



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



MATERIA: Teoría Computacional.

PRÁCTICA: Práctica 2. Manejo de cadenas en Python.

ALUMNO: Meza Vargas Brandon David.

PROFESOR: Jorge Luis Rosas Trigueros.

FECHA PRÁCTICA: 23-oct-2020

FECHA DE ENTREGA: 06-nov-2020



MARCO TEÓRICO

En este apartado se verán algunas funciones de Python que nos fueron de utilidad para resolver los problemas propuestos en esta práctica.

Clase Counter de collections

El módulo integrado de colecciones nos provee otros tipos o mejoras de las colecciones clásicas.

La clase Counter es una subclase de diccionario utilizada para realizar cuentas. En la imagen 1 se muestran dos ejemplos, en el primero cuenta la cantidad de cada número que hay en una lista y en el segundo la cantidad de cada carácter que conforma una cadena;

Código	Resultado	Código	Resultado
<pre>from collections import Counter l = [1,2,3,4,1,2,3,1,2,1] Counter(l)</pre>		<pre>Counter({1: 4, 2: 3, 3: 2, 4: 1})</pre>	
Código	Resultado	Código	Resultado
<pre>Counter("palabra")</pre>		<pre>Counter({'a': 3, 'b': 1, 'l': 1, 'p': 1, 'r': 1})</pre>	

Imagen 1. Ejemplos de Counter.

Métodos de las cadenas

En la mayoría de los problemas de esta práctica se hizo uso de los métodos de las cadenas, por lo que a continuación las veremos, así como un ejemplo de cada una.

- upper()
Devuelve la cadena con todos sus caracteres en mayúscula, un ejemplo lo vemos en la imagen 2;

```
>>> cadena= "ejemplo"  
>>> cadena.upper()  
'EJEMPLO'
```

Imagen 2. Ejemplo upper()

- lower()
Devuelve la cadena con todos sus caracteres en minúscula, un ejemplo lo vemos en la imagen 3;

```
>>> cadena="EJEMPLO"  
>>> cadena.lower()  
'ejemplo'
```

Imagen 3. Ejemplo lower()

- `title()`
Devuelve la cadena con el primer carácter de cada palabra en mayúscula, un ejemplo lo vemos en la imagen 4;

```
>>> cadena="Esto es una cadena"
>>> cadena.title()
'Esto Es Una Cadena'
>>>
```

Imagen 4. Ejemplo title().

- `count()`
Devuelve una cuenta de las veces que aparece una subcadena en la cadena, un ejemplo lo vemos en la imagen 5;

```
>>> cadena="el perro en el patio"
>>> cadena.count('el')
2
>>>
```

Imagen 5. Ejemplo count().

- `capitalize()`
Devuelve la cadena con su primer carácter en mayúscula, un ejemplo lo vemos en la imagen 6;

```
>>> cadena="soy una cadena"
>>> cadena.capitalize()
'Soy una cadena'
>>>
```

Imagen 6. Ejemplo capitalize().

- `isdigit()`
Devuelve True si la cadena es todo números y false en caso contrario, un ejemplo lo vemos en la imagen 7;

```
>>> cadena="12345"
>>> cadena.isdigit()
True
>>>
```

Imagen 7. Ejemplo de isdigit().

- `isalnum()`
Devuelve True si la cadena es todo números o caracteres alfabéticos, ejemplo en la imagen 8;

```
>>> cadena="abcA2012dsa"
>>> cadena.isalnum()
True
>>>
```

Imagen 8. Ejemplo isalnum().

- `isalpha()`
Devuelve True si la cadena es todo caracteres alfabéticos, ejemplo en la imagen 9;

```
>>> cadena="abcA2012dsa"
>>> cadena.isalpha()
False
```

Imagen 9. Ejemplo isalpha().

- islower()
Devuelve True si la cadena es todo minúsculas, ejemplo en la imagen 10;

```
>>> c="Soy una cadena"
>>> c.islower()
False
```

Imagen 10. Ejemplo islower().

- isupper()
Devuelve True si la cadena es todo mayúsculas, ejemplo en la imagen 11;

```
>>> c="SOY UNA CADENA"
>>> c.isupper()
True
```

Imagen 11. Ejemplo isupper().

- startswith()
Devuelve True si la cadena empieza con una subcadena, un ejemplo lo vemos en la imagen 12;

```
>>> c="soy una cadena"
>>> c.startswith("soy")
True
```

Imagen 12. Ejemplo startswith().

- endswith()
Devuelve True si la cadena acaba con una subcadena, un ejemplo lo vemos en la imagen 13;

```
>>> c="soy una cadena"
>>> c.endswith("cadena")
True
```

Imagen 13. Ejemplo endswith().

