

## 4. Bucles e Iterables

---

### 4.1. Iterables

- 4.1.1. Métodos Básicos

- 4.1.2. Índices

### 4.2. Estructuras de Repetición

- 4.2.1. Bloque for

  - 4.2.1.1. Recorrer un Iterable

  - 4.2.1.2. Recorrido con Índices

  - 4.2.1.3. Contador Condicional

  - 4.2.1.4. Acumulador

- 4.2.2. Bloque while

  - 4.2.2.1. Contadores no Lúneales

  - 4.2.2.2. Ejecucion por Evento

  - 4.2.2.3. Bucles Infinitos

- 4.2.3. Controladores de Bucles

---

### 4.1. Iterables

---

Un iterable es un objeto que es capaz de retornar sus elementos uno a la vez, es un tipo de dato cuyo manejo es especialmente útil y poderoso en Python.

En general, los iterables son aquellos tipos de datos que pueden contener varios elementos, y se llaman así porque Python incorpora funciones que permiten ejecutar operaciones sobre cada uno de los sus elementos con mucha facilidad, por ejemplo, se puede retornar o modificar cada uno de sus elementos con un [ciclo for](#) como veremos más adelante.

Los iterables que mas se van a usar en la guía son las variables tipo [string \*str\*](#), y las [listas \*list\*](#) que se verán en el siguiente capítulo.

## 4.1.1. Métodos Básicos

A continuación están los métodos básicos para interactuar con cualquier iterable, `len()`, `count()`, e `index()`, en este caso se aplicarán a cadenas.

### a. Método `len()`

Este método nos muestra la longitud del iterable, es decir, la cantidad de elementos que tiene.

```
# Mini_Ejemplo: Método len()

## Ingresos de datos:
cadena=input("Por favor, ingrese cualquier cadena")

## Contar los elementos:
cantidad=len(cadena)

## Imprimir resultado:
print("La cantidad de elementos que tiene la cadena ingresada '",cadena,"' es:",cantidad)
```

```
# Pase aquí el mini ejemplo anterior
```

### b. Método `.count()`

Este método nos muestra la frecuencia de un elemento en un iterable, es decir, la cantidad de veces que se encuentra un elemento en el iterable.

```
# Min_Ejemplo: Método count()

## Ingreso de datos:
palabra="interdisciplinarietà"
print("La palabra disponible para hacer la consulta es: '",palabra,"'.\n")
letra=input("Por favor, ingrese en minúscula la letra que a la quiera consultar su frecuencia.\n")

## Pequeña validación de datos:
if ((letra not in palabra) and (len(letra)==1)):
    import random
    letra=random.choice(palabra)
    print("Lo sentimos, no ha ingresado un valor inválido, el programa usará:",letra,"\n")

## Evaluación del índice:
veces=palabra.count(letra)
```

```
## Impresión de resultados:  
print("La cantidad de veces que esta la letra '", letra, "' en la palabra '", palabra, "' es:", vec
```

```
# Pase aquí el mini ejemplo anterior
```

### c. Método `.index()`

Con este método se puede solicitar la posición de un elemento en un iterable, si hay elementos repetidos arrojará el índice del que se encuentra más a la izquierda.

```
# Mini_Ejemplo: Método index  
  
## Ingreso de datos:  
palabra="centrifugados"  
print("La palabra disponible para hacer la consulta es: '", palabra, "'.\n")  
letra=input("Por favor, ingrese en minúscula la letra a la cual quiera consultarle su posición.  
  
## Pequeña validación de datos:  
if ((letra not in palabra) and (len(letra)==1)):  
    import random  
    letra=random.choice(palabra)  
    print("Lo sentimos, no ha ingresado un valor inválido, el programa usará:", letra, "\n")  
  
## Evaluación del índice:  
indice=palabra.index(letra)  
  
## Impresión de resultados:  
print("La posición de la letra '", letra, "' es:", indice)
```

```
# Pase aquí el mini ejemplo anterior
```

