LISTAS

Estructuras de Datos

01 {

[Continuación]

- Caracteres ASCII. Caracteres Unicode
- Break
- Continue
- Ciclo While True
- Ciclo While False
- Ejercicio 9.1

}

Caracteres ASCII. Caracteres Unicode {

Los **caracteres ASCII y Unicode** son dos **estándares de codificación de caracteres que se utilizan para representar caracteres** en la informática y en otros campos relacionados con la tecnología.

Mientras que el estándar **ASCII se limita a los caracteres en inglés** y algunos caracteres especiales, **Unicode es capaz de representar caracteres de múltiples idiomas** y sistemas de escritura.

Por lo tanto, Unicode se ha convertido en el estándar de codificación de caracteres más utilizado en la actualidad.

}

Caracteres ASCII. Caracteres Unicode {

Para **insertar un carácter ASCII**, **mantenga presionada la tecla ALT mientras escribe el código de carácter**.

Por ejemplo, para insertar el símbolo de grado (°), mantenga presionada la tecla ALT mientras escribe 0176 en el teclado numérico.

Debe usar el teclado numérico para escribir los números y no el teclado estándar. Asegúrese de que la tecla BLOQ NÚM está activada si su teclado lo requiere para que escriba números en el teclado numérico.

}

Caracteres ASCII. Caracteres Unicode {

Para **insertar un carácter Unicode**, **escriba el código de carácter, presione ALT y después presione X**.

Por ejemplo, para escribir un símbolo de dólar ($), escriba 0024, presione ALT y después presione X. Es posible que deba asegurarse de que el teclado esté configurado para admitir la entrada de caracteres Unicode.

También se puede utilizar un método diferente para insertar el carácter Unicode, como la **combinación de teclas "Win + ." (punto) en Windows**, que abrirá la ventana de emoji y símbolos donde puede buscar y seleccionar el carácter Unicode deseado.

}

Break {

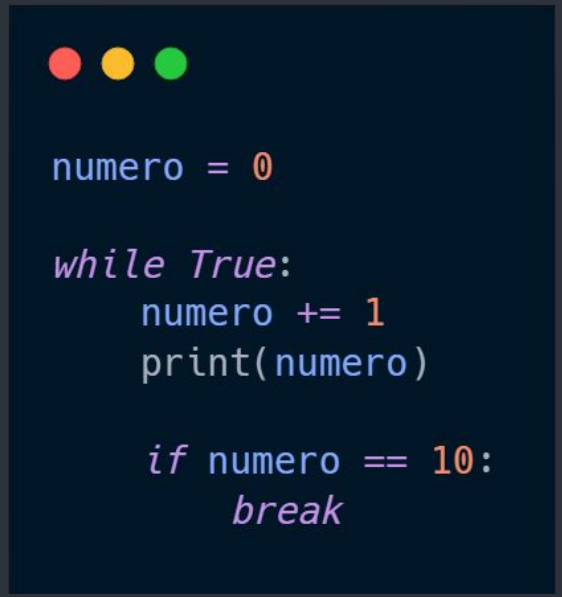
break es una instrucción en Python que **se utiliza para interrumpir la ejecución de un bucle for o while antes de que se complete**.

Cuando se ejecuta la instrucción break, el flujo de control del programa salta inmediatamente fuera del bucle y continúa con la siguiente línea de código después del bucle.

La instrucción break es útil cuando se desea detener un bucle antes de que se complete, por ejemplo, cuando se ha alcanzado una condición específica o cuando se desea salir del bucle en respuesta a una entrada del usuario.

}

Break {



```python
numero = 0

while True:
    numero += 1
    print(numero)

    if numero == 10:
        break
```

}

Continue {

continue es una instrucción en Python que **se utiliza para saltar a la siguiente iteración de un bucle for o while, omitiendo cualquier código restante en la iteración actual**.

Cuando se ejecuta la instrucción continue, el flujo de control del programa salta inmediatamente al final del bucle y continúa con la siguiente iteración.

La instrucción continue **es útil cuando se desea omitir una iteración específica del bucle y continuar con la siguiente**.

Por ejemplo, se podría utilizar para saltarse una iteración cuando se ha encontrado un valor no válido en una lista o para omitir un cálculo costoso que no es necesario en una iteración específica.

