* **Introducción a Python**: Python es un lenguaje de programación versátil y de código abierto creado por Guido Van Rossum. Es ampliamente utilizado para diversas aplicaciones, incluyendo ciencia de datos.
* **Python Interactivo (IPython)**: Has experimentado con IPython, una versión mejorada del shell estándar de Python, útil para obtener retroalimentación inmediata y experimentación.
* **Scripts de Python**: Has aprendido que los scripts de Python son archivos de texto con una extensión .py. Permiten ejecutar múltiples comandos de Python en secuencia.
* **Cálculos Básicos**: Has practicado usando Python para operaciones aritméticas básicas:
* print(4 + 5) *# Suma*
* print(5 - 5) *# Resta*
* print(3 \* 5) *# Multiplicación*
* print(10 / 2) *# División*
* **Comentarios en Python**: Has descubierto cómo añadir comentarios a tu código usando el símbolo # para hacerlo más legible.

Además de los tipos de datos numéricos, hay otros tres tipos de datos muy comunes:

* float o punto flotante: un número que tiene una parte entera y otra fraccionaria, separadas por un punto. 1.1 es un ejemplo de flotante.
* str o cadena: un tipo para representar texto. Puedes utilizar comillas simples o dobles para construir una cadena.
* bool o booleano: un tipo para representar valores lógicos. Solo puede ser True o False (¡las mayúsculas son importantes!).

# Operaciones con otros tipos

En Python existen distintos tipos de variables. Puedes ver el tipo de una variable utilizando type(). Por ejemplo, para ver el tipo de a, ejecuta lo siguiente: type(a).

Los distintos tipos se comportan de forma diferente en Python. Cuando sumes dos cadenas, por ejemplo, obtendrás un comportamiento diferente que cuando sumes dos enteros o dos booleanos

nombre = input("¿Cuál es tu nombre?")

edad= input("¿Cuántos años tienes?")

print (f"Hola {nombre}, tienes {edad} años.")

la f se usa para mezclar texto con variables.

**CONDICIONALES**

edad= int(input("Cuál es tu edad?"))

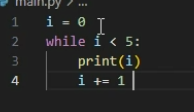
if edad >= 18:

    print("Eres mayor de edad")

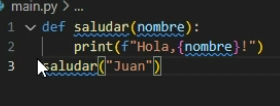
else:

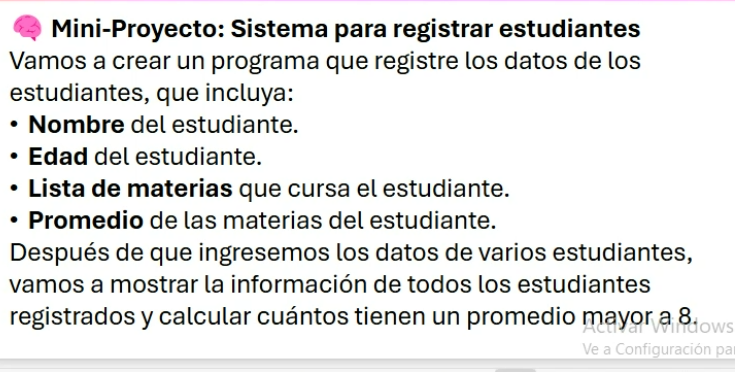
    print("Eres menor de edad")

**BUCLES**

 CONTADOR

**FUNCIONES**





Claro, aquí tienes un código en Python que te permite:

1. Ingresar estudiantes con su nombre, edad y una lista de materias con sus notas.esto que quede en un diccionario
2. Calcular el promedio de cada estudiante.
3. Mostrar todos los estudiantes con sus materias y notas.
4. Crear una lista de estudiantes con promedio mayor a 8.

## 

Vamos a mejorar el programa con estas dos funciones extra:

1. **Editar estudiantes ya ingresados** (cambiar nombre, edad, o actualizar materias y notas).
2. **Guardar y cargar datos desde un archivo JSON** para conservar la información entre ejecuciones.





## 🔹 **Funciones y Métodos para Listas (**list**)**

Las listas en Python son estructuras mutables y ordenadas.

