# **Módulo 3**

# **Estructuras de datos no secuenciales**

**Colecciones no secuenciales:**

**Objetos no iterables.**

* Cadenas (str)
* Listas (list)
* Tuplas (tuple)
* slice\_cortes (slice)
* Bytes (bytes) y Bytesarray (bytearray)
* Memoria compartida (memoryview)
* Index

**Set y FrozenSet (Conjuntos):**

Características

Métodos y atributos

* + union (|)
  + intersection (&)
  + difference (-)
  + symmetric\_difference (^)
  + issubset Booleana
  + issuperset Booleana
  + isdisjoint Booleana
  + ...\_update actualizan el conjunto inicial
  + add(elemento)
  + copy()
  + pop()
  + remove(objeto)
  + discard(objeto)
  + clear()

**Diccionarios (dict):**

Características:

**Métodos y atributos**

* + clear ()
  + copy ()
  + fromkeys(iterable, value=None)
  + get(key, default=None)
  + items ()
  + keys ()
  + values ()
  + pop(key, default=None)
  + popitem(key, default=None)
  + setdefault(key, default=None)
  + update(iterable\_or\_dict)

**Función slice()**

**Operadores ternarios:**

* + if
  + else

**Colecciones por comprensión (comprehension).**

* + for
  + if
  + else
* listas y sets:
* diccionario:

**Funciones all() y any().**

# **Estructuras de datos secuenciales**

## **Estructuras de datos**

Las estructuras de datos en Python son objetos que nos permiten almacenar y organizar datos en una forma determinada. Las secuencias son un tipo de estructura de datos que nos permiten almacenar un conjunto de elementos en un orden determinado. Hay tres tipos principales de secuencias en Python: cadenas, listas y tuplas.

**Colecciones no secuenciales:**

Una colección en Python es un tipo de objeto que puede contener múltiples elementos u objetos. Estas colecciones son utilizadas para almacenar, organizar y manipular conjuntos de datos de manera eficiente.

Hay diferentes tipos de colecciones integradas en su biblioteca estándar, que son ampliamente utilizadas en programación.

#### **Objetos no iterables.**

Algunos de estos objetos son los de tipo:

* set
* frozenset
* dict

**Set y FrozenSet (Conjuntos):**

|  |  |  |
| --- | --- | --- |
| **Son colecciones desordenadas, sin index, únicos y mutables de objetos separados por comas.**  Desordenadas  Sin index No tienen un index o indice.  Únicos No permiten duplicados  separados por comas.  **Set** Mutables  **FrozenSet** Inmutables  Estos objetos que la componen pueden ser de cualquier tipo (str, int, float, bool y…...colecciones listas,tuplas, diccionarios, sets, frozensets,etc)  La sintaxis básica es la siguiente:   |  |  | | --- | --- | | **un set vacío**  nombre\_set = **set**()  o  nombre\_set = {}  -------ver mas adelante con dict-------  **una set con objetos int**  nombre\_set = {1,2,3,4,5,6,7,8,9,0} | **un frozenset vacío**  nombre\_fset = **frozenset**()  o  nombre\_fset = ([])  -------**Absurda**, como crear una tupla vacia-------  **una frozenset con objetos int**  nombre\_set = ([1,2,3,4,5,6,7,8,9,0]) |   · |