}

Continue {

```python
numero = 0

while numero < 10:
    numero += 1
    if numero % 2 == 0:
        # Saltar a la siguiente iteración si el número es par
        continue

    print(numero)
```

}

Ciclo While True {

El **ciclo while True** es una estructura de ciclo en Python que **crea un bucle infinito.**

El bucle se ejecutará continuamente mientras la expresión booleana **True** sea verdadera, lo que significa que el bucle continuará ejecutándose indefinidamente a menos que se interrumpa o se detenga manualmente.

}

Ciclo While True {

Es importante tener en cuenta que los bucles while True **pueden ser peligrosos** si no se controlan adecuadamente, ya que **pueden agotar los recursos del sistema** y provocar bloqueos o congelamientos.

Por lo tanto, es importante asegurarse de tener una forma de interrumpir o detener el bucle en caso de que sea necesario.

}

Ciclo While True {

```python
while True:
    entrada = input("Ingrese un número o escriba 'salir' para terminar: ")

    if entrada == "salir":
        break

    numero = int(entrada)
    resultado = numero ** 2

    print(f"El cuadrado de {numero} es {resultado}")
```

}

Ejercicio 9 {

Valide que la **edad** sea una **cadena numérica**, y que sea un **número entre 0 y 100**. Debe utilizar **.isnumeric()**.

}

Ejercicio 9.1 {

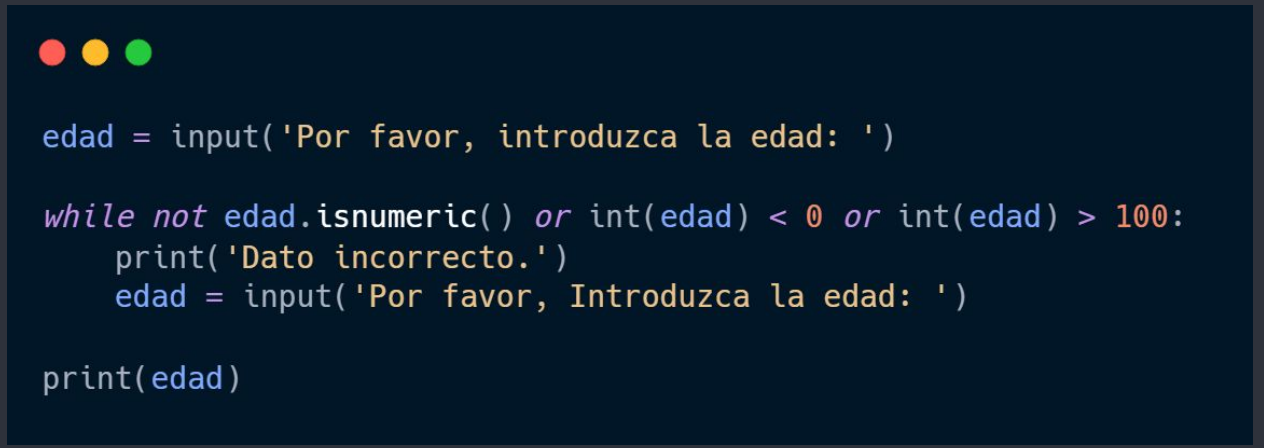
```python
edad = input("Ingrese la edad: ")
ciclo = True

while ciclo:
    while not edad.isnumeric():
        edad = input("Ingrese la edad: ")
    if(edad.isnumeric()):
        if int(edad) >= 1 and int(edad) <= 100:
            ciclo = False
        else:
            edad = ""

print(edad)
```

}

Ejercicio 9.2 {



```python
edad = input('Por favor, introduzca la edad: ')

while not edad.isnumeric() or int(edad) < 0 or int(edad) > 100:
    print('Dato incorrecto.')
    edad = input('Por favor, Introduzca la edad: ')

print(edad)
```

}

02 {

[Listas]

- Listas
- Características
- Creacion
- Indices
- Indices negativos
- Modificar elementos
- Métodos de listas
- Operadores de pertenencia

}

Listas {

En Python, una lista es una **estructura de datos** y **un tipo de dato** que se utiliza para **almacenar una colección de elementos**, como números, cadenas de texto, booleanos, otras listas, y objetos de cualquier tipo.

Las listas son **mutables**, lo que significa que se pueden modificar después de su creación, y se pueden acceder a sus elementos por índice.

}

Listas {

Las listas **se definen utilizando corchetes []** y los elementos se separan por **comas**.

