir, 'Pyth'
```

Visualmente lo podemos entender así:

0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n

Podemos también hacer *slice* con índices negativos:

```
texto = "Monty Python"
```

```
print(texto[-12:-7]) # Imprimirá un substring desde el índice -12 al -7, es decir,
```

M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
[-12:-7]											

? ¿Qué ocurre si no ponemos el inicio o el fin en el operador de *slice*?

Prueba correr el siguiente código en Python y responde:

```
texto = "Monty Python"
print(texto[:3])
print(texto[:-7])
print(texto[:0])
print(texto[-12:])
print(texto[-1:])
print(texto[:])
```

- ¿Qué ocurre si no ponemos el índice de inicio?
- ¿Y si no ponemos el índice del final?
- ¿Y si solo dejamos los : ?

? ¿Qué ocurre si intentamos acceder a un índice fuera de los rangos de un string?

Prueba correr el siguiente código en Python y responde:

```
texto = "Monty Python"
print(texto[12])
```

- ¿Por qué muestra un error?
- ¿Cuál es el máximo y mínimo índice entre los que se puede acceder a un string?

Veamos algunos ejercicios con operadores de slice. **Recuerda primero responder sin ayuda, y luego verificar corriendo el código en Python:**

1. ¿Cuál será el resultado?

```
mensaje = "universidad"  
print(mensaje[3:8])
```

- a. `'ivers'`
- b. `'iversi'`
- c. `'versi'`
- d. `'versid'`

2. ¿Qué imprime este código?

```
animal = "elefante"  
print(animal[1:4])
```

- a. `'elef'`
- b. `'lef'`
- c. `'ele'`
- d. `'lefa'`

3. ¿Qué imprime este código?

```
s = "dinamita"  
print(s[2:])
```

- a. `'dinamita'`
- b. `'namita'`
- c. `'inamita'`
- d. `'amita'`

4. ¿Qué imprime este código?

```
s = "ciencia"
```

```
print(s[:3])
```

- a. 'cie'
- b. 'cien'
- c. 'enci'
- d. 'ciencia'

5. Completa para imprimir solo "versi" a partir de la palabra "universidad"

```
palabra = "universidad"  
print(palabra[____:____])
```

6. ¿Qué resultado imprime este código?

```
frase = "Hola mundo"  
print(frase[-5:-1])
```

- a. 'mund'
- b. 'mundo'
- c. 'undo'
- d. 'l mu'

7. ¿Cuál es la diferencia entre `palabra[2:5]` y `palabra[2:-5]` si `palabra = "caramelo"` ?

Explica en tus palabras qué están haciendo esas dos expresiones y qué partes del *string* extraen.



Los Strings son INMUTABLES

¿Qué significa que los strings sean inmutables?

En Python, cuando decimos que un tipo de dato es **inmutable**, significa que **no se puede modificar directamente después de haber sido creado**.

En el caso de los **strings**, una vez que creas uno, **no puedes cambiar sus caracteres individuales** usando asignación.

Es decir, **no puedes hacer algo como esto**:

```
palabra = "perro"
palabra[0] = "t" # ❌ Esto lanza un error
```

Este código lanza un error porque estás tratando de cambiar directamente el primer carácter (**'p'**) por **'t'** , y eso no se permite con strings.

¿Cómo se podrían modificar entonces los strings?



La forma de "modificarlos" es **crear uno nuevo**, a partir del original.

```
palabra = "perro"
nueva_palabra = "t" + palabra[1:] # Creamos un nuevo string
print(nueva_palabra) # Imprime: 'terro'
```

Lo que hicimos fue **tomar todo menos la primera letra** de **palabra** usando slicing (**palabra[1:]** que da **'erro'**), y luego **concatenar** con la nueva letra **'t'** .

Actividad 3: Operaciones y funciones comunes sobre strings

Al igual como aprendimos con los números, en Python podemos hacer algunas operaciones y usar funciones sobre strings, pero funcionan un poco diferente:

- **Concatenación de strings (operador )**: podemos utilizar el operador  para unir strings con otros. Por ejemplo:

```
palabra1 = "par"
palabra2 = "aguas"
```



```
print(palabra1 + palabra2 + "!") # Concatenamos 3 strings, se imprimirá 'p
```

- **Repetición de strings (operador `*`):** podemos utilizar el operador `*` para repetir un mismo string varias veces. Por ejemplo:

```
risa = "ja"  
print(risa * 3) # Se imprimirá 'jajaja'
```

- **Ver si un string está dentro de otro (función `in`):** la función `in` nos devolverá `True` si un string está contenido dentro de otro, o `False` en caso contrario. Se suele utilizar mucho en condiciones de los `if`. Por ejemplo:

```
print("verso" in "universo") # Imprime True  
print("v" in "universo") # Imprime True  
print("planeta" in "universo") # Imprime False
```

- **Ver si un string no está dentro de otro (función `not in`):** también se puede negar la función `in` con un `not` para confirmar que un string **no esté dentro** de otro. Por ejemplo:

```
print("casa" not in "barrio") # Imprime True  
print("casa" not in "casamiento") # Imprime False
```

- **Conocer el largo de un string (función `len`):** la función `len` nos entrega un **número entero** que equivale a la cantidad de caracteres que tiene un string. Por ejemplo:

```
print(len("hola")) # Imprime 4 porque "hola" tiene 4 caracteres  
print(len("este es un string muy largo")) # Imprime 27. Recuerda que los e:
```

Apliquemos lo aprendido de operadores y funciones sobre strings en algunos ejercicios. Con testa sin ayuda, y luego corrobora tus respuestas en Python:

1. ¿Qué imprime el siguiente código?

```
texto = "Hola"  
resultado = texto * 2
```

```
print(resultado)
```

- a. "HolaHola"
- b. "Hola Hola"
- c. "H o l a H o l a"
- d. "HHoollaa"

2. ¿Qué imprime?

```
nombre = "Ana"  
print("n" in nombre)
```

- a. `False`
- b. `True`
- c. `"n"`
- d. Error

3. Completa el código para que imprima la cantidad de caracteres del string:

```
mensaje = "Hola mundo"  
# completar aquí
```

4. Crea un programa que reciba un string ingresado por el usuario, y luego

- Si el string contiene la palabra `"ito"` o `"ita"` debe imprimir el mensaje `"Es una palabra chiquitita"`.
- En caso contrario, debe imprimir `"Es una palabra grande"`.

Actividad 4: Métodos `upper()` y `lower()`

Los métodos `.upper()` y `.lower()` permiten cambiar un string a mayúsculas y minúsculas, respectivamente. Veamos un ejemplo:

```
palabra = "Hola"  
print(palabra.upper()) # Se imprime "HOLA"  
print(palabra.lower()) #Se imprime "hola"  
print(palabra) # La variable palabra sigue almacenando al string original "Hola"
```



Importante: los métodos `.upper()` y `.lower()` generan un **nuevo string** a partir del valor de string original. Es decir, **no modifican al string original**.

Por lo tanto, si necesitas utilizar el nuevo string generado en mayúsculas o minúsculas, es importante **guardarlo en una variable**:

```
palabra = "Hola"
palabra_en_mayusculas = palabra.upper()
print(palabra) # Se imprimirá "Hola"
print(palabra_en_mayusculas) # Se imprimirá "HOLA"
```

Veamos algunos ejercicios utilizando los métodos:

1. ¿Qué imprime?

```
nombre = "Juan"
print(nombre.upper())
```

- a. `JUAN`
- b. `juan`
- c. `Juan`
- d. Error

2. ¿Qué imprime?

```
grito = "AHHH"
print(grito.lower())
```

- a. `"AHHH"`
- b. `"ahhh"`
- c. `"ahHH"`
- d. Error

e. ¿Cuál es el resultado de este código?

