

Práctica 02 - Trabajo de investigación

Integrantes:

Leonardo Manuel López Benavides

Sebastián Moisés González Nightingale

Ricardo Antonio Manuel Sánchez

Daniel Socas Delgado

1º CFGS Desarrollo de aplicaciones multiplataforma

Programación

Índice

Librería “string”.....	3
Librería “datetime”.....	11
Librería “random”.....	17
Librería “qrcode”.....	21

Librería “string”

La librería string y los métodos integrados de las cadenas de texto en Python sirven para manipular, analizar y transformar texto.

Permiten convertir mayúsculas/minúsculas, buscar, dividir, unir, y formatear cadenas, entre muchas otras funciones.

Para usar estos métodos, no necesitas importar la librería “string” (ya vienen integrados con los objetos tipo “str”), aunque “import string” también está disponible para algunas constantes (como “string.ascii_letters”, etc).

1. upper()

Convierte todos los caracteres de una cadena a mayúsculas.

Parámetros: Ninguno

Retorna: Nueva cadena en mayúsculas

Ejemplo:

```
1 texto = "hola"
2 print(texto.upper()) # "HOLA"
```

2. lower()

Convierte todos los caracteres a minúsculas.

Parámetros: ninguno

Retorna: Nueva cadena con todos los caracteres en minúsculas.

Ejemplo:

```
1 texto = "HoLa"
2 print(texto.lower()) # "hola"
```

3. capitalize()

Convierte la primera letra en mayúscula y el resto en minúsculas.

Parámetros: Ninguno

Retorna: Nueva cadena con la primera letra en mayúscula y el resto en minúsculas.

Ejemplo:

```
1 print("python es genial".capitalize()) # "Python es genial"
```

4. title()

Parámetros: Ninguno

Retorna: Nueva cadena con la primera letra de cada palabra en mayúscula (aplica reglas de título).

Ejemplo:

```
1 print("hola mundo desde python".title()) # "Hola Mundo Desde Python"
```

5. swapcase()

Parámetros: Ninguno

Retorna: Nueva cadena con mayúsculas y minúsculas intercambiadas.

Ejemplo:

```
print("HoLa".swapcase()) # "hOlA"
```

6. find(sub[, start[, end]])

Parámetros:

sub(str): Subcadena a buscar (obligatorio).

start(int, opcional): Índice donde empezar la búsqueda.

end(int, opcional): Índice donde terminar la búsqueda (no incluido).

Retorna: Índice de la primera aparición de sub, o -1 si no se encuentra.

Ejemplo:

```
1 print("banana".find("na")) # 2
```

7. index(sub[, start[, end]])

Parámetros: Iguales a find.

Retorna: Índice de la primera aparición. Lanza ValueError si no se encuentra (en lugar de devolver -1).

Ejemplo:

```
1 print("banana".index("na")) # 2
```

8. count(sub[, start[, end]])

Parámetros:

sub (str): Subcadena a contar.

start, end (int, opcionales): Rango en el que contar.

Retorna: Número de veces que aparece sub en el rango.

Ejemplo:

```
1 print("banana".count("a")) # 3
```

9. startswith(prefix[, start[, end]])

Parámetros:

prefix (str o tupla de str): Prefijo(s) a comprobar.

start, end (int, opcionales): Rango a comprobar.

Retorna: True si la cadena empieza con prefix en el rango.

Ejemplo:

```
1 print("python".startswith("py")) # True
```

10. endswith(suffix[, start[, end]])

Parámetros: Similares a startswith, suffix puede ser str o tupla.

Retorna: True si la cadena termina con suffix en el rango.

Ejemplo:

```
print("python".endswith("on")) # True
```

11.isalpha()

Parámetros: Ninguno

Retorna: True si la cadena no está vacía y todos los caracteres son letras (según Unicode).

Ejemplo:

```
1 print("Hola".isalpha()) # True
```

12. isdigit()

Parámetros: Ninguno

Retorna: True si la cadena no está vacía y todos los caracteres son dígitos (Unicode).