- split()
Separa la cadena en subcadenas a partir de sus espacios y devuelve una lista, podemos indicar el carácter a partir del que se separa, un ejemplo lo vemos en la imagen 14;

```
>>> c="soy,una,cadena"
>>> c.split(',')
['soy', 'una', 'cadena']
```

Imagen 14. Ejemplo Split().

- join()

Une todos los caracteres de una cadena utilizando un carácter de unión, un ejemplo lo vemos en la imagen 15;

```
>>> " ".join("Hola")
'H o l a'
```

Imagen 15. Ejemplo join()

- strip()

Borra todos los espacios por delante y detrás de una cadena y la devuelve, podemos indicar el carácter a borrar, un ejemplo lo vemos en la imagen 16;

```
>>> "*****hola*****".strip("*")
'hola'
```

Imagen 16. Ejemplo strip().

- replace()

Reemplaza una subcadena de una cadena por otra y la devuelve, podemos indicar un limite de veces a reemplazar, un ejemplo lo vemos en la imagen 17;

```
>>> "esto es es es es es una cadena".replace(' es', '', 4)
'esto es una cadena'
```

Imagen 17. ejemplo replace().

sort() : El método sort() ordena la lista de forma ascendente por defecto

Su sintaxis es: lista.sort(reverse=True|False, key=myfunc)

En donde reverse y key son opcionales, si reverse es True, se ordena de forma descendente, key sirve para dar criterios de ordenación. Un ejemplo lo vemos en la imagen 18;

```
>>> vocales = ["i","o","u","e","a"]
>>> vocales.sort()
>>> vocales
['a', 'e', 'i', 'o', 'u']
```

Imagen 18. Ejemplo sort()

Entrada por teclado

La función input() permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla Intro, como muestra podemos ver el ejemplo de la imagen 19;

```
>>> nombre=input()
pedro
>>> nombre
'pedro'
>>> |
```

Imagen 19. Ejemplo input()

Funciones en Python

La sentencia `def` es una definición de función usada para crear objetos funciones definidas por el usuario.

La definición de función no ejecuta el cuerpo de la función, esta es ejecutada solo cuando es llamada.

La sintaxis para una definición de función en Python es:

```
def NOMBRE(LISTA_DE_PARAMETROS):  
  
    """DOCSTRING_DE_FUNCION"""  
  
    SENTENCIAS  
  
    RETURN [EXPRESION]
```

sorted()

Esta función ordena los elementos en un orden específico (ascendente o descendiente) y retorna la ordenación como una lista.

La sintaxis de esta función es: `sorted (string, tupla, lista o diccionario, key=None, reverse=False)`

`reverse`: es opcional, si está en `True` ordena en forma descendente.

`key`: es opcional y es el valor que se tiene en cuenta para hacer la comparación.

Un ejemplo lo vemos en la imagen 20;

```
>>> lista=['r','a','d','z','o']  
>>> sorted(lista)  
['a', 'd', 'o', 'r', 'z']
```

Imagen 20. Ejemplo con sorted().

list index()

El método `index()` retorna la posición de la primera ocurrencia de un valor especificado

Su sintaxis es: `list.index(elemento)`

Donde el elemento puede ser cualquier tipo, será el elemento a buscar. Un ejemplo lo vemos en la imagen 21:

```
>>> lista=[1,3,2,4]  
>>> lista.index(4)  
3
```

Imagen 21. Ejemplo index().

Formateo

El uso básico del método `str.format()` es como esto:

```
>>> print('We are the {} who say "{}!"'.format('knights', 'Ni'))  
We are the knights who say "Ni!"
```

Las llaves y caracteres dentro de las mismas (llamados campos de formato) son reemplazadas con los objetos pasados en el método `str.format()`. Un número en las llaves se refiere a la posición del objeto pasado en el método `str.format()`.

Dentro de las llaves podemos añadir un tipo de formateo, aquí algunos de ellos:

:< alinea el resultado a la izquierda

:> alinea el resultado a la derecha

:^ alinea el resultado al centro

:+ usa un signo positivo para indicar si el resultado es positivo o negativo

:- usa un signo menos para valores negativos

:, usa una coma como un separador de miles

:d formato decimal

:f formato de punto fijo

:x formato hexadecimal en minúsculas

:X formato hexadecimal en mayúsculas

:% formato de porcentaje

zfill()

Este método añade ceros al inicio de una cadena hasta alcanzar una longitud específica.

maketrans()

Este método se utiliza para crear asignaciones de carácter tabla de conversión. El primer argumento es una cadena que representa el carácter que desea convertir, el segundo parámetro es la representación de cadena de la conversión metas.