·

|  |
| --- |
| Los conjuntos se utilizan para almacenar valores sin duplicados y permiten realizar operaciones de **matemática de conjuntos** como intersección, unión y diferencia de manera eficiente.  **Características:**  **Elementos únicos:**  Los conjuntos no pueden contener elementos duplicados. Cada elemento en un conjunto es único.  **Desordenados:**  Los conjuntos no mantienen un orden específico para sus elementos. No puedes acceder a los elementos mediante índices.  **Mutables:**  Los conjuntos **set** son **mutables**, lo que significa que puedes agregar y eliminar elementos después de haber creado el conjunto.  Los conjuntos **frozenset** son **inmutables**, lo que significa que no puedes agregar ni eliminar elementos después de haber creado el conjunto.  **Creación de conjuntos:**  **Set:** Puedes crear un conjunto usando llaves {} o la función set().  **Frozenset:** Puedes crear un conjunto usando paréntesis y dentro corchetes ([]) o la función frozenset(). |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **frozenset**  ·Código Python   |  | | --- | | mi\_conjunto = (["A","B","C","D","E","F"])  print (f"el contenido de {mi\_conjunto=}\t{type(mi\_conjunto)}")  print (f"los métodos disponibles son {dir(mi\_conjunto)=}") |   Salida esperada por consola   |  | | --- | | el el contenido de mi\_conjunto= (['A', 'D', 'B', 'C', 'F', 'E']) <class 'frozenset'>  los métodos disponibles son dir(mi\_conjunto)=[''\_\_and\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_or\_\_', '\_\_rand\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_ror\_\_', '\_\_rsub\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_xor\_\_', 'copy', 'difference', 'intersection', 'isdisjoint', 'issubset', 'issuperset', 'symmetric\_difference', 'union'] |   ·  set  Código Python   |  | | --- | | mi\_conjunto = {"A","B","C","D","E","F"}  print (f"el contenido de {mi\_conjunto=}\t{type(mi\_conjunto)}")  print (f"los métodos disponibles son {dir(mi\_conjunto)=}") |   Salida esperada por consola   |  | | --- | | el el contenido de mi\_conjunto={'A', 'D', 'B', 'C', 'F', 'E'} <class 'set'>  los métodos disponibles son dir(mi\_conjunto)=['\_\_add\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_gt\_\_','\_\_hash\_\_', '\_\_iand\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_ior\_\_', '\_\_isub\_\_', '\_\_iter\_\_', '\_\_ixor\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_or\_\_', '\_\_rand\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_ror\_\_', '\_\_rsub\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_xor\_\_', 'add', 'clear', 'copy', 'difference', 'difference\_update', 'discard', 'intersection', 'intersection\_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric\_difference', 'symmetric\_difference\_update', 'union', 'update'] |   · |

·**Métodos y atributos:**

|  |
| --- |
| Los frozenset son más eficientes en cuanto al uso de la memoria y al rendimiento, lo que las hace más adecuadas para almacenar grandes cantidades de datos. |