Por ejemplo, aquí hay una lista que contiene algunos números enteros:

```
numeros = [1, 2, 3, 4, 5]
#Pos      0  1  2  3  4
```

}

Características de las listas {

1. **Ordenadas:** Mantienen el orden en el que se insertó la data.
2. **Mutables:** Se pueden modificar los items que tiene guardados.
3. **Heterogeneas:** Pueden guardar múltiples tipos de datos al mismo tiempo.
4. **Contienen duplicados:** Permite tener objetos duplicados.

}

Creación de listas {

Para crear una lista en Python, se utiliza la sintaxis de **corchetes []** y se separan los elementos por **comas**.

Por ejemplo, podemos tener una lista con elementos que tienen tipos de datos distintos:

```
lista_de_elementos = [1, 2, "tres", True, 4.5, [5, 7]]
#Pos                  0  1    2      3     4     5
```

}

Indices {

Los índices se utilizan para **acceder a elementos específicos de una lista**.

Los índices son **números enteros que indican la posición de un elemento** en la lista.

En Python, los índices **comienzan en 0**, lo que significa que el primer elemento de la lista tiene un índice de 0, el segundo elemento tiene un índice de 1, y así sucesivamente.

}

Indices {

Para acceder a un elemento de la lista mediante índices, debemos primero colocar el **nombre de la lista**, seguido de **corchetes[]** y dentro de este colocaremos la **posición deseada**.

}

Indices {

Podemos acceder al **primer elemento** de la lista utilizando el **índice 0**:

```
numeros = [10, 20, 30, 40, 50]
primer_elemento = numeros[0]
print(primer_elemento)  # Imprime: 10
```

}

Indices {

Podemos acceder al **tercer elemento** de la lista utilizando el **índice 2**:

```
numeros = [10, 20, 30, 40, 50]
tercer_elemento = numeros[2]
print(tercer_elemento)  # Imprime: 30
```

}

Indices negativos {

También se puede acceder a los elementos de la lista utilizando **índices negativos**, que cuentan desde el final de la lista hacia el principio.

Por ejemplo, **el último elemento de la lista tiene un índice de -1**, el segundo último tiene un índice de -2, y así sucesivamente.

}

Indices negativos {

```python
numeros = [10, 20, 30, 40, 50]
#          -5  -4  -3  -2  -1

ultimo_elemento = numeros[-1]
print(ultimo_elemento)  # Imprime: 50

penultimo_elemento = numeros[-2]
print(penultimo_elemento)  # Imprime: 40
```

}

Indices {

n = tamaño de la lista

n - 1 = indice mas alto

```
numeros = [1, 2, 3]
```

Indices negativos

[1, 2, 3]

-3 -2 -1

Indices positivos

[1, 2, 3]

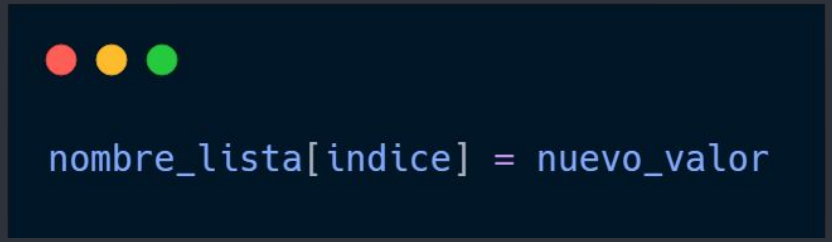
0 1 2

}

Modificar elementos {

En Python, los elementos de una lista se pueden modificar después de su creación, ya que las listas son **estructuras de datos mutables**.

Para modificar un elemento de una lista, se utiliza el **índice del elemento** y se asigna un **nuevo valor** a esa posición de la lista.



```
nombre_lista[indice] = nuevo_valor
```

}

Modificar elementos {

```
lista = [1, 2, 3, 4, 5]
lista[2] = "tres"
# [1, 2, "tres", 4, 5]
```

}

Modificar elementos {

También se pueden **modificar varios elementos de una lista a la vez** utilizando la sintaxis de **segmento de lista (`[inicio:fin]`)** y la asignación a un nuevo segmento de lista.