Ejemplo:

```
1 print("1234".isdigit()) # True
```

13. isalnum()

Parámetros: Ninguno

Retorna: True si la cadena no está vacía y todos los caracteres son letras o dígitos.

Ejemplo:

```
1 print("Python3".isalnum()) # True
```

14. isspace()

Parámetros: Ninguno

Retorna: True si la cadena no está vacía y todos los caracteres son espacios en blanco (incluye tab, newline según Unicode).

Ejemplo:

```
1 print(" ".isspace()) # True
```

15. islower()

Parámetros: Ninguno

Retorna: True si hay al menos un carácter cased y todos los caracteres cased están en minúscula.

Ejemplo:

```
1 print("hola".islower()) # True
```

16. isupper()

Parámetros: Ninguno

Retorna: bool — True si hay al menos un carácter cased y todos los caracteres cased están en mayúscula.

Ejemplo:

```
1 print("HOLA".isupper()) # True
```

17. strip([chars])

Parámetros:

chars (str, opcional): Conjunto de caracteres a eliminar del principio y final; si se omite, elimina espacios en blanco.

Retorna: Nueva cadena sin los caracteres especificados al inicio y al final.

Ejemplo:

```
1 print(" hola ".strip()) # "hola"
```

18. lstrip([chars])

Parámetros: Igual que strip, pero actúa solo a la izquierda.

Retorna: Nueva cadena con los caracteres eliminados a la izquierda.

Ejemplo:

```
1 print(" hola".lstrip()) # "hola"
```

19.rstrip([chars])

Parámetros: Igual que strip, pero actúa solo a la derecha.

Retorna: Nueva cadena con los caracteres eliminados a la derecha.

Ejemplo:

```
1 print("hola ".rstrip()) # "hola"
```

20. center(width[, fillchar])

Parámetros:

width (int): Ancho total deseado.

fillchar (str de longitud 1, opcional): Carácter de relleno (por defecto ' ').

Retorna: Nueva cadena centrada en un campo de ancho width.

Ejemplo:

```
1 print("hi".center(6, "-")) # "--hi--"
```

21. ljust(width[, fillchar])

Parámetros: Iguala a center, alinea a la izquierda.

Retorna: Cadena justificada a la izquierda en campo de ancho width.

Ejemplo:

```
1 print("hi".ljust(6, ".")) # "hi...."
```

22. rjust(width[, fillchar])

Parámetros: Iguales a center, alinea a la derecha.

Retorna: Cadena justificada a la derecha en campo de ancho width.

Ejemplo:

```
1 print("hi".rjust(6, ".")) # "...hi"
```

23. zfill(width)

Parámetros:

width (int): Ancho total; rellena con ceros a la izquierda.

Retorna: Nueva cadena rellena con '0' a la izquierda hasta width (respeta signo si existe).

Ejemplo:

```
1 print("42".zfill(5)) # "00042"
```

24. split(sep=None, maxsplit=-1)

Parámetros:

sep (str, opcional): Separador; si es None (por defecto) usa cualquier whitespace y gestiona múltiples separadores consecutivos.

maxsplit (int, opcional): Número máximo de separaciones (valor por defecto -1 = sin límite).

Retorna: Lista de fragmentos.

Ejemplo:

```
1 print("uno dos tres".split()) # ['uno', 'dos', 'tres']
```

25. rsplit(sep=None, maxsplit=-1)

Parámetros: Iguales a split, pero comienza a dividir desde la derecha cuando maxsplit > 0.

Retorna: List de str.

Ejemplo:

```
print("uno-dos-tres".rsplit("-", 1)) # ['uno-dos', 'tres']
```

26. splitlines([keepends])

Parámetros:

keepends (bool, opcional): Si True, mantiene los caracteres de salto de línea al final de cada línea (por defecto False).

Retorna: Lista con las líneas.

Ejemplo:

```
1 print("hola\nmundo".splitlines()) # ['hola', 'mundo']
```

27. join(iterable)

Parámetros:

iterable (iterable de str): Secuencia de cadenas a unir.