## 4.1.2. Índices

- **Retornar Valores:** Es posible acceder a cualquier elemento de un iterable si conocemos su posición o índice, es vital recordar que la indexación en Python **empieza en cero**, esto quiere decir que el primer elemento tiene índice cero.

```
# Mini_Ejemplo: Índices

## Ingreso de datos:
palabra=input("Por favor, ingrese una palabra de más de tres letras:\n")

## Pequeña validación de datos:
if not((palabra.isalpha()) and (len(palabra)>3)):
    palabra= 'electroencefalografista'
    print("Lo sentimos, no ha ingresado una palabra válida, el programa usará:\n",palabra,"\n")

## Asignación de resultados:
resultado_1=palabra[0]
resultado_2=palabra[1]
resultado_3=palabra[2]

## Impresión de resultados:
print("El primer elemento del iterable es: '",resultado_1,'" y su índice es 0.\n")
print("El segundo elemento del iterable es: '",resultado_2,'" y su índice es 1.\n")
print("El tercer elemento del iterable es: '",resultado_3,'" y su índice es 2.\n")
```

# Pase aquí el mini ejemplo anterior

- **Índices negativos:** Las posiciones en un iterable también pueden ser representadas con número negativos, como si se cambiara de punto de referencia al último elemento y las posiciones se contáran hacia atrás, esta indexación **empieza en -1**.

```
# Mini_Ejemplo: Índices negativos

## Ingreso de datos:
palabra=input("Por favor, ingrese una palabra de más de tres letras:\n")

## Pequeña validación de datos:
if not((palabra.isalpha()) and (len(palabra)>3)):
    palabra= 'electroencefalografista'
    print("Lo sentimos, no ha ingresado una palabra válida, el programa usará:",palabra,"\n")

## Asignación de resultados:
resultado_1=palabra[-1]
resultado_2=palabra[-2]
resultado_3=palabra[-3]

## Impresión de resultados:
print("El último elemento del iterable es: '",resultado_1,'" y su índice es -1.\n")
```

```
print("El penúltimo elemento del iterable es: '",resultado_2,'" y su índice es -2.\n")
print("El antepenúltimo elemento del iterable es: '",resultado_3,'" y su índice es -3.\n")
```

# Pase aquí el mini ejemplo anterior

- **Secciones:** También se puede acceder a través de los índices a secciones de un iterable, para esto se deben indicar la posición inicial y final+1 de la sección.

# Mini\_Ejemplo: Secciones

## Ingreso de datos:

```
palabra=input("Por favor, ingrese una palabra de más de seis letras:\n")
```

## Pequeña validación de datos:

```
if not((palabra.isalpha()) and (len(palabra)>3)):
```

```
    palabra= 'electroencefalografista'
```

```
    print("Lo sentimos, no ha ingresado una palabra válida, el programa usará:",palabra,"\n")
```

## Asignación de resultados:

```
resultado_1=palabra[0:4]
```

```
resultado_2=palabra[:4]
```

```
resultado_3=palabra[:-3]
```

```
resultado_4=palabra[2:5]
```

```
resultado_5=palabra[1:]
```

```
resultado_6=palabra[-5:]
```

## Impresión de resultados:

```
print("Los primeros cuatro elementos del iterable son: '",resultado_1,'" es decir, los elementos
```

```
print("Lo anterior también se puede decir: Los elementos que están antes del quinto elemento son: '",resultado_2,'" es decir, los elementos
```

```
print("Los elementos que están antes del antepenúltimo elemento son: '",resultado_3,'" es decir, los elementos que están antes del tercer elemento
```

```
print("Los elementos que están desde el tercer elemento hasta el quinto son: '", resultado_4,'" es decir, los elementos que están desde el tercer elemento hasta el quinto elemento
```

```
print("Los elementos que están después del segundo elemento son: '", resultado_5,'" es decir, los elementos que están después del segundo elemento
```

```
print("Los últimos 5 elementos son: '", resultado_6,'" es decir, los elementos que están desde el quinto elemento hasta el final del iterable")
```

# Pase aquí el mini ejemplo anterior

- **Error de Indexación:** Si se intenta acceder a una posición que no existe al correr el código arrojará.

```
# Mini_Ejemplo: Error de Indexación

iterable="01234567"
indice=iterable[8]

">>> IndexError: string index out of range"
```

## 4.2. Estructuras de Repetición

Las estructuras de repetición son un tipo de estructura de control, usada para repetir un conjunto de instrucciones durante una cantidad de ciclos determinada o hasta que una condición no se cumpla, normalmente son llamados bloques de repetición, bucles o ciclos.