### 📌 ****Funciones globales para listas****

* len(lista): Devuelve el número de elementos en la lista.
* sum(lista): Retorna la suma de todos los elementos numéricos de la lista.
* max(lista): Devuelve el elemento más grande de la lista.
* min(lista): Devuelve el elemento más pequeño de la lista.
* sorted(lista): Retorna una nueva lista ordenada sin modificar la original.
* list(iterable): Convierte un iterable en una lista.

### 📌 ****Métodos de listas****

* lista.append(elemento): Agrega un elemento al final de la lista.
* lista.extend(otra\_lista): Agrega los elementos de otra lista al final.
* lista.insert(posición, elemento): Inserta un elemento en una posición específica.
* lista.remove(elemento): Elimina la primera aparición del elemento en la lista.
* lista.pop([índice]): Elimina y devuelve el elemento en la posición indicada (por defecto, el último).
* lista.clear(): Elimina todos los elementos de la lista.
* lista.index(elemento): Devuelve el índice de la primera aparición del elemento.
* lista.count(elemento): Devuelve cuántas veces aparece un elemento en la lista.
* lista.sort([reverse=False]): Ordena la lista en su lugar.
* lista.reverse(): Invierte el orden de los elementos en la lista.
* lista.copy(): Devuelve una copia de la lista.

**🔹 Métodos de listas en Python**

1. **append()**– Agrega un elemento al final de la lista

pitón

CopiarEditar

lista = [1, 2, 3]

lista.append(4)

print(lista) # [1, 2, 3, 4]

1. **extend()**– Agrega múltiples elementos al final

pitón

CopiarEditar

lista = [1, 2, 3]

lista.extend([4, 5, 6])

print(lista) # [1, 2, 3, 4, 5, 6]

1. **insert()**– Inserte un elemento en una posición específica

pitón

CopiarEditar

lista = [1, 2, 4]

lista.insert(2, 3) # Inserta el número 3 en la posición 2

print(lista) # [1, 2, 3, 4]

1. **remove()**– Eliminar el primer elemento con un valor específico

pitón

CopiarEditar

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

lista.remove(3)

print(lista) # [1, 2, 4, 3]

1. **pop()**– Elimina y devuelve un elemento de la lista (por defecto el último)

pitón

CopiarEditar

lista = [1, 2, 3, 4]

elemento = lista.pop(2) # Elimina el elemento en la posición 2

print(lista) # [1, 2, 4]

print(elemento) # 3

1. **index()**– Retorna el índice de la primera ocurrencia de un valor

pitón

CopiarEditar

lista = [10, 20, 30, 40]

print(lista.index(30)) # 2

1. **count()**– Cuenta cuantas veces aparece un elemento en la lista

pitón

CopiarEditar

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

print(lista.count(1)) # 3

1. **sort()**– Ordena la lista en orden ascendente (o descendente con reverse=True)

pitón

CopiarEditar

lista = [5, 2, 9, 1]

lista.sort()

print(lista) # [1, 2, 5, 9]

lista.sort(reverse=True)

print(lista) # [9, 5, 2, 1]

1. **reverse()**– Invierte el orden de los elementos de la lista.

pitón

CopiarEditar

lista = [1, 2, 3, 4]

lista.reverse()

print(lista) # [4, 3, 2, 1]

1. **copy()**– Crea una copia de la lista

pitón

CopiarEditar

lista = [1, 2, 3]

copia = lista.copy()

print(copia) # [1, 2, 3]

1. **clear()**– Elimina todos los elementos de la lista.

pitón

CopiarEditar

lista = [1, 2, 3]

lista.clear()

print(lista) # []

**🔹 Funciones útiles para listas en Python**

1. **len()**– Obtiene la cantidad de elementos

pitón

CopiarEditar

lista = [1, 2, 3, 4]

print(len(lista)) # 4

1. **sum()**– Calcula la suma de los elementos.

pitón

CopiarEditar

lista = [1, 2, 3, 4]

print(sum(lista)) # 10

1. **max()**– Obtén el valor máximo

pitón

CopiarEditar

lista = [10, 20, 5, 40]

print(max(lista)) # 40

1. **min()**– Obtenga el valor mínimo

pitón

CopiarEditar

lista = [10, 20, 5, 40]

print(min(lista)) # 5

1. **sorted()**– Retorna una nueva lista ordenada

pitón

CopiarEditar

lista = [3, 1, 4, 2]

print(sorted(lista)) # [1, 2, 3, 4]

print(sorted(lista, reverse=True)) # [4, 3, 2, 1]