Este método crea una tabla de translación que usa el método **translate()**, este método retorna una cadena donde cada carácter es mapeado a su correspondiente carácter en la tabla de translación. Un ejemplo de estas dos funciones en la imagen 22:

```
>>> cantidad="32.054,23"
>>> maketrans=cantidad.maketrans
>>> cantidad = cantidad.translate(maketrans(','.',' ','.''))
>>> cantidad
'32,054.23'
```

Imagen 22. Ejemplo usando maketrans() y translate().

Listas de comprensión

Es una funcionalidad que nos permite crear listas avanzadas en una misma línea de código. Esto se ve mucho mejor en la práctica, unos ejemplos a continuación:

Así sería un método tradicional:

```
lista = []
for letra in 'casa':
    lista.append(letra)
print(lista)
```

Siendo la salida: ['c', 'a', 's', 'a']

Este ejemplo usando la comprensión de listas:

```
lista = [letra for letra in 'casa']
print(lista)
```

Siendo la misma salida: ['c', 'a', 's', 'a']

MATERIAL Y EQUIPO

Para la realización de esta práctica son necesarias las siguientes herramientas:

- Un equipo de cómputo que cumpla con los requerimientos para el uso del lenguaje de programación Python.
- Tener instalado el lenguaje de programación Python.
- Contar con un IDE para programar con Python, cualquiera es útil.

DESARROLLO

En el desarrollo de la práctica se mostrarán 43 problemas relacionados a la manipulación de cadenas en Python, en donde haremos uso de los métodos y funciones presentadas en el marco teórico de esta práctica.

1.- En el primer programa no se presentó ningún detalle ya que solo es calcular la longitud de una cadena, aquí usamos len(), el código junto con su salida lo podemos ver en la imagen 23;


```
#Programa No.1
print("Programa 1")
cadena="Hola"

print("La longitud de la cadena", cadena, "es: ",len(cadena));

Programa 1
La longitud de la cadena Hola es:  4
```

Imagen 23. Problema 1.

2.- En el segundo programa hicimos uso de collections y Counter para contar el numero de caracteres de una cadena, el código junto con su salida lo podemos ver en la imagen 24;

```
#Programa No.2
print("\nPrograma 2")
from collections import Counter

cadena= "Esto es una cadena"
c= Counter(cadena)

print(c)

Programa 2
Counter({' ': 3, 'a': 3, 's': 2, 'e': 2, 'n': 2, 'E': 1, 't': 1, 'o': 1, 'u': 1, 'c': 1, 'd': 1})
```

Imagen 24. Problema 2.

3.- En el programa nos piden formar una cadena con los dos primeros y dos últimos caracteres de una cadena, este problema no fue problema, pues solo se usó un if y un else, el código junto con su salida lo podemos ver en la imagen 25;

```
#Programa No.3
print("\nPrograma 3")
cadena="cadena"

if len(cadena) < 2:
    print("")
else:
    print(cadena[0:2] + cadena[-2:])

Programa 3
cana
```

Imagen 25. Problema 3

4.-En este problema use el método replace(), el cual está explicado en el marco teórico, para poder hacer lo que este problema solicitaba, el código junto con su salida, lo podemos ver en la imagen 26;

```

#Programa No.4
print("\nPrograma 4")
cadena="recordar"

cadenaRep= cadena[0] + cadena[1:].replace(cadena[0],"$")
print(cadenaRep)

Programa 4
reco$da$

```

Imagen 26. Problema 4

5.- Este problema me resulto un poco confuso al inicio, pues teníamos que formar una sola cadena a partir de dos cadenas e intercambiar algunos caracteres de ellas, pero al final lo resolví con una concatenación, haciendo uso de un indizado extendido, el código junto con su salida lo podemos ver en la imagen 27;

```

#Programa No.5
print("\nPrograma 5")
cadena1="abcabc"
cadena2="xyzxyz"

cadena1n=cadena1[:2].replace(cadena1[:2],cadena2[:2] + cadena1[2:])
cadena2n=cadena2[0:2].replace(cadena2[0:2],cadena1[0:2] + cadena2[2:])
print(cadena1n + ' ' + cadena2n);

Programa 5
xycabc abzxyz

```

Imagen 27. Problema 5

6.- En este problema tenemos que encontrar la longitud de la cadena más larga de una lista, fue sencillo, pues solo ordene la lista por el tamaño de las cadenas en ella y posteriormente imprimí la longitud de la última cadena, que, al estar ordenada sería la más grande, el código junto con su salida lo podemos ver en la imagen 28;

```

#Programa No.6
print("\nPrograma 6")
palabras= ["hola", "pedro", "holapedro"]
palabras.sort(key=len);
print(len(palabras[-1]))

Programa 6
9

```

Imagen 28. Problema 6.