El Bloque for y el Bloque while son las principales estructuras de repetición, y muchas veces pueden ser usadas cualquiera de las dos para crear un ciclo en específico, sin embargo, hay varias diferencias entre ellos.

Ojo: Al igual que con las estructuras de decisión, la indentación es crucial para que el código corra correctamente.

### 4.2.1. Bloque for

La ejecución de este ciclo siempre va a depender de un iterable, hay diferentes maneras en que se usa un ciclo for, a continuación, se mostrarán algunas de ellas.

#### 4.2.1.1 Recorrer un Iterable

El bloque for se puede usar para recorrer e interactuar con los elementos de un iterable, normalmente se lee: *por cada elemento en un iterable realice las siguientes instrucciones:*.

```
# Mini_Ejemplo: Recorrer un iterable
iterable="cadena"

for elemento in iterable:
    "Aquí van las instrucciones a realizar"
    print(elemento.upper())

print("Aquí ya salimos del bloque")
```

```
# Pase aquí el mini ejemplo anterior
```

## 4.2.1.2 Recorrido con Índices

Es muy útil cuando se necesita interactuar con dos iterables a la vez, es muy importante que estos sean del mismo tamaño, porque se puede tener un ciclo incompleto o que al correrlo arroje [index error](#).

Para esto se debe usar la función `range(k)`

- **Función range:** Los parámetros de esta función son (start, stop, step), el tercero es opcional, por ejemplo si las entradas son (j,k,n), retornará una lista inmutable de números desde (j) hasta (k-1), saltando de (n) en (n). Si sólo se ingresa un valor la función tomará las entradas por defecto como (0,k)

```
# Mini_Ejemplo Función range
```

```
print("Rango a: 1,11,2")
```

```
for i in range(1,11,2):  
    print(i)
```

```
print("Rango b: 1,11")
```

```
for i in range(1,11):  
    print(i)
```

```
print("Rango c: 11")
```

```
for i in range(11):  
    print(i)
```

```
print("Rango d: -2,2")
```

```
for i in range (-2,2):  
    print(i)
```

```
# Pase aquí el mini ejemplo anterior
```

En el siguiente se verá cómo crear una cadena, fusionando dos cadenas predefinidas intercalando sus elementos.

```
# Mini_Ejemplo: Recorrido con Índices
```

```
consonantes="TYT" #Iterable 1
```

```

vocales="OOA" #Iterable 2
resultado="" # Se crea una cadena vacía para acumular resultados internos.
rango_indices=range(len(consonantes)) # Se crea el rango para los índices = 0 hasta (tamaño de v

for i in rango_indices:
    print("Esta es la vez No.",i+1,"en la que entramos al ciclo, el índice actual es:",i,"\n")
    resultado_interno=consonantes[i]+vocales[i] # Concatenar los elementos de mismo índice de v
    print("Resultado interno=",resultado_interno)
    resultado=resultado+resultado_interno # Acumular los resultados internos
    print("Resultado acumulado=",resultado,"\n\n")

print("El resultado final=",resultado)

```

# Pase aquí el mini ejemplo anterior

### 4.2.1.3 Contador Condicional

Es usado cuando se necesita contar la cantidad de elementos de un iterable que cumplen con una condición.

Como el contador es una variable acumulativa, será de gran ayuda conocer los operadores de asignación.