Retorna: Nueva cadena resultado de concatenar los elementos de iterable, separados por la cadena sobre la que se llama join.

Ejemplo:

```
1 print("-".join(["a", "b", "c"])) # "a-b-c"
```

28. partition(sep)

Parámetros:

sep (str): Separador (obligatorio).

Retorna: Tuple de tres str: (antes, sep, despues) — divide en la primera aparición de sep. Si sep no existe, devuelve ("", "", original).

Ejemplo:

```
1 print("hola mundo".partition(" ")) # ('hola', ' ', 'mundo')
```

29. rpartition(sep)

Parámetros: Igual que partition.

Retorna: Tuple de tres str: (antes, sep, después) — divide en la última aparición. Si no existe sep, devuelve (original, "", "").

Ejemplo:

```
1 print("uno dos tres".rpartition(" ")) # ('uno dos', ' ', 'tres')
```

30. `replace(old, new[, count])`

Parámetros:

old (str): Subcadena a reemplazar.

new (str): Subcadena reemplazo.

count (int, opcional): Número máximo de reemplazos (por defecto todos).

Retorna: Nueva cadena con los reemplazos aplicados.

Ejemplo:

```
1 print("hola mundo".replace("hola", "adiós")) # "adiós mundo"
```

31. `removeprefix(prefix)` (Python 3.9+)

Parámetros:

prefix (str): Prefijo a eliminar si está presente.

Retorna: Nueva cadena sin el prefijo si estaba; si no, devuelve la cadena original sin cambios.

Ejemplo:

```
1 print("pythonista".removeprefix("py")) # "thonista"
```

32. `removesuffix(suffix)` (Python 3.9+)

Parámetros:

suffix (str): Sufijo a eliminar si está presente.

Retorna: Nueva cadena sin el sufijo si estaba; si no, devuelve la cadena original sin cambios.

Ejemplo:

```
1 print("documento.txt".removesuffix(".txt")) # "documento"
```

Librería “datetime”

La librería datetime de Python se utiliza para trabajar con fechas, horas y períodos de tiempo de una manera sencilla y precisa. Es una de las herramientas más importantes cuando se necesita manejar información temporal, como registrar cuándo ocurrió un evento, calcular diferencias entre fechas o programar tareas en momentos específicos.

Con esta librería, Python permite representar momentos exactos en el tiempo (por ejemplo, “26 de octubre de 2025, 14:35:00”) y realizar operaciones con ellos. Además, facilita la conversión entre distintos formatos de fecha y hora, como transformar una cadena de texto en un objeto de fecha o mostrar una fecha con un formato personalizado

1. now()

Obtiene la fecha y hora actuales del sistema en el momento de la ejecución del programa.

Parámetros: Ninguno (aunque opcionalmente puede recibir tz para especificar una zona horaria)

Retorna: Un objeto datetime que representa la fecha y hora actual.

Ejemplo:

```
1  from datetime import datetime
2
3  actual = datetime.now()
4  print(actual)
5
6  #Hora y fecha del momento que se ejecuta
```

2. strptime()

Convierte una cadena de texto que representa una fecha y hora en un objeto datetime, siguiendo un formato específico.

Parámetros:

- date_string → La cadena de texto que contiene la fecha (por ejemplo, "26/10/2025 14:30").

- **format** → El formato que indica cómo está estructurada la fecha en la cadena (por ejemplo, "%d/%m/%Y %H:%M").

Retorna: Un objeto datetime con la fecha y hora indicadas en la cadena.

Ejemplo:

```
1  from datetime import datetime
2
3  fecha_texto = "26/10/2025 14:30"
4  fecha_objeto = datetime.strptime(fecha_texto, "%d/%m/%Y %H:%M")
5
6  print(fecha_objeto)
7
8  #SALIDA: 2025-10-26 14:30:00
```

3. strftime()

Convierte un objeto datetime en una cadena de texto con un formato específico.

Permite mostrar la fecha y hora en el estilo que desees (por ejemplo, "26/10/2025" o "domingo, 26 de octubre de 2025").