```
saludo = "hola"  
print(saludo.upper() + " AMIGOS")
```

- a. "HOLA AMIGOS"
- b. "hola AMIGOS"
- c. "holaAMIGOS"
- d. "HOLAAMIGOS"

3. Identificador codificado:

Un estudiante quiere generar un "identificador" con las 3 primeras letras de su nombre en mayúsculas, seguidas de su edad repetida dos veces. Completa el siguiente programa para que haga lo solicitado:

```
nombre = "Camila"  
edad = "19"  
  
# COMPLETAR para obtener el string: "CAM1919"
```

4. ¿Está en el saludo?:

Escribe un programa que diga si la palabra "hola" está dentro de un mensaje, sin importar si está en mayúsculas o minúsculas.

```
mensaje = "¡HOLA AMIGOS!"  
  
# COMPLETAR para que imprima True
```

5. Mensaje cifrado simple:

Completa el siguiente programa que contiene un mensaje, y debe devolver solo las letras del medio (sin la primera ni la última), todo en mayúsculas.

```
mensaje = "secreto"  
  
# COMPLETAR para obtener "ECRET"
```

6. Generador de contraseñas:

Escribe un programa que reciba una palabra clave y:

1. Le agregue tres al inicio y final.
2. La convierta a mayúsculas.
3. Imprima su longitud final.

Por ejemplo, si el usuario ingresa la palabra `"secreto"`, el programa debe imprimir:

```
Ingrese una palabra: secreto
La contraseña es: ***SECRETO***
La longitud de la contraseña es 13
```

7. Primera y última letra:

Escribe un programa que reciba un string y diga si su primera y última letra son iguales (sin importar si son mayúsculas o minúsculas).

Por ejemplo:

```
Ingresar una palabra: AleGría
La primera y última letra son iguales
```

```
Ingresar una palabra: espera
La primera y última letra son diferentes
```

Actividad 6: Comparar strings

Se pueden utilizar los operadores `<`, `<=`, `>`, `>=`, `==` y `!=` para comparar strings entre sí.

Los operadores de igualdad `==` y desigualdad `!=` funcionan tal como los hemos estado utilizando. Solo hay que tener cuidado con las **mayúsculas y minúsculas**, ya que en Python se consideran caracteres diferentes aunque sean la misma letra. Por ejemplo:

```
print("hola" == "hola") # True
print("hola" == "chao") # False
```

```
print("hola" == "HOLA") # False
print("hola" != "chao") # True
```

Por otro lado, los operadores `<`, `<=`, `>`, `>=` también se pueden utilizar, pero para entender cómo saber si un string es mayor o menor que otro hay que entender cómo se representan los caracteres a partir del código ASCII.

El código ASCII asocia cada caracter imprimibles (letras, números y símbolos) con un número específico que no se repiten. Estos números se utilizan para decidir el ordenamiento de los caracteres:

decimal	caracter	decimal	caracter
48	"0"	65	"A"
49	"1"	66	"B"
...
61	"="	97	"a"
62	">"	98	"b"
...

Si les da curiosidad, pueden revisar la tabla ASCII completa [aquí](#).

La idea no es aprenderse de memoria los códigos ASCII de cada uno de los caracteres, pero sí entender que **si comparamos entre strings, los strings con un código ASCII más pequeño serán menores que un caracter con un código más grande.**

Hay algunas reglas generales también (aunque hay excepciones):

- Los números y símbolos tienen códigos ASCII más pequeños que las letras mayúsculas.
- A su vez las letras mayúsculas tienen un código menor que las minúsculas.
- También si comparamos letras entre sí, sus códigos van en orden alfabético, por lo que `"a"` tiene un código menor que `"z"`.

Por ejemplo:

```
print("H" < "h")
print("3" < "M")
```

```
print("c" < "d")
```



¿Y qué ocurre si queremos comparar strings que tienen más de un caracter?

Cuando comparas strings con operadores como `<`, `>`, `==`, etc., **Python compara los caracteres de izquierda a derecha, uno por uno**, usando su **código ASCII**.

Por ejemplo:

```
print("gato" < "perro")
```

Python compara:

1. `'g'` vs `'p'` → como `'g'` viene antes que `'p'`, ya sabe que `"gato" < "perro"`

Resultado: `True`

Veamos otro ejemplo:

```
print("hola" < "hormiga")
```

1. `'h'` vs `'h'` → iguales
2. `'o'` vs `'o'` → iguales
3. `'l'` vs `'r'` → como `'l' < 'r'`, entonces `"hola" < "hormiga"`

Resultado: `True`

Por último, veamos qué pasa cuando un string es prefijo del otro

```
print("sol" < "solcito")
```

1. `'s'` vs `'s'` → iguales
2. `'o'` vs `'o'` → iguales
3. `'l'` vs `'l'` → iguales

→ pero el primer string **se acaba**, así que `"sol"` es considerado **menor** que `"solcito"`

Resultado: `True`

Reglas clave:

La comparación para en cuanto encuentra caracteres distintos.

Si uno de los strings es más corto **pero todos los caracteres anteriores son iguales**, ese string es considerado menor.

Parte 2: Recorriendo Strings

Como los strings son una secuencia de caracteres, podemos recorrerlos utilizando ciclos. Esto quiere decir que en cada iteración del ciclo tomaremos de a uno en uno cada caracter del string.

Actividad 1: Recorriendo con un Ciclo For

El ciclo `for` es otra forma de **repetir instrucciones**, tal como el ciclo `while` que ya conocemos. Pero a diferencia del `while`, el `for` se usa cuando **ya sabes cuántas veces quieres repetir algo**, o cuando quieres **recorrer una secuencia**, como un string que es una secuencia de caracteres.

El ciclo for se ejecuta un **número predeterminado de veces** y este número **siempre se conoce de antemano**, a diferencia del ciclo while que puede ejecutarse distinta cantidad de veces dependiendo de la condición definida y los datos de usuario.

La estructura del ciclo for es la siguiente:

```
for i in secuencia:  
    sentencias a ejecutar  
    (hacer algo con el valor de i)
```

El ciclo termina cuando **i** toma el **último valor** de la secuencia.

i es una variable (puede tener otro nombre) que va a tomar **de uno en uno** los valores de la secuencia (por ejemplo, cada carácter de un string).

Ejemplo 1: como un string es una **secuencia de caracteres**, entonces lo podemos recorrer con `for`. Por ejemplo, podemos imprimir cada una de las letras de una palabra:

```
palabra = "hola"
for letra in palabra:
    print(letra)
```

La salida sería:

```
h
o
l
a
```

En este caso, cada vuelta del `for`, la variable `letra` toma el siguiente carácter del string.

Ejemplo 2: también podemos usar un ciclo for para contar cuántas veces aparece una letra en un string:

```
mensaje = "abracadabra"
contador = 0

for letra in mensaje:
    if letra == "a":
        contador += 1

print("La letra 'a' aparece", contador, "veces.")
```

Actividad 2: Recorriendo con un Ciclo While

También podemos utilizar un ciclo `while` para recorrer un string caracter por caracter. En este caso, necesitaríamos una variable **contadora** que representará el **índice de cada caracter dentro del string**, por lo que **partirá en 0 y terminará en el el largo del string - 1**:

```
palabra = "hola"
cont = 0 # Necesitamos un contador que ira desde 0 al largo de palabra - 1
```

```
while cont < len(palabra): # Mientras cont sea menor que el largo total del string
    letra = palabra[cont] # La letra en cada iteración será el caracter en el índice
    print(letra)
    cont += 1
```


Si comparamos esta forma de recorrer un string utilizando el ciclo `while`, versus la forma utilizando el ciclo `for`, podremos ver que **hacen exactamente lo mismo**, pero usando el ciclo `for` **el código es más corto y claro**.

Actividad 3: Ejercitemos

1. Contar vocales:

- a. Completa el código que cuenta cuántas vocales hay en un string utilizando un ciclo `while`:

```
texto = input("Ingresa un texto: ")
i = 0
vocales = 0
while i < _____:
    letra = texto[i]
    if _____:
        vocales += 1
    i += 1
print("Hay", vocales, "vocales.")
```

 **Tip:** el operador `in` puede ser muy útil en este caso.

- b. Escribe el mismo programa, pero esta vez utilizando un ciclo `for` en vez de un ciclo `while`. Luego describe con tus palabras qué cambios tuviste que hacer.

2. Analizador de contraseñas

Crea un programa que reciba una contraseña ingresada por el usuario y diga si es segura. Una contraseña es segura si:

- Tiene al menos 8 caracteres
- Contiene una letra mayúscula
- Contiene al menos un número

3. Detector de palabras espejo

En un lenguaje secreto, las palabras espejo son aquellas que se leen igual al revés (como "oso", "reconocer", "ana"). Haz un programa que reciba una palabra y diga si es una palabra espejo o no.

Importante: No puedes usar funciones como `[::-1]` ni métodos como `.reverse()`, debes hacerlo recorriendo el string con un ciclo.

Guíate con los siguientes ejemplos de ejecución:

```
Palabra: radar
Resultado: ES PALABRA ESPEJO
```

```
Palabra: planeta
Resultado: NO ES PALABRA ESPEJO
```

4. Comprimidor de letras consecutivas

En un sistema de compresión, se quiere simplificar palabras largas repitiendo solo la letra y cuántas veces seguidas aparece. Haz un programa que reciba un string, por ejemplo, `aaabbc`, y cree uno nuevo que tenga la forma: `a3b2c1`

Debe recorrer el string y contar cuántas veces seguidas aparece cada letra. Guíate por el siguiente ejemplo de ejecución:

```
Palabra: hheeellooo
Resultado: h2e3l2o3
```

5. Primer y último carácter

Para un análisis de texto, necesitas saber la primera y la última vez que aparece cierta letra en un string. Haz un programa que reciba una palabra y una letra, y diga en qué posiciones aparece por primera y última vez la letra.

Guíate por el siguiente ejemplo de ejecución:

```
Palabra: programacion
Letra: o
Resultado:
```

Primera aparición: índice 2
Última aparición: índice 10

6. Codificador de vocales

Un sistema de codificación básico reemplaza cada vocal por un número:

- a → 1
- e → 2
- i → 3
- o → 4
- u → 5

Haz un programa que reciba un texto y genere una versión donde cada vocal haya sido reemplazada por su número. Todas las demás letras quedan igual.

Guíate por el siguiente ejemplo de ejecución:

Texto: hola mundo
Resultado: h4l1 m5nd4

Parte 3: Desafíos

Desafío 1: Durante la época de las clases online ...

Durante la época de clases online que se vivió en la pandemia, para evitar cualquier tipo de problemas los profesores y profesoras permitieron el acceso a las clases solamente con las cuentas institucionales.

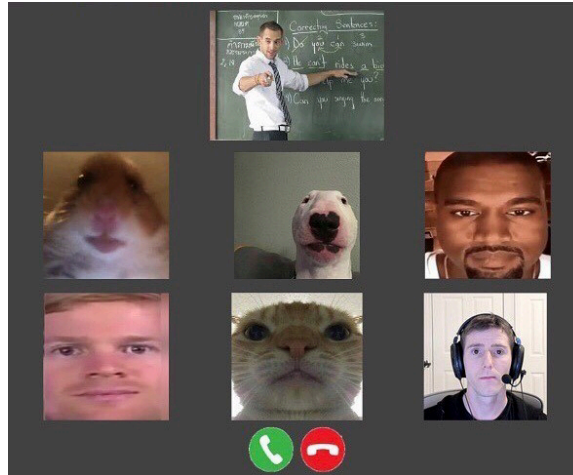


Figura 1: Una clase cualquiera.

Hasta 2019, las cuentas de todos los alumnos y alumnas de la Universidad terminaban en

`@sansano.usm.cl`, pero últimamente se prefieren las cuentas `@usm.cl`. Sin embargo, ambos

dominios siguen siendo utilizados por los estudiantes y ambos son válidos.

Escribe un programa que revise si se está intentando entrar a la clase con una cuenta válida (que tenga como dominio `@sansano.usm.cl` o `@usm.cl`).

Guíate con el siguiente ejemplo de ejecución:

Ingrese su correo: kiwi.progra@sansano.usm.cl
Correo valido!

Ingrese su correo: sneki.progra@gmail.com
Dominio invalido!

▼ 👁 Pista

El caracter `@` permite identificar donde comienza el dominio.

Desafío 2: Telégrafo

Durante esta última semana, en tu juego favorito - en el que por alguna razón se simula una oficina de correos virtual en medio de las batallas para ganar oro extra - has estado recibiendo muchos nuevos encargos.

El último encargo te llegó desde las oficinas de correos pidiendo ayuda para entregar un paquete perdido. Para entregar el paquete tienes que enviar un mensaje de aviso al receptor utilizando un telégrafo. En la oficina te proporcionan los costos de enviar un mensaje:

- Cada letra cuesta: \$10
- Los caracteres especiales que no sean letras cuestan: \$30.
- Los dígitos (números) cuestan: \$20.
- Los espacios no tienen valor.

Ya que probablemente te lleguen muchos encargos como este, decides escribir un programa que te ayude a calcular los costos de los mensajes. El programa debe recibir el mensaje original y luego calcular su costo y mostrarlo por pantalla. Las letras del castellano (ñ, á, é, í, ó, ú) se consideran caracteres especiales.

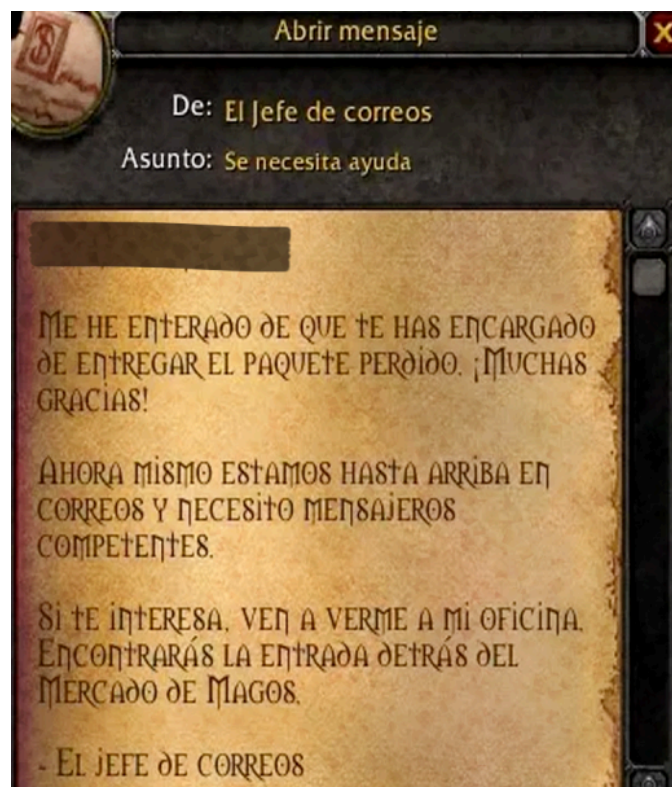


Figura 2: El correo que recibiste.

Guíate con los siguientes ejemplos de ejecución:

Ingrese su mensaje: Feliz Aniversario!

El costo de su mensaje es \$190

Ingrese su mensaje: Hola! me gusta usar el telégrafo. El telégrafo es genial :D
El costo de su mensaje es \$590

▼ 👁 Pista

Recuerda que utilizando `.lower()` puedes convertir el mensaje en minúsculas.
El operador `in` puede ser muy útil para la solución de este ejercicio.

Desafío 3: KiwiAirlines

La empresa KiwiAirlines requiere implementar un sistema de asignación de asientos para sus aviones.

La idea que les presenta el gerente de la empresa es la siguiente:

- La disponibilidad de los asientos se mantiene en un string.
- Los asientos vacíos se marcan con la letra `v`, ocupados de clase económica con la letra `e` y ocupados de la clase business con la letra `b`.

Por ejemplo, si el string es `"bbevv"` quiere decir que hay 6 puestos (**enumerados de 1 a 6**):

los puestos 1 y 2 están ocupados por pasajeros de la clase business, los puestos 3 y 4 están ocupados por pasajeros de la clase económica, y los puestos 5 y 6 están vacíos.



Figura 3: KiwiAirlines en su glori

Escribe un programa que al inicio le permita al usuario ingresar la cadena de caracteres con los asientos utilizados y no utilizados. Luego, le debe permitir ingresar la clase de asiento (Económico o Business) y el número del asiento que el pasajero desea utilizar. Si el asiento se encuentra **ocupado**, **se deberá volver a ingresar otro número de asiento**.

El programa deberá solicitar clases y números de asiento hasta que se ingrese el valor "Salir" como tipo de asiento, o bien, hasta que ya no queden más asientos disponibles (vacíos).

Una vez que termine el programa, se debe mostrar el total de dinero recaudado por las ventas, es decir, de todos los asientos de la fila (no solo los recién vendidos), considerando que las tarifas son:

- \$10.000 para clase económica
- \$30.000 para la clase business.

Guíate con los siguientes ejemplos de ejecución:

```
Ingrese la cadena: vbbeev
Ingrese la clase: Economico
Ingrese el número de su asiento: 1
Asignado!
Ingrese la clase: Business
Ingrese el número de su asiento: 3
Ocupado!
Ingrese el número de su asiento: 6
Asignado!
Ingrese la clase: Salir
-----Reporte de ventas-----
Total: $140000
```

```
Ingrese la cadena: bbb
Avión lleno!
-----Reporte de ventas-----
Total: $90000
```

```
Ingrese la cadena: eveebvvvv
Ingrese la clase: Business
```

Ingrese el número de su asiento: 1
Ocupado!
Ingrese el número de su asiento: 2
Asignado!
Ingrese la clase: Salir
-----Reporte de ventas-----
Total: \$90000

▼ 👁 Pista

Para resolver este ejercicio, es buena idea separarlo en dos partes: primero, desarrolla la lógica de la compra de asientos, que incluye verificar si el asiento está disponible o no. Luego, cuando ya no se vayan a ingresar más asientos, ya sea porque el avión está lleno o porque se ingresó "Salir", recién recorre el string final de asientos para calcular el monto total vendido.

Desafío 4: El arte de mezclar colores

Después de trabajar mucho, te compras una tablet digital para finalmente aprender a dibujar. Si bien podrías dibujar con lápices y pinturas, la última vez que lo intentaste fue un desastre: la mesa todavía tiene las manchas de acrílico rojo, dándole un toque digno de una película de horror.



Figura 4: Tu ídolo y motivación.

La aplicación que ocupas en tu tablet simula totalmente la pintura clásica. Es decir, para obtener nuevos colores tienes que mezclarlos. Ya sabes qué colores

necesitas, y también tienes un string de combinaciones que contiene los colores que, al mezclarse, dan un color específico.

Este string tiene la siguiente forma (este es solo un posible ejemplo):

```
"rojo+blanco=rosa;azul+blanco=celeste;"
```

Desarrolla un programa que reciba un string de combinaciones y el color requerido, y muestre por pantalla un string con todos los colores necesarios para hacer las mezclas, separados por `"+"`.

Guíate con los siguientes ejemplos de ejecución:

```
Ingresa las combinaciones: rojo+blanco=rosa;azul+blanco=celeste;  
Que color deseas: celeste  
azul+blanco
```

```
Ingresa las combinaciones: rojo+blanco=rosa;azul+blanco=celeste;  
Que color deseas: dorado  
No puedes obtener este color :(
```

▼ 👁 Pista

Para poder encontrar el color deseado dentro del string de combinaciones, fíjate en los símbolos que separan las palabras: el resultado siempre viene después del signo `"="`, y antes de `","`.

Lo mismo ocurre con los colores necesarios para hacer esa combinación: van antes del signo `"="`.