7.-En este problema se tenía que quitar algún carácter en una posición específica, lo conseguí realizar de una forma sencilla haciendo uso de un indizado extendido, el código junto con su salida lo podemos ver en la imagen 29;

```

#Programa 7
print("\nPrograma 7")
cadena = "hola"
pos=0
cadena=cadena[:pos]+cadena[pos+1:]
print(cadena)

Programa 7
ola

```

Imagen 29. Problema 7

8.- El programa pide formar una nueva cadena intercambiando el primer y último carácter de una cadena dada, se realizó con una concatenación de varios indizados de la cadena original, el código junto con su salida lo podemos ver en la imagen 30;

```

#Programa No.8
print("\nPrograma 8")
cadena="calabaza"

nueva=cadena[-1] + cadena[1:-1] +cadena[0]
print(nueva);

Programa 8
aalabazc

```

Imagen 30. Problema 8

9.- Aquí se eliminaron los caracteres que están en un índice impar de una cadena, se solucionó haciendo uso de un ciclo for y una condición, el código junto con su salida lo podemos ver en la imagen 31;

```

#Programa No.9
print("\nPrograma 9")
cadena= "celular"
nueva=''

for i in range(len(cadena)):
    if i%2 ==0 :
        nueva+=cadena[i]

print(nueva)

Programa 9
cllr

```

Imagen 31. Problema 9

10.- En este programa se hizo uso de Counter para contar las ocurrencias de cada palabra en una cadena, no hubo complicaciones pues es directo, el código junto con su salida lo podemos ver en la imagen 32;

```
#Programa No.10
print("\nPrograma 10")
cadena="El barco de el niño que viajo en barco el año que cumplio años"
list=cadena.split()
from collections import Counter

c= Counter(list)
print(c)

Programa 10
Counter({'barco': 2, 'el': 2, 'que': 2, 'El': 1, 'de': 1, 'niño': 1, 'viajo': 1,
'en': 1, 'año': 1, 'cumplio': 1, 'años': 1})
```

Imagen 32. Problema 10.

11.- Aquí usamos input() para recibir una entrada del usuario, además de lower() y upper(), el código junto con su salida lo podemos ver en la imagen 33;

```
#Programa No.11
print("\nPrograma 11")
print("Escribe algo: ")
entrada=input()
print("Tu entrada en minusculas es: ", entrada.lower())
print("Tu entrada en mayusculas es: ", entrada.upper())

Programa 11
Escribe algo:
pedro
Tu entrada en minusculas es:  pedro
Tu entrada en mayusculas es:  PEDRO
```

Imagen 33. Problema 11.

12.- En este problema se usó Split, para separar la cadena por comas y posteriormente ordenarla con sorted, el código junto con su salida lo podemos ver en la imagen 34;

```
#Programa No.12
print("\nPrograma 12")
cadena="bb,aa,cc,zz"
x=cadena.split(",")
orde=sorted(x)
print(orde)

Programa 12
['aa', 'bb', 'cc', 'zz']
```

Imagen 34. Problema 12

13.- Aquí nos piden obtener una cadena de 4 copias de los últimos caracteres de una cadena dada, se logra con un indizado y este ponerlo 4 veces, es decir, multiplicarlo por 4, el código junto con su salida lo podemos ver en la imagen 35;

```

#programa No.13
print("\nPrograma 13")
def trece(cadena):
    rep=cadena[-2:]*4
    return rep
print(trece("pedro"))

Programa 13
rorororo

```

Imagen 35. Problema 13

14.- Aquí se pide crear una cadena a partir de los primeros tres caracteres de una cadena si su longitud es mayor o igual que 3, no represento ningún problema este programa, el código junto con su salida lo podemos ver en la imagen 36;

```

#programa No.14
print("\nPrograma 14")
def firstTree(cadena):
    if(len(cadena) >= 3):
        res=cadena[:3]
    else:
        return cadena
    return res
print(firstTree("python"))

Programa 14
pyt

```

Imagen 36. Problema 14.

15.- Aquí tenemos que obtener la mitad de una cadena con longitud par, el problema es sencillo, pero me surgió un detalle, al momento de hacer la indización dentro del if no me dejaba ya que la división no la contaba como entero, esto lo resolví usando // ya que esto lo convierte a int, el código junto con su salida lo podemos ver en la imagen 37.

```

#programa No.15
print("\nPrograma 15")
def firstHalf(cadena):
    if(int(len(cadena)) % 2 == 0):
        n=cadena[:int(len(cadena)) // 2] ### lo convierte en int
    else:
        n="La longitud de la cadena no es par"
    return n
print(firstHalf("Holasa"))

Programa 15
Hol

```

Imagen 37. Problema 15.