Parámetros: **format** → Una cadena que indica el formato deseado de salida (por ejemplo, "%d/%m/%Y" o "%H:%M:%S").

Retorna: Una cadena de texto que representa la fecha y hora en el formato indicado.

Ejemplo:

```
1  from datetime import datetime
2
3  ahora = datetime.now()
4  formateada = ahora.strftime("%d/%m/%Y %H:%M:%S")
5
6  print(formateada)
7
8  #SALIDA: Hora y fecha quie se ejecuta (de forma exacata)
```

4. timedelta()

Representa un intervalo de tiempo, es decir, una diferencia entre dos fechas u horas. Se utiliza para sumar o restar tiempo a un objeto datetime, o para calcular la duración entre fechas.

Parámetros: **days** → número de días (positivo o negativo)

seconds → número de segundos

microseconds → número de microsegundos

milliseconds → número de milisegundos

minutes → número de minutos

hours → número de horas

weeks → número de semanas

Retorna: Un objeto timedelta que puede sumarse o restarse a objetos datetime.

Ejemplo:

```
1  from datetime import datetime, timedelta
2
3  ahora = datetime.now()
4  intervalo = timedelta(days=6, hours=3)
5
6  futuro = ahora + intervalo
7  print("Fecha y hora actual:", ahora)
8  print("Fecha y hora después de 6 días y 3 horas:", futuro)
9
10 #SALIDA: Dia y hora en este instante y hora y dia 6 dias y 3 horas despues
```

5. date()

Representa sólo la fecha (año, mes y día) sin información de hora.

Parámetros: year → año (entero, por ejemplo, 2025)

month → mes (1–12)

day → día del mes (1–31, según el mes)

Retorna: Un objeto date que representa la fecha indicada.

Ejemplo:

```
1  from datetime import date
2
3  hoy = date(2025, 10, 26)
4  print("Fecha:", hoy)
5
6  #SALIDA: Solo la fecha en el momento de ejecutar el programa
```

6. time()

Representa **solo la hora** (hora, minuto, segundo y microsegundo) sin información de fecha.

Parámetros: hour → hora (0–23, por defecto 0)

minute → minuto (0–59, por defecto 0)

second → segundo (0–59, por defecto 0)

microsecond → microsegundo (0–999999, por defecto 0)

tzinfo → información de zona horaria (por defecto None)

Retorna: Un objeto time que representa la hora indicada.

Ejemplo:

```
1  from datetime import time
2
3  hora_actual = time(14, 30, 15) # 14:30:15
4  print("Hora:", hora_actual)
5
6  #SALIDA: Solo la hora en el momento de ejecutar el programa
```

7. today()

Devuelve la fecha actual del sistema como un objeto date, sin información de hora.

Parámetros: Ninguno

Retorna: Un objeto date con la fecha de hoy.

Ejemplo:

```
1  from datetime import date
2
3  hoy = date.today()
4  print("Fecha de hoy:", hoy)
5
6  #SALIDA: Solo el día del momento que se ejecute el programa
```

8. combine()

Combina un objeto date y un objeto time en un único objeto datetime, que representa fecha y hora completas.

Parámetros:

date → un objeto date que representa la fecha.

time → un objeto time que representa la hora.

Retorna: Un objeto datetime que une la fecha y la hora indicadas.

Ejemplo:

```

1  from datetime import date, time, datetime
2
3  mi_fecha = date(2025, 10, 26)
4  mi_hora = time(14, 30, 0)
5
6  fecha_hora = datetime.combine(mi_fecha, mi_hora)
7  print("Fecha y hora combinadas:", fecha_hora)
8
9  #SALIDA: transforma las fechas y hora puestas en formato hora

```

9. replace()

Crea un nuevo objeto datetime reemplazando uno o más de sus componentes (año, mes, día, hora, minuto, segundo, microsegundo) sin modificar el objeto original.