1. **enumerate()**– Permite recorrer la lista obteniendo índices y valores

pitón

CopiarEditar

lista = ['a', 'b', 'c']

for indice, valor in enumerate(lista):

print(f"Índice: {indice}, Valor: {valor}")

**Salida:**

archivo make

CopiarEditar

Índice: 0, Valor: a

Índice: 1, Valor: b

Índice: 2, Valor: c

1. **zip()**– Combina varias listas en pares

pitón

CopiarEditar

nombres = ["Ana", "Luis", "Sofía"]

edades = [25, 30, 22]

combinados = list(zip(nombres, edades))

print(combinados) # [('Ana', 25), ('Luis', 30), ('Sofía', 22)]

1. **map()**– Aplicar una función a cada elemento

pitón

CopiarEditar

numeros = [1, 2, 3, 4]

cuadrados = list(map(lambda x: x\*\*2, numeros))

print(cuadrados) # [1, 4, 9, 16]

1. **filter()**– Filtrar elementos según una condición

pitón

CopiarEditar

numeros = [10, 20, 25, 30, 35]

mayores\_que\_20 = list(filter(lambda x: x > 20, numeros))

print(mayores\_que\_20) # [25, 30, 35]

## 🔹 **Funciones y Métodos para Diccionarios (**dict**)**

Los diccionarios son estructuras de clave-valor

### 📌 iloc en Python (Pandas)

El método **.iloc** pertenece a la librería **pandas** y se utiliza para la indexación y selección de datos en un **DataFrame** o una **Serie** mediante índices de posición (números enteros).

📌 **Características principales de .iloc:**  
✔ Permite acceder a filas y columnas por índice de posición (no por etiquetas).  
✔ Usa indexación basada en enteros.  
✔ Acepta números enteros, listas de índices, slices (:) y arrays booleanos.

## 🔹 **Ejemplos de Uso**

### 🟢 ****1. Acceder a una fila específica por índice****

python

CopiarEditar

import pandas as pd

# Crear un DataFrame de ejemplo

data = {'Nombre': ['Ana', 'Juan', 'Luis', 'María'],

'Edad': [25, 30, 35, 28],

'Ciudad': ['Madrid', 'Barcelona', 'Sevilla', 'Bilbao']}

df = pd.DataFrame(data)

# Obtener la primera fila

print(df.iloc[0])

🔹 **Salida:**

vbnet

CopiarEditar

Nombre Ana

Edad 25

Ciudad Madrid

Name: 0, dtype: object

### 🟢 ****2. Acceder a un rango de filas****

python

CopiarEditar

# Obtener las primeras dos filas

print(df.iloc[0:2])

🔹 **Salida:**

nginx

CopiarEditar

Nombre Edad Ciudad

0 Ana 25 Madrid

1 Juan 30 Barcelona

### 🟢 ****3. Acceder a una columna específica****

python

CopiarEditar

# Obtener la segunda columna (Edad)

print(df.iloc[:, 1])

🔹 **Salida:**

yaml

CopiarEditar

0 25

1 30

2 35

3 28

Name: Edad, dtype: int64

📌 **Explicación:**

* : → Selecciona todas las filas.
* 1 → Selecciona la columna en la posición 1 (Edad).

### 🟢 ****4. Acceder a un elemento específico (fila, columna)****

python

CopiarEditar

# Obtener la edad de la tercera persona (índice 2, columna 1)

print(df.iloc[2, 1])

🔹 **Salida:**

CopiarEditar

35

### 🟢 ****5. Seleccionar múltiples filas y columnas****

python

CopiarEditar

# Obtener filas 1 y 3, y columnas 0 y 2

print(df.iloc[[1, 3], [0, 2]])

🔹 **Salida:**

css

CopiarEditar

Nombre Ciudad

1 Juan Barcelona

3 María Bilbao

### 🟢 ****6. Usar condiciones booleanas con**** .iloc

python

CopiarEditar

# Filtrar filas donde la edad sea mayor a 28

print(df.iloc[(df['Edad'] > 28).values])

🔹 **Salida:**

nginx

CopiarEditar

Nombre Edad Ciudad

1 Juan 30 Barcelona

2 Luis 35 Sevilla

## 🔹 **Diferencia entre** .iloc **y** .loc

| **Método** | **Selección por...** | **Ejemplo** |
| --- | --- | --- |
| .iloc | Índice de posición (entero) | df.iloc[0:2, 1] |
| .loc | Etiqueta de fila/columna | df.loc[0:2, 'Edad'] |

### 🏆 ****Conclusión****

📌 .iloc es muy útil para seleccionar datos de un DataFrame basándose en posiciones numéricas, permitiendo una manipulación eficiente de datos en Python con pandas.