16.- Aquí se tenía que invertir una cadena si su longitud era múltiplo de la suma de los tres últimos dígitos de nuestro número de boleta, no hubo mayor problema pues solo fue hacer una condición y listo, el código junto con su salida lo podemos ver en la imagen 38.

```
#programa No.16
print("\nPrograma 16")
sumaBol=2+8+8
def inver(cadena):
    if sumaBol % len(cadena) == 0:
        return(cadena[::-1])
    else:
        return "la longitud no es multiplo"
print(inver("ESCOMA"))
print(inver("ESCOM"))

Programa 16
AMOCSE
la longitud no es multiplo
```

Imagen 38. Problema 16.

17.- Este problema nos pide pasar a mayúsculas una cadena solo si tiene dos mayúsculas en los primeros 4 caracteres, se logra fácilmente incluyendo algunas condiciones y un contador, el código junto con su salida lo podemos ver en la imagen 39.

```
#programa No.17
print("\nPrograma 17")

def mayus(cadena):
    cont=0
    for c in cadena[:4]:
        if c.isupper():
            cont+=1
    if(cont>=2):
        return cadena.upper()
    else:
        return "Tiene menos de dos mayusculas"
print(mayus("Si puedO"))
print(mayus("SImon"))

Programa 17
Tiene menos de dos mayusculas
SIMON
```

Imagen 39. Problema 17.

18.- Este programa es prácticamente ordenar una cadena, lo logramos con el método join() y sorted(), el código junto con su salida lo podemos ver en la imagen 40.

```

#programa No.18
print("\nPrograma 18")

def lexi(cadena):
    return ''.join(sorted(cadena))
print(lexi("xawmzrqw"))

Programa 18
amqrwxz

```

Imagen 40. Problema 18.

19.- Aquí usamos el método `startswith()` y como parámetro especificamos el carácter a verificar si es con el que inicia una cadena dada, el código junto con su salida lo podemos ver en la imagen 41.

```

#programa No.19
print("\nPrograma 19")
cadena="pedro"
print(cadena.startswith("p"))

Programa 19
True

```

Imagen 41. Problema 19.

20.- En este problema tenemos que crear un cifrado de César, el cifrado de Cesar ya fue explicado en el marco teórico, aquí hubo varias complicaciones, la primera fue conocer en que consiste este cifrado de César para poder darnos una idea al programarlo, para hacer este problema se importó el alfabeto inglés tanto en mayúsculas como minúsculas, se logró implementando un ciclo `for` que va recorriendo nuestra cadena, dentro de este incluimos condicionales los cuales se van a encargar de ir intercambiando las letras de la cadena por las del alfabeto según el desplazamiento que indicamos, se hizo uso de `index()` para saber los índices de nuestra cadena, además de `append()` para ir guardando el cifrado de Cesar correspondiente, el código junto con su salida lo podemos ver en la imagen 42.

```

#programa No.20
print("\nPrograma 20")

from string import ascii_lowercase, ascii_uppercase #agrega el alfabeto ingles
                                                    #en minusculas y mayusculas

def cifradoCesar(cadena, despl):
    res=[]

    for i in cadena:
        if i in ascii_lowercase:
            indice = ascii_lowercase.index(i)
            newindice = (indice + despl) % len(ascii_lowercase) # modulo nos per
            res.append(ascii_lowercase[newindice])
        elif i in ascii_uppercase:
            indice = ascii_uppercase.index(i)
            newindice = (indice + despl) % len(ascii_uppercase) # modulo nos per
            res.append(ascii_uppercase[newindice])
        else:
            res.append(i)

    return ''.join(res)

print(cifradoCesar('Teoria computacional',1))
print(cifradoCesar('Teoria computacional',2))

Programa 20
Ufpsjb dpnqvubdjpbm
Vgqtkc eqorwvcekqpcn

```

Imagen 42. Problema 20.

21.- En este programa se añade un prefijo a todas las líneas de una cadena multilínea, una cadena multilínea la podemos hacer insertando "", y la función indent() nos ayuda a añadir este prefijo, el código junto con su salida lo podemos ver en la imagen 43.

```

#programa No.21
print("\nPrograma 21")

from textwrap import indent
cadena = """línea 1
línea 2
línea 3
ultima"""

print(indent(cadena,".- ")) #indent agrega un prefijo a cada linea de la cadena

Programa 21
.- línea 1
.- línea 2
.- línea 3
.- ultima

```

Imagen 43. Problema 21.