Parámetros:

year → año (entero)

month → mes (1–12)

day → día del mes (1–31)

hour → hora (0–23)

minute → minuto (0–59)

second → segundo (0–59)

microsecond → microsegundo (0–999999)

tzinfo → zona horaria (opcional)

Retorna: Un nuevo objeto datetime con los valores reemplazados.

Ejemplo:

```

1  from datetime import datetime
2
3  ahora = datetime.now()
4  nuevo = ahora.replace(year=2026, month=1, day=1)
5
6  print("Fecha original:", ahora)
7  print("Fecha modificada:", nuevo)
8
9  #SALIDA: Primero fecha del momento a ejecutar y su hora y luego se reemplaza por la fecha puesta

```

10. weekday()

Devuelve el día de la semana correspondiente a un objeto datetime o date como un número entero.

Parámetros: Ninguno

Retorna: Un entero entre 0 y 6:

0 → lunes

1 → martes

2 → miércoles

3 → jueves

4 → viernes
5 → sábado
6 → domingo

Ejemplo:

```
1  from datetime import datetime
2
3  fecha = datetime(2025, 10, 26)
4  dia_semana = fecha.weekday()
5
6  print("Número del día de la semana:", dia_semana)
7
8  #SALIDA: Aparece el día de la semana que se ejecuta el programa
```

11. date.today()

Devuelve la fecha actual del sistema como un objeto date, sin incluir información de hora.

Parámetros: Ninguno

Retorna: Un objeto date con la fecha de hoy

Ejemplo:

```
1  from datetime import date
2
3  hoy = date.today()
4  print("Fecha de hoy:", hoy)
5
6  #SALIDA: Te da la fecha exacta de hoy
```

Librería Random

Random es un módulo de la biblioteca estándar de Python que te permite generar números y elegir elementos de forma aleatoria. No está pensado para seguridad criptográfica (es **pseudorandom**, reproducible si fijas la semilla), pero es perfecto para juegos, simulaciones, pruebas y tareas donde la seguridad no sea crítica.

1. random.random()

- **Función:** Genera un número decimal (float) aleatorio entre 0.0 y 1.0 (sin incluir 1).
- **Parámetros:** Ninguno
- **Retorno:** un numero float entre 0.0 y 1.0.
- **Ejemplo:** Se genera un número decimal aleatorio entre 0 y 1, sin necesidad de ingresar datos.

```
import random
numero = random.random()
print(numero)
```

2. random.randint(a, b)

- **Función:** Devuelve un **entero aleatorio** entre a y b (ambos incluidos).
- **Parámetros:**
 - a: límite inferior (int)
 - b: límite superior (int)
- **Retorno:** int entre a y b
- **Ejemplo:** Se piden al usuario los límites inferior y superior, se genera un número entero aleatorio dentro de ese rango (incluyendo los límites) y se muestra el resultado.

```
import random

a = int(input("Ingresa el límite inferior: "))
b = int(input("Ingresa el límite superior: "))

numero = random.randint(a, b)
print(f"Número entero aleatorio entre {a} y {b}: {numero}")
```

3. random.uniform(a, b)

- **Función:** Devuelve un número decimal aleatorio entre a y b.
- Parámetros:**
 - a: límite inferior
 - b: límite superior
- **Retorno:** float
- **Ejemplo:** Se solicita al usuario ingresar dos números decimales, y el programa genera un número aleatorio dentro de ese intervalo

```
import random

a = float(input("Ingresa el límite inferior: "))
b = float(input("Ingresa el límite superior: "))

numero = random.uniform(a, b)
print(f"Número decimal aleatorio entre {a} y {b}: {numero}")
```

4. random.randrange(inicio, fin, paso)

- **Función:** Devuelve un número entero aleatorio dentro de un rango, con un paso específico.
- Parámetros:**
 - inicio: valor inicial
 - fin: valor final (no incluido)
 - paso: incremento entre valores posibles
- **Retorno:** int
- **Ejemplo:** Se genera un número entero aleatorio entre 0 y 20, tomando solo los valores que avanzan de 5 en 5

```
import random

numero = random.randrange(0, 20, 5)
print("Número aleatorio entre 0 y 20 (de 5 en 5):", numero)
```