# **Funciones y Métodos para Trabajar con Diccionarios en Python (**dict**)**

Los **diccionarios** en Python son estructuras de datos que almacenan pares clave-valor. Son mutables y permiten acceso rápido a los valores a través de sus claves.

## 🔹 **Funciones Globales para Diccionarios**

Estas funciones operan sobre diccionarios sin ser métodos específicos.

### 1️⃣ len(diccionario)

* 📌 Devuelve el número de elementos en el diccionario.
* 🔹 **Ejemplo:**

python

CopiarEditar

dic = {'a': 10, 'b': 20, 'c': 30}

print(len(dic)) # Salida: 3

### 2️⃣ dict(iterable)

* 📌 Crea un diccionario a partir de una lista de tuplas clave-valor.
* 🔹 **Ejemplo:**

python

CopiarEditar

lista = [('nombre', 'Ana'), ('edad', 25)]

dic = dict(lista)

print(dic) # Salida: {'nombre': 'Ana', 'edad': 25}

### 3️⃣ max(diccionario) y min(diccionario)

* 📌 Devuelven la clave máxima y mínima del diccionario (según orden alfabético o numérico de las claves).
* 🔹 **Ejemplo:**

python

CopiarEditar

dic = {'a': 10, 'c': 30, 'b': 20}

print(max(dic)) # Salida: 'c'

print(min(dic)) # Salida: 'a'

## 🔹 **Métodos de Diccionarios**

Los métodos de diccionarios permiten manipular y obtener información de ellos.

### 1️⃣ .keys()

* 📌 Devuelve una vista de todas las claves.
* 🔹 **Ejemplo:**

python

CopiarEditar

dic = {'a': 10, 'b': 20, 'c': 30}

print(dic.keys()) # Salida: dict\_keys(['a', 'b', 'c'])

### 2️⃣ .values()

* 📌 Devuelve una vista de todos los valores.
* 🔹 **Ejemplo:**

python

CopiarEditar

print(dic.values()) # Salida: dict\_values([10, 20, 30])

### 3️⃣ .items()

* 📌 Devuelve una vista de los pares clave-valor en tuplas.
* 🔹 **Ejemplo:**

python

CopiarEditar

print(dic.items()) # Salida: dict\_items([('a', 10), ('b', 20), ('c', 30)])

### 4️⃣ .get(clave, valor\_por\_defecto)

* 📌 Obtiene un valor dado su clave. Si no existe, devuelve el valor por defecto.
* 🔹 **Ejemplo:**

python

CopiarEditar

print(dic.get('b')) # Salida: 20

print(dic.get('d', 'No existe')) # Salida: 'No existe'

### 5️⃣ .update(otro\_diccionario)

* 📌 Fusiona otro diccionario dentro del actual, sobrescribiendo claves repetidas.
* 🔹 **Ejemplo:**

python

CopiarEditar

dic.update({'b': 50, 'd': 40})

print(dic) # Salida: {'a': 10, 'b': 50, 'c': 30, 'd': 40}

### 6️⃣ .pop(clave, valor\_por\_defecto)

* 📌 Elimina y devuelve el valor asociado a una clave.
* 🔹 **Ejemplo:**

python

CopiarEditar

valor = dic.pop('b')

print(valor) # Salida: 50

print(dic) # Salida: {'a': 10, 'c': 30, 'd': 40}

### 7️⃣ .popitem()

* 📌 Elimina y devuelve un par clave-valor al azar (en Python 3.7+ elimina el último).
* 🔹 **Ejemplo:**

python

CopiarEditar

par = dic.popitem()

print(par) # Salida: ('d', 40)

print(dic) # Salida: {'a': 10, 'c': 30}

### 8️⃣ .setdefault(clave, valor\_por\_defecto)

* 📌 Si la clave existe, devuelve su valor. Si no, la agrega con un valor por defecto.
* 🔹 **Ejemplo:**