22.- Este problema fue de formateo, se logró usando una f-string, se limitaron los decimales a solo dos, el código junto con su salida lo podemos ver en la imagen 44.

```
#programa No.22
print("\nPrograma 22")

numero=142.3268
print(f"{numero:.2f}") #uso de una f-string

Programa 22
142.33
```

Imagen 44. Problema 22.

23.- De nuevo en este programa fue para formateo de flotantes, fue lo mismo que en el problema 22 pero añadiendo un signo, el código junto con su salida lo podemos ver en la imagen 45.

```
#programa No.23
print("\nPrograma 23")
numero=142.3268
print(f"{numero:+,.2f}") #uso de una f-string

Programa 23
+142.33
```

Imagen 45. Problema 23.

24.- En este problema se imprimen flotantes sin decimales, se logra usando una f-string con su formateo correspondiente, el código junto con su salida lo podemos ver en la imagen 46.

```
#programa No.24
print("\nPrograma 24")

numero=142.3268
print(f"{numero:.0f}") #uso de una f-string

Programa 24
142
```

Imagen 46. Problema 24

25.- Ahora, en este problema se trató del formateo de enteros, este nos pide llenar un entero con ceros a la izquierda hasta un ancho específico, esto se logra con la función `zfill()`, el código junto con su salida lo podemos ver en la imagen 47.

```
#programa No.25
print("\nPrograma 25")
num="50"
ancho=5
print(num.zfill(ancho))

Programa 25
00050
```

Imagen 47. Problema 25.

26.- En este programa se imprimieron * a la derecha de un entero hasta un ancho especifico, lo logramos haciendo uso del formateo con enteros, el código junto con su salida lo podemos ver en la imagen 48.

```
#programa No.26
print("\nPrograma 26")

num=50
print("{:*<5d}".format(num))

Programa 26
50***
```

Imagen 48. Problema 26

27.- Aquí se hizo uso del formateo de números, en este caso para separar un número con comas, el código junto con su salida lo podemos ver en la imagen 49.

```
#programa No.27
print("\nPrograma 27")

num=300000000
print('{:,}'.format(num))

Programa 27
30,000,000
```

Imagen 49. Problema 27.

28.- Aquí se hizo uso del formateo de porcentaje, no hubo mayores complicaciones, el código junto con su salida lo podemos ver en la imagen 50.

```
#programa No.28
print("\nPrograma 28")
num=25
print(f'{num:.2%}')

Programa 28
2500.00%
```

Imagen 50. Problema 28.

29.- En este problema se hizo uso de f-strings y un formateo para alinear el número a la izquierda, centro y derecha, no hubo problemas, el código junto con su salida lo podemos ver en la imagen 51.

```
#programa No.29
print("\nPrograma 29")

num=25
print('Original:',num)
print(f'izq(ancho 10){num:<10}')      #a la izquierda
print(f'der(ancho 10){num:>10}')      #a la derecha
print(f'cen(ancho 10){num:^10}')      #al centro

Programa 29
Original: 25
izq(ancho 10)25
der(ancho 10)      25
cen(ancho 10)      25
```

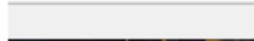


Imagen 51. Problema 29.

30.- Aquí se hizo uso del formateo para hexadecimal, no hubo mayor complicación, el código junto con su salida lo podemos ver en la imagen 52.

```
#programa No.30
print("\nPrograma 30")

def rgb(r,g,b):
    return '#{ :2x}{ :2x}{ :2x}'.format(r,g,b)

print(rgb(64,204,208))

Programa 30
#40ccd0
```

Imagen 52. Problema 30.

31.- En este programa se hizo uso de count() para contar las ocurrencias de una subcadena en una cadena, el código junto con su salida lo podemos ver en la imagen 53.

```
#programa No.31
print("\nPrograma 31")
cadena='cadena cadena cadena soy una cadena cadena'
ocurrencias=cadena.count("cadena")
print("Hay ",ocurrencias," ocurrencias")

Programa 31
Hay 5 ocurrencias
```

Imagen 53. Problema 31.

32.- En este problema me surgió una confusión, pues pensé que solo era invertir la cadena en su totalidad, pero analizándolo bien, me di cuenta que eso no tendría mucho sentido, sino que era intercambiar las palabras de una cadena, lo solucioné implementando un ciclo for, dividiendo la cadena por saltos de línea e ir la uniendo por la última parte con la función join() y un indizado, el código junto con su salida lo podemos ver en la imagen 54.

```
#programa No.32
print("\nPrograma 32")
cadena="soy una cadena"
for i in cadena.split('\n'):
    print(' '.join(i.split()[::-1]))

Programa 32
cadena una soy
```

Imagen 54. Problema 32.