5. random.choice(secuencia)

- **Función:** Devuelve un solo elemento aleatorio de una lista, tupla o cadena.
- **Parámetros:**
 - secuencia: conjunto de elementos (lista, tupla o string)
- **Retorno:** tipo del elemento
- **Ejemplo:** Se tiene una lista con diferentes colores, y el programa elige uno al azar y lo muestra en pantalla.

```
import random

colores = ["rojo", "verde", "azul", "amarillo"]
color = random.choice(colores)
print("Color elegido al azar:", color)
```

6. random.choices(secuencia, weights=None, k=1)

- **Función:** Devuelve una lista de elementos elegidos al azar, con posibilidad de repetición.
- **Parámetros:**
 - secuencia: lista o tupla
 - weights: pesos o probabilidades (opcional)
 - k: cantidad de elementos a elegir
- **Retorno:** list
- **Ejemplo:** Se tiene una lista de frutas, se asigna mayor probabilidad a “banana” y se eligen 5 frutas al azar.

```
import random

frutas = ["manzana", "banana", "cereza"]
eleccion = random.choices(frutas, weights=[1, 3, 1], k=5)
print("Frutas elegidas al azar:", eleccion)
```

7. random.sample(secuencia, k)

- **Función:** Devuelve k elementos distintos elegidos al azar (sin repetir).
- **Parámetros:**
 - secuencia: lista o tupla
 - k: cantidad de elementos
- **Retorno:** list
- **Ejemplo:** Se eligen tres números diferentes de una lista (sin repetir ninguno).

```
import random

numeros = [1, 2, 3, 4, 5, 6]
muestra = random.sample(numeros, 3)
print("Tres números elegidos sin repetir:", muestra)
```

8. random.shuffle(lista)

- **Función:** Mezcla los elementos de una lista, cambiando su orden aleatoriamente.
- **Parámetros:**
 - lista: lista a mezclar
- **Retorno:** ninguno (no devuelve nada)
- **Ejemplo:** Se tiene una lista con las cartas de un mazo y se mezclan aleatoriamente como si se barajaran.

```
import random

numeros = [1, 2, 3, 4, 5, 6]
muestra = random.sample(numeros, 3)
print("Tres números elegidos sin repetir:", muestra)
```

9. random.seed(valor)

- **Función:** Establece una semilla para obtener los mismos resultados aleatorios cada vez.
- **Parámetros:**
 - valor: número o texto que define la semilla
- **Retorno:** ninguno

- **Ejemplo:** Se fija una semilla con el valor 100, lo que hace que cada vez que ejecutes el programa se generen los mismos tres números aleatorios.

```
import random

random.seed(100)

print("Primer número aleatorio:", random.randint(1, 50))
print("Segundo número aleatorio:", random.randint(1, 50))
print("Tercer número aleatorio:", random.randint(1, 50))
```

Librería “Qrcode”

La librería **Qrcode** te permite generar un código QR a partir de una URL. A diferencia del resto de librerías, esta necesita una instalación externa (Biblioteca Pillow) a través de la terminal.

- **Biblioteca Pillow:** Es una biblioteca que agrega la capacidad para poder abrir, manipular y guardar muchos formatos de archivo de imagen diferentes

Instalación:

1. Primero hay que abrir la terminal y escribir el siguiente comando `pip install qrcode`

```
PS C:\DAM> pip install qrcode
```

2. Y ya estaría instalada.

Sus parámetros más importantes son:

Versión:

Controla el tamaño del QR (1 a 40). Cada número permite más datos.

Error_correction:

Define cuánto daño puede tener el código sin perder información es decir cuanto puede estar tapado y seguir siendo escaneable su porcentaje se define con las siguientes letras: **L** (7%), **M** (15%), **Q** (25%), **H** (30%).

box_size:

Tamaño de cada cuadradito del QR en píxeles.

border:

Grosor del borde blanco alrededor del QR (mínimo 4).

fill_color:

Color del código QR.