Desafíos

- Para realizar estos desafíos debes haber estudiado previamente todo el material disponibilizado correspondiente a la unidad.
- Una vez terminado los desafíos, comprime la carpeta que contiene el desarrollo de los requerimientos solicitados y sube el `.zip` en el LMS.
- Desarrollo desafíos:
    - Los desafíos se deben desarrollar de manera Individual.

Instrucciones

A continuación se detallan variados desafíos a desarrollar.

Para su correcta corrección, los programas deben ser almacenados en un comprimido `.zip` de la siguiente manera:

Desafíos.zip
├── iterador.py
├── cuenta_regresiva.py
├── solo_pares.py
├── solo_pares_refactor.py
├── solo_impares.py
├── suma_pares.py
├── genera_patron.py
├── lorem_generator.py
└── fuerza_bruta.py

Importante: Cada vez que se utilice el método **input** debe agregar un salto de línea al final (\n).

Ejemplo:

```
cuenta_regresiva = int(input("Ingrese un número para comenzar la cuenta\n"))
```

Reemplazar while por for

Reemplazar la instrucción `while` por `for` dentro del programa llamado `iterador.py`.

La impresión debe ser la misma:

```
i = 0
while i < 50:
   print("Iteración {}".format(i + 1))
   i +=1
```

Tip: *Cuidado con condición.*

Desafío - Reemplazar instrucción for por while

Reemplaza la instrucción `for` por `while` dentro del programa llamado `cuenta_regresiva.py`.

La impresión debe ser la misma:

```
cuenta_regresiva = int(input("Ingrese un número para comenzar la
cuenta\n"))

for i in range(cuenta_regresiva):
   tmp = cuenta_regresiva
   print("Iteración {}".format(tmp - i))
```

Desafío - Números pares

1. Crear un programa llamado `solo_pares.py`, que muestre todos los números pares hasta "n" (incluyendo "n", si éste es par), donde "n" es un valor ingresado por el usuario.

Uso: `python solo_pares.py`

```
0
2
4
6
8
```

2. Crear una variante del programa anterior llamado `solo_pares_refactor.py`. En este caso, el cero no debe ser considerado (el cero no es par).

Uso: `python solo_pares_refactor.py`

```
2
4
6
8
10
```

Desafío - Números impares

Crear un programa llamado `solo_impares.py`, que muestre todos los números **impares** hasta "n" (incluyendo "n", si éste es impar), donde "n" es un valor ingresado por el usuario.

Tip: *El número siguiente a un par siempre es un impar.*

Uso: `python solo_impares.py`

```
1
3
5
7
9
```

Desafío - Sumar pares

Crear un programa llamado `suma_pares.py` que sume todos los números pares hasta "n" (incluyendo "n" si este es par), donde "n" es ingresado por el usuario.

Tip: *El cero no es par; no afecta en la suma, pero se debe tener cuidado con los bordes del ciclo.*

Uso: `python suma_pares.py`

```
2 + 4 + 6 + 8 + 10 # iteraciones
30 # salida
```

Desafío - Generar patrón

Debe crear un programa que logre replicar el siguiente patrón, donde el usuario ingrese un número, y ese número corresponderá al número de filas que se debe generar. La solución debe estar dentro del programa llamado `genera_patron.py`.

```
1
12
123
1234
12345
```

Considerar que el patrón **debe comenzar por el número 1**, y por ende, el último número de la última fila corresponderá al número ingresado.

Desafío - Generador de Lorem ipsum

Crear un programa llamado `lorem_generator.py`, que sea capaz de mostrar en pantalla varios párrafos de "Lorem ipsum", donde el número de párrafos se especifica al cargar el script.