python

CopiarEditar

print(dic.setdefault('c', 100)) # Salida: 30 (ya existía)

print(dic.setdefault('e', 50)) # Salida: 50 (clave nueva añadida)

print(dic) # Salida: {'a': 10, 'c': 30, 'e': 50}

### 9️⃣ .clear()

* 📌 Elimina todos los elementos del diccionario.
* 🔹 **Ejemplo:**

python

CopiarEditar

dic.clear()

print(dic) # Salida: {}

## 🔹 **Conclusión**

📌 Python ofrece múltiples métodos y funciones para manipular **diccionarios**, desde acceder a claves y valores hasta modificarlos de forma eficiente.

### 1. ****Listas**** ( list)

La última lista

#### **Funciones y métodos principales**

* append(elemento):
* insert(pos, elemento): Insertar
* remove(elemento):Eliminar
* pop(pos):Elpos.
* sort():Orden
* reverse():

#### **Ejemplo**

pag

CopiarEditar

numeros = [3, 1, 4, 1, 5]

numeros.append(9) # Agrega 9 al final

numeros.sort() # Ordena la lista

print(numeros) # Salida: [1, 1, 3, 4, 5, 9]

### 2. ****Conjuntos****(set)

Los

#### **Funciones y métodos principales**

* add(elemento):Ag
* remove(elemento):Eli
* discard(elemento):Eliminar
* union(otro\_set):Desaparece
* intersection(otro\_set):Devu
* difference(otro\_set):Desarrollador

#### **Ejemplo**

pito

CopiarEditar

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

interseccion = set1.intersection(set2)

print(interseccion) # Salida: {3, 4}

### 3. ****Tuplas****(tuple)

Último

#### **Funciones y métodos principales**

* count(elemento):
* index(elemento):Devu

#### **Ejemplo**

pitón

CopiarEditar

tupla = (10, 20, 30, 40, 10)

print(tupla.count(10)) # Salida: 2

print(tupla.index(30)) # Salida: 2

### 4. ****Diccionarios****(dicdict)

yo

#### **Funciones y métodos principales**

* keys():
* values():De
* items():
* get(clave, valor\_predeterminado):Obtener
* update(otro\_dict):Estoy de acuerdo
* pop(clave):Eli

#### **Ejemplo**

pitón

CopiarEditar

datos = {"nombre": "Ana", "edad": 25, "ciudad": "Madrid"}

print(datos.get("edad")) # Salida: 25

datos.update({"profesion": "Ingeniera"}) # Agrega una nueva clave-valor

print(datos) # Salida: {'nombre': 'Ana', 'edad': 25, 'ciudad': 'Madrid', 'profesion': 'Ingeniera'}

Estas son las estructuras de datos básicos en Python junto con sus métodos y funciones más utilizadas. ¿Te gustaría profundizar en alguna de ellas? 🚀

4o

##### Dijiste:

Me gustaria profundizar en las funciones y métodos para conjuntos y para tuplas.

###### ChatGPT dijo:

¡Claro! Vamos a profundizar en **conjuntos ( set)** y**tuplas ( tuple)**, ex

## **1. Conjuntos (** set**) en Python**

Los conjuntos son colecciones **desordenadas** ,**mutables** (**no permiten elementos duplicados** .

### ****Funciones y métodos en conjuntos****

### 🔹 ****Añadir y eliminar elementos****

* add(elemento):Ag
* remove(elemento):Eliminar**genera error** si no ex
* discard(elemento):El**Sin generar error**si no existe
* pop():**ALEATORIO** DEL
* clear():Va

#### **Ejemplo**

pitón

CopiarEditar

conjunto = {1, 2, 3}

conjunto.add(4) # Agrega el número 4

conjunto.discard(2) # Elimina el número 2

print(conjunto) # Salida: {1, 3, 4}

### 🔹 ****Operaciones entre conjuntos****

* union(otro\_set): Re
* intersection(otro\_set):Retornador
* difference(otro\_set):
* symmetric\_difference(otro\_set): Retirarse