La sintaxis básica para modificar varios elementos de una lista a la vez es la siguiente:

```
lista[inicio:fin] = mini_lista_cambios
```

}

Modificar elementos {

[inicio,fin)
intervalo

[inicio:fin]

Se debe tener en cuenta que, se modificará desde la **posición inicio** hasta la posición que esté antes de la **posición fin**.

```
lista = [1, 2, 3, 4, 5]
#        0  1  2  3  4

lista[1:4] = ["dos", "tres", "cuatro"]

# [1, 'dos', 'tres', 'cuatro', 5]
#  0    1       2        3     4
```

}

Modificar elementos {

```
lista = [1, 2, 3, 4, 5]
#        0  1  2  3  4

lista[1:4] = ["dos", "tres", "cuatro"]

# [1, 'dos', 'tres', 'cuatro', 5]
#  0    1       2        3     4
```

}

Funcion len() {

La función len() se utiliza para obtener **la longitud de una lista**, es decir, el **número de elementos** que contiene la lista.

```python
frutas = ["manzana", "banana", "naranja", "piña"]

# Obtener la longitud de la lista de frutas
longitud = len(frutas) # 4 elementos

# Imprimir la longitud
print("La lista de frutas tiene", longitud, "elementos.")
```

}

Métodos de listas {

En Python, las listas son una estructura de datos muy **flexible** y tienen una gran cantidad de métodos integrados para realizar operaciones.

}

Método .append() {

Agrega un elemento al final de la lista.

```
lista = [1, 2, 3, 4, 5]

nuevo_numero = 6

lista.append(nuevo_numero)

# [1, 2, 3, 4, 5, 6]
```

}

Método .pop() {

Este método permite **eliminar y retornar un elemento en la última posición** si se deja su parámetro vacío, o **eliminar un elemento en una posición específica** colocando el índice del elemento como argumento.

```python
numeros = [1, 2, 3, 4, 5]

# Eliminar el último elemento de la lista y asignarlo a una variable
ultimo_numero = numeros.pop()

# Imprimir la lista original y el último número eliminado
print("Lista original:", numeros) # [1, 2, 3, 4]
print("Último número eliminado:", ultimo_numero) # Eliminado: 5
```

}

Método .pop() {

Este método permite **eliminar y retornar un elemento en la última posición** si se deja su parámetro vacío, o **eliminar un elemento en una posición específica** colocando el índice del elemento como argumento.

```python
frutas = ["manzana", "banana", "naranja", "piña"]

# Eliminar la segunda fruta de la lista y asignarla a una variable
segunda_fruta = frutas.pop(1)

# Imprimir la lista original y la segunda fruta eliminada
print("Lista original:", frutas) # ['manzana', 'naranja', 'piña']
print("Segunda fruta eliminada:", segunda_fruta) # Eliminada: banana
```

}

Métodos de listas {

- **.extend():** Agrega elementos de otra lista (o cualquier iterable) al final de la lista actual.
- **.insert():** Inserta un elemento en una posición específica de la lista.
- **.remove():** Elimina el primer elemento de la lista que coincide con el valor dado.
- **.index():** Devuelve el índice de la primera aparición de un elemento en la lista.

}

Métodos de listas {

- **.count():** Devuelve el número de veces que un elemento aparece en la lista.
- **.sort():** Ordena los elementos de la lista en orden ascendente.
- **.reverse():** Invierte el orden de los elementos de la lista.
- **.copy():** Devuelve una copia superficial (shallow copy) de la lista.

}

Operadores de pertenencia {

in y **not in** son operadores de pertenencia que se utilizan en Python para verificar **si un elemento está o no en una lista** (u otra estructura de datos iterable, como una tupla o un conjunto).

}

Operadores de pertenencia {

Operador in

El **operador in** devuelve **True** si el elemento está en la lista, y **False** en caso contrario.

```
(variable in nombre_lista)
```

Operador not in

El **operador in** devuelve **True** si el elemento no está en la lista, y **False** en caso contrario.

```
(variable not in nombre_lista)
```

}

Operador in {

```python
# Verificar si un elemento está en la lista
frutas = ["manzana", "banana", "naranja", "piña"]
print("banana" in frutas) # True
```

}

Operador in {

```python
# Verificar si un elemento está en la lista
frutas = ["manzana", "banana", "naranja", "piña"]
if "banana" in frutas: # True
    print("Sí, la banana está en la lista de frutas.")
else:
    print("No, la banana no está en la lista de frutas.")
```