33.- Aquí se hizo uso de una lista de comprensión para recorrer la cadena e ir quitando las vocales en este caso de una cadena, el código junto con su salida lo podemos ver en la imagen 55.

```
#programa No.33
print("\nPrograma 33")

cadena="institucional"
fuera='aeiou'
cad= [ c for c in cadena if c not in fuera]
print(''.join(cad))

Programa 33
nsttcnl
```

Imagen 55. Problema 33.

34.- Este programa pide contar los caracteres repetidos de una cadena, lo logré solucionar haciendo uso de Counter de collections para contar los caracteres de la cadena y por último una lista de comprensión para seleccionar los repetidos y su cantidad, el código junto con su salida lo podemos ver en la imagen 56.

```

#programa No.34
print("\nPrograma 34")

from collections import Counter
cadena="cabballo"
contador= Counter(cadena)
print(contador)
print([t[0:] for t in contador.items() if t[1]> 1])

Programa 34
Counter({'a': 2, 'b': 2, 'l': 2, 'c': 1, 'o': 1})
[('a', 2), ('b', 2), ('l', 2)]

```

Imagen 56. Problema 34.

35.- Este problema fue sencillo, pues solo fue usar enumerate() para enumerar los elementos de la cadena y con un ciclo for imprimirlos, el código junto con su salida lo podemos ver en la imagen 57.

```

#Programa 35
print("\nPrograma 35")
cadena="youtube"
for i, c in enumerate(cadena):
    print("Carácter actual ",c," posicion ",i)

Programa 35
Carácter actual  y  posicion  0
Carácter actual  o  posicion  1
Carácter actual  u  posicion  2
Carácter actual  t  posicion  3
Carácter actual  u  posicion  4
Carácter actual  b  posicion  5
Carácter actual  e  posicion  6

```

Imagen 57. Problema 35.

36.- Este problema lo resolví importando el alfabeto inglés, para posteriormente, con la función set() comparar la cadena y el alfabeto, el código junto con su salida lo podemos ver en la imagen 58.

```

#Programa 36
print("\nPrograma 36")
from string import ascii_lowercase

cadena="Soy una cadena"
vali=set(cadena) >= set(ascii_lowercase)
print(vali)
cadena="The quick brown fox jumps over the lazy dog"
vali=set(cadena) >= set(ascii_lowercase)
print(vali)

Programa 36
False
True

```

Imagen 58. Problema 36.

37.- Para convertir una cadena en una lista se usa split(), por lo que no hubo mayor complicación, el código junto con su salida lo podemos ver en la imagen 50.

```
#Programa 37
print("\nPrograma 37")
cadena="Soy una cadena y sere lista"
print(cadena.split())

Programa 37
['Soy', 'una', 'cadena', 'y', 'sere', 'lista']
```

Imagen 59. Problema 37.

38.- Aquí se tienen que poner en minúsculas los primeros n caracteres de una cadena, se logra haciendo un indizado y usando la función lower(), no hubo complicaciones, el código junto con su salida lo podemos ver en la imagen 60.

```
#Programa 38
print("\nPrograma 38")
cadena = "ESCUELA SUPERIOR "
n=4
print(cadena[:n].lower()+cadena[n:])

Programa 38
escuELA SUPERIOR
```

Imagen 60. Problema 38.

39.- Este problema me resulto un poco complicado, primero intente usando replace(), pero este intercambia todos los puntos por comas y viceversa, esto no es lo que quería lograr, investigando encontré las funciones maketrans() y translate(), las cuales hacen el intercambio de dos caracteres indicados, el código junto con su salida lo podemos ver en la imagen 61.

```
#Programa 39
print("\nPrograma 39")

cantidad="32.054,23"
maketrans=cantidad.maketrans
cantidad = cantidad.translate(maketrans(','.',' ','.'))
print(cantidad)

Programa 39
32,054.23
```

Imagen 61. Problema 39

40.- Aquí tenemos que contar las vocales en un texto dado, para eso se crea una variable que contenga las vocales y se hace uso de una lista de comprensión para ir buscando las vocales que hay en el texto y al final su longitud serán las vocales encontradas, el código junto con su salida lo podemos ver en la imagen 62.

```

#Programa 40
print("\nPrograma 40")

cadena="hola"
vocales="aeiouAEIOU"
print(len([l for l in cadena if l in vocales]))

Programa 40
2

```

Imagen 62. Problema 40.

41.- Este programa fue sencillo, pues solo fue hacer uso de la función `rsplit()`, el código junto con su salida lo podemos ver en la imagen 63.

```

#Programa 41
print("\nPrograma 41")
cadena="hola*soy*una*cadena"
print(cadena.rsplit('*',2))

Programa 41
['hola*soy', 'una', 'cadena']

```

Imagen 63. Problema 41.