#### **Ejemplo**

pag

CopiarEditar

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

print(set1.union(set2)) # Salida: {1, 2, 3, 4, 5, 6}

print(set1.intersection(set2)) # Salida: {3, 4}

print(set1.difference(set2)) # Salida: {1, 2}

print(set1.symmetric\_difference(set2)) # Salida: {1, 2, 5, 6}

### 🔹 ****Subconjuntos y Superconjuntos****

* issubset(otro\_set):DescubrimientoTruesi
* issuperset(otro\_set):Truesi y

#### **Ejemplo**

pag

CopiarEditar

a = {1, 2, 3}

b = {1, 2, 3, 4, 5}

print(a.issubset(b)) # Salida: True

print(b.issuperset(a)) # Salida: True

## **2. Tuplas (** tuple**) en Python**

Las tuplas son **estructuras ordenadas e inmutables** , lo que significa**No se pueden modificar** después

### ****Funciones y métodos en Tuplas****

### 🔹 ****Acceso a elementos****

Vaina

pitón

CopiarEditar

tupla = (10, 20, 30, 40)

print(tupla[1]) # Salida: 20

print(tupla[-1]) # Salida: 40 (índice negativo cuenta desde el final)

### 🔹 ****Conteo y búsqueda de elementos****

* count(elemento):Cuenta
* index(elemento): Devuelve el índice de la primera aparición del elemento.

#### **Ejemplo**

pitón

CopiarEditar

tupla = (5, 10, 15, 10, 20)

print(tupla.count(10)) # Salida: 2

print(tupla.index(15)) # Salida: 2

### 🔹 ****Conversión entre listas y tuplas****

A veces necesitamos convertir entre listas y tuplas para

#### **Ejemplo**

pitón

CopiarEditar

lista = [1, 2, 3, 4]

tupla = tuple(lista) # Convierte una lista en tupla

print(tupla) # Salida: (1, 2, 3, 4)

nueva\_lista = list(tupla) # Convierte una tupla en lista

nueva\_lista.append(5) # Ahora podemos modificar la lista

print(nueva\_lista) # Salida: [1, 2, 3, 4, 5]

### 🔹 ****Desempaquetado de Tuplas****

Podemos

#### **Ejemplo**

pito

CopiarEditar

coordenadas = (10, 20)

x, y = coordenadas # Asigna 10 a x y 20 a y

print(x, y) # Salida: 10 20

También podemos utilizar\*párrafo c

pitón

CopiarEditar

numeros = (1, 2, 3, 4, 5)

a, \*b, c = numeros

print(a) # Salida: 1

print(b) # Salida: [2, 3, 4]

print(c) # Salida: 5

## **Conclusión**

### ✅ ****Conjuntos (**** set****)****

* No se permiten duplicados.
* Se pueden modificar (mutables).
* Operaciones matemáticas: unión, intersección, diferencia.
* Métodos útiles: add(), remove(), union(), intersection().

### ✅ ****Tuplas (**** tuple****)****

* Son **ordenadas e inmutables** .
* Permitan acceder a sus elementos con índices.
* Métodos clave: count(), index().
* Se pueden convertir a listas para modificar su contenido.
* Se pueden **desempaquetar** en variables.

Ejemplo de diccionario



<https://aprendeconalf.es/docencia/python/ejercicios/listas-tuplas/>

ejercicios de listas y tuplas

Datos para trabajar

* [Analgésicos](javascript:void(0)): acetaminofén paracetamol, aspirina, diclofenaco, ibuprofeno, naproxeno, nimesulida, piroxicam
* [Antibióticos](javascript:void(0)): amoxicilina, azitromicina, gentamicina, cefazolina, vancomicina.
* Antidepresivos Amitriptilina,fluoxetina (Prozac), Sertralina, Duloxetina
* Antipiréticos: ketoprofeno, nimesulida, metamizol, cilecoxib
* Antialérgicos: Antihistamínicos Cetirizina (Zyrtec), Desloratadina (Clarinex), Fexofenadina (Allegra), Levocetirizina (Zyzal), Loratadina (Claritin).
* Antimicóticos: Clotrimazol, Miconazol, Terbinafina, Ciclopirox, Ketoconazol, Econazol,
* Corticoides: Cortisona, Hidrocortisona, Prednisona, Metilprednisolona, Deflazacort.

Presentación: cápsulas, píldoras, tabletas, soluciones, polvos, granulados, cremas, geles, aerosoles, nebulizadores, jarabes, gotas,