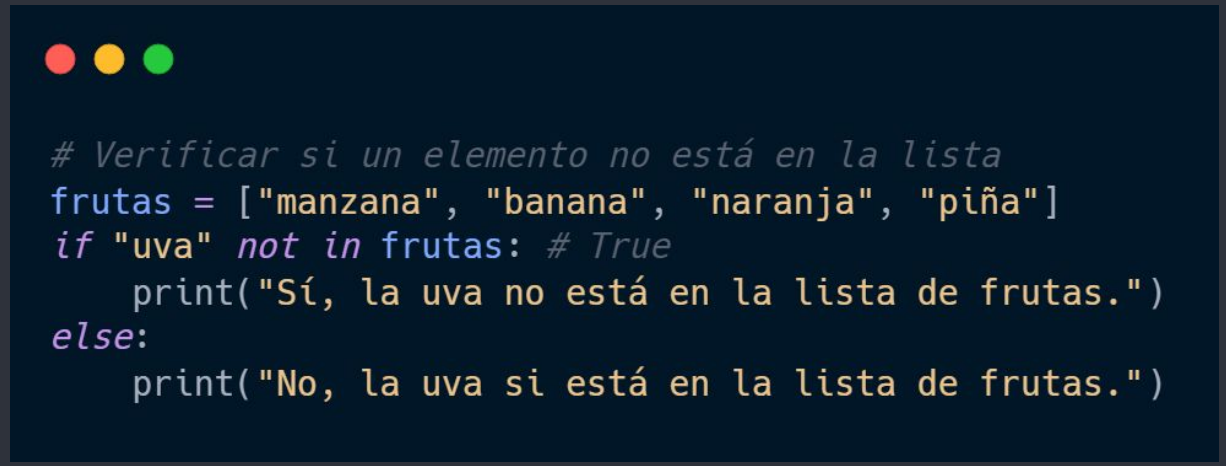
}

Operador not in {

```python
# Verificar si un elemento no está en la lista
frutas = ["manzana", "banana", "naranja", "piña"]
print("uva" not in frutas) # True
```

}

Operador not in {



```python
# Verificar si un elemento no está en la lista
frutas = ["manzana", "banana", "naranja", "piña"]
if "uva" not in frutas: # True
    print("Sí, la uva no está en la lista de frutas.")
else:
    print("No, la uva si está en la lista de frutas.")
```

}

Ejercicio 10 {

Realice un programa que dada una **lista de nombres**, pida al usuario **introducir un nombre** y **verificar si este está o no en la lista**, en caso de que no esté, el programa debe preguntarle al usuario si desea introducir el nombre a la lista, se debe imprimir la lista al final.

nombres = [“Andres”, “Yoselyn”, “Gabriela”, “Juan”, “Marco”, “Veronica”]

}

Ejercicio 10 {

```
}
```

```python
nombres = ["Andres", "Yoselyn", "Gabriela", "Juan", "Marco", "Veronica"]

nombre_a_buscar = input("Ingrese el nombre que quiere buscar: ")
if(nombre_a_buscar in nombres):
    print(f"El nombre {nombre_a_buscar} fue encontrado.")
else:
    print("El nombre no existe.")
    opcion = input("¿Desea agregar el nombre a la lista? Si(1), No(2)")
    opciones_disponibles = ["1", "2"]
    while opcion not in opciones_disponibles:
        opcion = input("¿Desea agregar el nombre a la lista? Si(1), No(2)")
    if opcion == "1":
        nombres.append(nombre_a_buscar)
        print(nombres)
    else:
        print("Gracias por usar el programa.")
```

ESTRUCTURAS REPETITIVAS

El uso de ciclos en Python

02 {

[Estructuras repetitivas]

- Ciclos
- Ciclo For
- Advertencia
- Ciclo For con Strings
- Funcion range

}

Ciclos {

En programación, los ciclos (también llamados bucles) son estructuras de control que **permiten ejecutar repetidamente un bloque de código mientras se cumple una condición determinada**.

En Python, existen dos tipos de ciclos: el **ciclo "for"** y el **ciclo "while"**.

El número de **iteraciones** de una estructura repetitiva puede ser definido de una de dos maneras:

- Por condición
- Por iterador

}

Ciclo For {

El **ciclo for** es una estructura de control de flujo en Python que se utiliza para **iterar sobre un conjunto de elementos (iterable)**.

Un iterable es **cualquier dato (simple o complejo) que puede recorrerse**, hasta ahora solo hemos visto dos datos que cumplen esta condición: **listas** y **strings**.

La sintaxis básica de un **bucle for** en Python es la siguiente:

```
for elemento in iterable:
    # código que se ejecuta para cada elemento del iterable
```

}

Ciclo For {

En cada iteración del bucle, se ejecuta el código dentro del bloque de código indentado, y la **variable elemento** toma el valor del siguiente elemento del iterable. El bucle continúa hasta que se han procesado todos los elementos del iterable.

```python
numeros = [1, 2, 3, 4, 5]
for numero in numeros:
    print(numero)
```

}

Advertencia {

Resulta importante tomar en cuenta que **las asignaciones realizadas sobre la variable de iteración NO afectarán a los valores originales dentro del iterable.**

A continuación podemos ver un ejemplo de esto.