42.- Aquí obtuvimos la última parte de una cadena antes de un carácter especificado, es muy parecido al anterior, el código junto con su salida lo podemos ver en la imagen 64.

```

#Programa 42
print("\nPrograma 42")
cadena="hola soy una cadena*"
print(cadena.rsplit('*',1))

Programa 42
['hola soy una cadena', '']

```

Imagen 64. Problema 42.

43.- Este último programa ya se había resuelto en la práctica 1, el código junto con su salida lo podemos ver en la imagen 65.

```

#Programa 43
print("\nPrograma 43")
cadena="perrera"
prefijos=[]
sufijos=[]
subcadenas=[]

for n in range(len(cadena)+1):
    print("Prefijo ", n, "=", cadena[:n], end=" ") #
    prefijos.append(cadena[:n])
    if cadena[:n] != cadena:
        print("Prefijo propio")

print('\nHay',len(prefijos),' prefijos, ',len(prefijos)-1, ' propios\n')

for n in range(len(cadena)+1):
    if n==0:
        print("Sufijo ", n, "=", sufijo propio", end=" \n")
        sufijos.append('')
    else:
        print("Sufijo ", n, "=", cadena[-n:], end=" ") #
        sufijos.append(cadena[-n:])
        if cadena[-n:] != cadena:
            print("Sufijo propio")
        else:
            print("\n")

print('\nHay',len(sufijos),' sufijos, ',len(prefijos)-1, ' propios\n')

cont=0
for n in range(len(cadena)+1):
    for m in range(n, len(cadena)+1):
        if(cadena[n:m] == "" and n>=1):
            cont=0;
        else:
            print(cadena[n:m])
            subcadenas.append(cadena[n:m])
print('\nHay',len(subcadenas),' subcadenas\n')

```

```

Programa 43
Prefijo 0 = Prefijo propio
Prefijo 1 = p Prefijo propio
Prefijo 2 = pe Prefijo propio
Prefijo 3 = per Prefijo propio
Prefijo 4 = perr Prefijo propio
Prefijo 5 = perre Prefijo propio
Prefijo 6 = perrer Prefijo propio
Prefijo 7 = perrera
Hay 8 prefijos, 7 propios

Sufijo 0 = sufijo propio
Sufijo 1 = a Sufijo propio
Sufijo 2 = ra Sufijo propio
Sufijo 3 = era Sufijo propio
Sufijo 4 = rera Sufijo propio
Sufijo 5 = rrera Sufijo propio
Sufijo 6 = errera Sufijo propio
Sufijo 7 = perrera

Hay 8 sufijos, 7 propios

p
pe
per
perr
perre
perrer
perrera
e
er
err
erre
errer
errera
r
rr
rre
rrer
rrera
r
re
rer
rera
e
er
era
r
ra
a

Hay 29 subcadenas

```

Imagen 65. Problema 43

CONCLUSIONES Y RECOMENDACIONES

En general, esta práctica consistió en el manejo de cadenas en Python a partir de ciertos problemas propuestos por el profesor, en lo personal, lo complicado fue investigar los métodos que tiene Python para las cadenas, ya que muchos de estos problemas eran básicamente usar de forma adecuada estos métodos.

Realice varios experimentos previos a realizar los programas con los métodos que iba encontrando para ver como funcionaban y posteriormente implementarlos al programa correspondiente.

El desarrollo de esta práctica, a mi parecer, fue adecuado, el profesor dio tiempo suficiente y hubo sesiones para dudas en donde fueron aclaradas.

Gracias a esta práctica me llevo un amplio conocimiento de las cadenas en Python y sus métodos, además que adquirí un poco más de práctica en este lenguaje de programación que es nuevo para mí.

BIBLIOGRAFÍA

- González Duque, R. (2014). Python para todos.
- Guzmán Costa, H. (2019). "Métodos de las cadenas". Obtenido de: <https://docs.hektorprofe.net/python/metodos-de-las-colecciones/metodos-de-las-cadenas/>
- Bartolomé Sintés, M. (2020). "Entrada por teclado en Python". Obtenido de: <https://www.mclibre.org/consultar/python/lecciones/python-entrada-teclado.html>
- Información de la página: <https://www.python.org/>
- Covantec, R, L. (2019). "Funciones en Python". Obtenido de: <https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/funciones.html>
- Guzmán Costa, H. (2019). "Comprensión de listas". Obtenido de: <https://docs.hektorprofe.net/python/funcionalidades-avanzadas/compreension-de-listas/>