El texto puede ser extraído del primer párrafo de lipsum.com

Uso: `python lorem_generator.py`

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac
lacinia nibh, nec faucibus enim. Nullam quis lorem posuere, hendrerit
tellus eget, tincidunt ipsum. Nam nulla tortor, elementum in elit nec,
fermentum dignissim sapien. Sed a mattis nisi, sit amet dignissim elit.
Sed finibus eros sit amet ipsum scelerisque interdum. Curabitur justo
nibh, viverra a elit vel, elementum hendrerit erat. Duis feugiat mattis
ante vel hendrerit. Etiam nec nibh nulla. Class aptent taciti sociosqu
ad litora torquent per conubia nostra, per inceptos himenaeos.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac
lacinia nibh, nec faucibus enim. Nullam quis lorem posuere, hendrerit
tellus eget, tincidunt ipsum. Nam nulla tortor, elementum in elit nec,
fermentum dignissim sapien. Sed a mattis nisi, sit amet dignissim elit.
Sed finibus eros sit amet ipsum scelerisque interdum. Curabitur justo
nibh, viverra a elit vel, elementum hendrerit erat. Duis feugiat mattis
ante vel hendrerit. Etiam nec nibh nulla. Class aptent taciti sociosqu
ad litora torquent per conubia nostra, per inceptos himenaeos.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi ac
lacinia nibh, nec faucibus enim. Nullam quis lorem posuere, hendrerit
tellus eget, tincidunt ipsum. Nam nulla tortor, elementum in elit nec,
fermentum dignissim sapien. Sed a mattis nisi, sit amet dignissim elit.
Sed finibus eros sit amet ipsum scelerisque interdum. Curabitur justo
nibh, viverra a elit vel, elementum hendrerit erat. Duis feugiat mattis
ante vel hendrerit. Etiam nec nibh nulla. Class aptent taciti sociosqu
ad litora torquent per conubia nostra, per inceptos himenaeos.
```

Desafío - Fuerza bruta

Se busca crear un programa `fuerza_bruta.py` que revise cuántos intentos se requieren para hackear un password por fuerza bruta.

Uso: `python fuerza_bruta.py`

```
Ingresa contraseña
gato
43 intentos
```

Para ello, el programa debe intentar todas las combinaciones de letras posibles, en orden alfabético, hasta que la combinación de letras sea igual a la de la contraseña indicada. Deberá hacer este proceso letra por letra, de izquierda a derecha.

Es decir:

- Primero probará con a, luego b, luego c ... hasta z, o **hasta que encuentre** una letra igual a la primera letra de la contraseña.
- Suponiendo que la primera letra correspondía a "d", después empezará a comparar la segunda letra de la forma da, db, dc, dd... hasta encontrar la coincidencia de la segunda letra.
- Suponiendo que la segunda letra era "e", continuará luego comparando la tercera letra de la forma dea, deb, dec ...etc.
- Y así sucesivamente hasta completar la comparación con cada letra de la contraseña.

Considerar:

- Se asume que el programa sólo tiene letras, las cuales se pueden repetir.
- Considera mayúsculas y minúsculas como una misma letra.
- Se considera "**intento**" cada vez que se compara una letra.

  Ejemplo:
  - Usuario ingresa "abc"
  - El computador compara:
    - **a** es igual a **a** => Sí (1 intento), continúa con la siguiente letra.
    - **b** es igual a **a** => No (2 intentos), prueba otra comparación.
    - **b** es igual a **b** => Sí (3 intentos), continúa con la siguiente letra.
    - **c** es igual a **a** => No (4 intentos), prueba con otra comparación.
    - **c** es igual a **b** => No (5 intentos), prueba con otra comparación.
    - **c** es igual a **c** => Sí (6 intentos), continúa con la siguiente letra.
    - No hay más letras. Se adivinó la palabra en 6 intentos.

Debes saber:

Para este ejercicio, se debe recordar que un ciclo `for` recorrerá los elementos de una estructura **iterable**. En la lectura, se vieron ejemplos recorriendo rangos, por lo que los elementos recorridos siempre fueron números. Pero también es posible recorrer **strings**, donde cada elemento iterado (dado por el iterador del ciclo, normalmente `i`) corresponderá a una letra de la palabra o texto contenedor (se va recorriendo de una en una letra, de izquierda a derecha).

Se sugiere investigar el módulo `string` para ver cómo recorrer el abecedario.

La solución debe estar dentro del programa `fuerza_bruta.py`.

Tips:

- *Partir con intento = 'a'.*
- *Si sospecha que su programa se ha "colgado" porque ha entrado en un "loop infinito", puede detener su ejecución con la combinación de letras `ctrl + c` en la terminal.*