}

Advertencia {

```
numeros = [1, 2, 3, 4, 5]
for numero in numeros:
    numero *= 100
    print(numero)
print(numeros)
```

}

Ciclo For con Strings {

En Python, también se puede utilizar un **bucle for** para **iterar sobre los caracteres de una cadena de texto** (string).

```python
cadena = "PYTHON"

for letra in cadena:
    print(letra)
```

}

Ejercicio 11 {

Se tiene una **lista de varios estudiante**, donde cada elemento es una cadena que contiene su nombre y apellido, separado por un espacio.

Recorra la lista, y genere una nueva lista con nombres de usuario para estos estudiantes, reemplace los espacios por una guión bajo “_” y ponga las palabras en minúscula.

Finalmente imprima la lista de nombres de usuario.

}

Ejercicio 11 {

}

```python
estudiantes = [
    'Luis Rojas',
    'Andres Betancourt',
    'Andres Rodriguez',
    'Marco Aurelio',
    'Samuel Vera',
    'Alberto Carrillo',
    'Javier Medina',
    'Veronica Paez',
]
usernames = []
for estudiante in estudiantes:
    # Colocamos todos los caracteres en minuscula
    username = estudiante.lower()
    # Quitamos los espacios en blanco al inicio y al final de la cadena
    username = username.strip()
    # Reemplazamos los espacios por pisos
    username = username.replace(" ", "_")
    # Agregamos a la nueva lista
    usernames.append(username)

print(usernames)
```

Funcion range() {

Existen otros constructos lógicos denominados **funciones generadoras** que también pueden ser iteradas por esta estructura de control (ciclo for).

range() es una función incorporada de Python que se utiliza para **generar una secuencia de números enteros.**

La sintaxis básica de la función **range()** es la siguiente:

```
range(inicio, fin, incremento)
```

}

Funcion range() {

range(<START>,<STOP>,<STEP>)

- **Start (opcional):** Se refiere al valor de inicio de nuestra colección temporal de datos, el valor por default es 0.
- **Stop (obligatorio):** Se refiere al valor anterior al final de nuestra colección temporal de datos.
- **Step (opcional):** Se refiere al incremento o decremento que se dará sobre cada valor sucesivo para llegar desde el Start hasta el Stop. El valor por defecto es 1.

}

Funcion range() {

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

```
1
2
3
4
5
6
7
8
9
10
```

}

Funcion range() {

```python
for i in range(5,11):
    print(i)
```

```
5
6
7
8
9
10
```

}

Funcion range() {

```python
for i in range(0,11,2):
    print(i)
```

```
0
2
4
6
8
10
```

}

Funcion range() {

```python
for i in range(10,0,-1):
    print(i)
```

```
10
9
8
7
6
5
4
3
2
1
```

}

Ejercicio 12 {

Solicitar al usuario un número entero desde el cual se desea contar y otro número entero hasta el cual se desee contar.

Utilizando un **bucle for** y la **función range**, mostrar en un String concatenado todos los números generados en el rango definido por el usuario.

}

Ejercicio 12 {

```python
# Solicitar al usuario el número de inicio y el número de fin
inicio = int(input("Ingrese el número de inicio: "))
fin = int(input("Ingrese el número de fin: "))

# Inicializar una variable para almacenar la cadena concatenada
cadena = ""

# Utilizar un bucle for y la función range para generar los números en el rango definido
for numero in range(inicio, fin + 1):
    # Concatenar cada número a la cadena
    cadena += str(numero) + " "

# Mostrar la cadena concatenada en la consola
print("Los números generados son:", cadena)
```

}

Ejercicio 13 {

Solicitar al usuario un número entero positivo.

Utilizando un **bucle for** calcular el **factorial del número** proporcionado por el usuario.

}

Ejercicio 13 {

```python
# Solicitar al usuario un número entero positivo
numero = int(input("Ingrese un número entero positivo: "))

# Inicializar una variable para almacenar el factorial
factorial = 1

# Utilizar un bucle for para calcular el factorial del número ingresado
for i in range(1, numero + 1):
    factorial *= i

# Mostrar el factorial del número en la consola
print("El factorial de", numero, "es", factorial)
```

}