- **\*\*Operadores de asignación:** \*\* Estos operadores son muy útiles para abreviar operaciones sobre la misma variable.

| Operador | Descripción  | Operación | Descripción    |
|----------|--------------|-----------|----------------|
| =        | Igualdad     | /=        | c/=3(c=c/3)    |
| +=       | c+=1 (c=c+1) | %=        | c%=4 (c=c%4)   |
| -=       | c-=1 (c=c-1) | **=       | c**=2(c=c**2)  |
| *=       | c*=2 (c=c*2) | //=       | c//=5 (c=c//5) |

El siguiente Mini\_ejemplo cuenta cuantos números primos hay entre 1 y 100. Si desea imprimir la lista de los números identificados, como primos o no, deshabilite los comentarios.

```

# Mini_Ejemplo: Contador Condicional (Números primos hasta el 100)

rango=range(1,101) # Se crea el rango de 1 hasta 100

```



```

contador=0

for i in rango:
    if i!=1 and ((i in (2,3,5,7))or not((i%2==0) or (i%3==0) or (i%5==0) or (i%7==0))):
#         print(i,"es primo")
        contador+=1 #contador=contador+1
#     else:
#         print(i,"no es primo")

print("\nLa cantidad de números primos que hay hasta 100 es:",contador)

```

```
# Pase aquí el mini ejemplo anterior
```

#### 4.2.1.4 Acumulador

Este ciclo tambien se puede utilizar para acumular en una variable los resultados de cada ciclo, sobretodo cuando se necesita interactuar con los resultados de ciclos anteriores y se debe crear una variable que haga el papel de memoria.

En el siguiente ejemplo se va a guardar en un string la secesión de Fibonacci hasta el n-avo valor, el cual es ingresado por el usuario.

La suseción de fibonacci  $(x_n)$  está descrita por la siguiente formula:  $x_1=1, \quad x_2=1, \quad x_n=x_{n-1}+x_{n-2}; (n>2)$ , es decir,  $(x_n)$  es igual, a la suma de los dos  $(x_n)$  anteriores, por lo tanto la lógica que siguió fue la siguiente:

Sea:

nuevo\_fibonacci=  $(x_n)$

fibonacci\_actual=  $x_{n-1}$

fibonacci anterior=  $x_{n-2}$

Crear una lista de 1 hasta el n-avo Fibonacci.

Para cada n evaluar:

Si n es 1:

El fibonacci actual es 0

Y el anterior es 1.

Si es diferentes de 1 entonces:

El nuevo fibonacci es el actual + el anterior

(Después de calculado el nuevo fibonacci)

El fibonacci actual se convierte en el anterior.  
Y el nuevo fibonacci se convierte en el actual.

```
# Mini_Ejemplo: Acumulador (Sucesión de Fibonacci)

#Ingresar datos
n_avo=input("Por favor ingrese el n-avo Fibonacci hasta donde quiere calcular la sucesión\n")
sucesion=""

## Pequeña validación de datos:
if not n_avo.isnumeric():
    n_avo=100
    print("Ha ingresado un valor inválido, el programa ha definido que el n-avo Fibonacci será:",

## Conversión:
n_avo=int(n_avo)

## Crear rango:
rango=range(1,n_avo+1) #creamos el rango para que empiece en 1 y termine en n.

for i in rango:
    # Si es el Fibonacci no. 1 entonces:
    if i==1:
        #El fibonacci anterior es igual 0 y el actual es igual 1
        fibonacci_anterior=0
        fibonacci=1
    else: #si no
        #El nuevo fibonacci es igual al actual mas el anterior.
        nuevo_fibonacci=fibonacci+fibonacci_anterior
        fibonacci_anterior=fibonacci
        fibonacci=nuevo_fibonacci
    final=",\n" if i!=n_avo else "."
    sucesion=sucesion+"Fibonacci No."+str(i)+"= "+str(fibonacci)+final

print("La sucesión Fibonacci hasta el índice 100 es:")
print(sucesion)
```

# Pase aquí el mini ejemplo anterior

## 4.2.2. Bloque while

Este bloque ejecuta las instrucciones que contiene mientras se cumpla la condición a la que esté sujeto, cuando esta condición deja de ser verdadera se pasa al siguiente bloque del código.

Como se verá más adelante este bloque es muy útil cuando no se tiene predefinida la cantidad de iteraciones a realizar, y normalmente se lee: *Mientras se cumpla la condición x haga:*

A continuación, están algunas formas de usar este bloque.

#### 4.2.2.1. Contadores no Lineales.

En el siguiente ejemplo, el usuario tiene 5 intentos para adivinar un número, así que puede haber hasta 5 interacciones, pero no se sabe exactamente cuántas, ya que, el usuario puede adivinar en la oportunidad 1,2,3 o 4, por lo tanto, lo más común es usar un bloque while, ya que, para hacerlo con un bloque for habría que usar el [controlador break](#) que se explicará más adelante.

```
# Mini_Ejemplo: Contadores no lineales

## Ingreso datos:
import random
numero=str(random.randint(1,10))
intentos=1
print("Adivina el número entre 1 y 10, cuenta con 5 intentos")

while intentos<=5:
    num_usuario=input("Ingresa un número.")

    if num_usuario==numero:
        print("!Has adivinado, felicitaciones!")
        intentos=5
    elif intentos==4:
        print("Fallaste, ya solo te queda una oportunidad.")
    elif intentos==5:
        print("Fallaste de nuevo, el número era",numero)
    else:
        print("El número no es",num_usuario,"vuelve a intentarlo.")

    intentos+=1

print("Terminamos de jugar, !gracias!")
```

```
# Pase aquí el mini ejemplo anterior
```

#### 4.2.2.2 Ejecución por Evento

Cuando se necesita que se cumpla cierta condición para poder continuar, se crea un ciclo while, como muestra el siguiente mini\_ejemplo no se puede seguir hasta que no se ingrese un número entero.

```
# Mini_Ejemplo: Ejecución por Evento (Pequeña validación de datos)

numero=input("Por favor, ingrese un número entero:")

while not(numero.isnumeric()):
    numero=input("No ha ingresado un número entero, por favor vuelva a intentarlo")

print("Gracias por ingresar un número válido")
```


```
# Pase aquí el mini ejemplo anterior
```

### 4.2.2.3 Bucles Infinitos

Como se vió el ciclo while a diferencia del ciclo for no requiere tener definida la cantidad de iteraciones, y se puede repetir un bloque hasta o mientras que se cumpla una condición como vimos en ejemplo anterior, por lo cual, es posible tener un ciclo infinito si esta condición nunca se llega a cumplir, por lo tanto, el programa nunca dejará de ejecutarse.

Por ejemplo si ejecuta el siguiente Mini\_Ejemplo en una consola está no dejará de ejecutarse hasta que la reinicie con ( ctrl + L + C ) o se acabe la memoria.

```
# Mini_Ejemplo: Bucle infinito (Factorial infinito)
numero=1
factorial=1
while numero>=1:
    factorial*=numero
    print(factorial, "\n")
    numero+=1
```

 Fig1\_Bucle\_Infinito

### 4.2.3. Controladores de Bucles

Las palabras reservadas `Break` y `Continue`, son instrucciones que modifican el flujo de una estructura de repetición.

- **Break:** Se utiliza para salir de un bucle si se cumple una condición.

- **Continue:** Con esta instrucción se puede saltar una parte del bucle si se cumple una condición.

En la guía no se va hacer mucho uso de ellos, sin embargo, el siguiente ejemplo muestra cómo hacer el [Mini\\_Ejemplo Contadores no Lineales](#), con un bloque for haciendo uso de la instrucción break.

```
# Mini_Ejemplo: Break (Adivinar un número con for)

## Ingreso datos:
import random
numero=str(random.randint(1,10))
numero=str(4)
print("Adivina el número entre 1 y 10, cuenta con 5 intentos")

for i in range(5):
    num_usuario=input("Ingresa un número")
    if num_usuario==numero:
        print("!Has adivinado, felicitaciones!")
        break
    elif i==3:
        print("Fallaste, ya solo te queda una oportunidad.")
    elif i==4:
        print("Fallaste de nuevo, el número era",numero)
    else:
        print("El número no es",num_usuario,"vuelve a intentarlo.")

print("Terminamos de jugar, !gracias!")
```

```
# Pase aquí el mini ejemplo anterior
```

[Anterior](#)

-

[Siguiente](#)[Home](#)