·

|  |
| --- |
| Los métodos de los **frozenset** se incluyen en los de los **set**.  Ademas los **set** poseen los siguientes métodos ['add', 'clear', 'discard', 'intersection\_update', 'pop', 'remove', 'symmetric\_difference\_update', ''update'] que son todos aquellos que permiten modificar el objeto frozenset ya que es inmutable. |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | Conjunto = {1,2,3,4,5,6,7,8,9,8,7,6,5,4,3,2,1,0,1,0,2,0,3,4,0,5,0,6,0,7,8,9}  print (f"{Conjunto=}") |   Salida esperada por consola   |  | | --- | | Conjunto={0, 1, 2, 3, 4, 5, 6, 7, 8, 9} |   · |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **union (|):** Devuelve un set con los elementos únicos que están en el conjunto\_a agregando los del conjunto\_b, su viceversa es igual.  Código Python   |  | | --- | | # obj1.union(obj2)  conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  union\_a\_b = conjunto\_a.union(conjunto\_b)  print (f"{union\_a\_b=}")  union\_b\_a = conjunto\_b.union(conjunto\_a)  print (f"{union\_b\_a=}") |   Salida esperada por consola   |  | | --- | | union\_a\_b={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}  union\_b\_a={1, 2, 3, 4, 5, 6, 7, 8, 9, 10} |   ·con signos |  Código Python   |  | | --- | | # obj1 | obj2  conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  union\_a\_b = conjunto\_a|conjunto\_b  print (f"{union\_a\_b=}")  union\_b\_a = conjunto\_b|conjunto\_a  print (f"{union\_b\_a=}") |   Salida esperada por consola   |  | | --- | | union\_a\_b={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}  union\_b\_a={1, 2, 3, 4, 5, 6, 7, 8, 9, 10} |   · |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **intersection (&):** Devuelve un set con los elementos únicos que están tanto en el conjunto\_a como en el conjunto\_b, su viceversa es igual.  Código Python.   |  | | --- | | # obj1.intersection(obj2)  conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  intersection\_a\_b = conjunto\_a.intersection(conjunto\_b)  print (f"{intersection\_a\_b=}")  intersection\_b\_a = conjunto\_b.intersection(conjunto\_a)  print (f"{intersection\_b\_a=}") |   Salida esperada por consola   |  | | --- | | intersection\_a\_b={4, 5, 6}  intersection\_b\_a={4, 5, 6} |   ·con signos &  Código Python   |  | | --- | | # obj1 & obj2  conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  intersection\_a\_b = conjunto\_a & conjunto\_b  print (f"{intersection\_a\_b=}")  intersection\_b\_a = conjunto\_b & conjunto\_a  print (f"{intersection\_b\_a=}") |   Salida esperada por consola   |  | | --- | | intersection\_a\_b={4, 5, 6}  intersection\_b\_a={4, 5, 6} | |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **difference (-):** Devuelve un set con los elementos que están en el conjunto\_a y no en el conjunto\_b.  Código Python   |  | | --- | | # obj1.difference (obj2)  conjunto\_a = {1, 2, 3, 4, 5, 6, 7 ,8, 9}  conjunto\_b = {2, 4, 6, 8, 10}  diferencia\_a\_b = conjunto\_a.difference(conjunto\_b)  print (f"{diferencia\_a\_b=}")  diferencia\_b\_a = conjunto\_b.difference(conjunto\_a)  print (f"{diferencia\_b\_a=}") |   Salida esperada por consola   |  | | --- | | diferencia\_a\_b={1, 3, 5, 7, 9}  diferencia\_b\_a={10} |   ·con signos -  Código Python   |  | | --- | | # obj1-obj2  conjunto\_a = {1, 2, 3, 4, 5, 6, 7 ,8, 9}  conjunto\_b = {2, 4, 6, 8, 10}  diferencia\_a\_b = conjunto\_a-conjunto\_b  print (f"{diferencia\_a\_b=}")  diferencia\_b\_a = conjunto\_b-conjunto\_a  print (f"{diferencia\_b\_a=}") |   Salida esperada por consola   |  | | --- | | diferencia\_a\_b={1, 3, 5, 7, 9}  diferencia\_b\_a={10} |   · |

·

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **symmetric\_difference (^):** Devuelve un set con los elementos únicos que están en el conjunto\_a y en el conjunto\_b, pero no los que están en común (elimina los de intersección), su viceversa es igual.  Código Python   |  | | --- | | # obj1.symmetric\_difference(obj2)  conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  symmetric\_difference\_a\_b = conjunto\_a.symmetric\_difference(conjunto\_b)  print (f"{symmetric\_difference\_a\_b=}")  symmetric\_difference\_b\_a = conjunto\_b.symmetric\_difference(conjunto\_a)  print (f"{symmetric\_difference\_b\_a=}") |   Salida esperada por consola   |  | | --- | | symmetric\_difference\_a\_b={4, 5, 6}  symmetric\_difference\_b\_a={4, 5, 6} |   ·con signos ^  Código Python   |  | | --- | | # obj1 ^ obj2  conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  symmetric\_difference\_a\_b = conjunto\_a ^ conjunto\_b  print (f"{symmetric\_difference\_a\_b=}")  symmetric\_difference\_b\_a = conjunto\_b ^ conjunto\_a  print (f"{symmetric\_difference\_b\_a=}") |   Salida esperada por consola   |  | | --- | | symmetric\_difference\_a\_b={4, 5, 6}  symmetric\_difference\_b\_a={4, 5, 6} |   · |

·

|  |  |  |
| --- | --- | --- |
| **issubset:** Booleana, True si el conjunto\_a es un subconjunto del conjunto\_b especificado .  False si no lo es.  Código Python   |  | | --- | | conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6}  conjunto\_c = { 4, 5, 6, 7, 8, 9,10}  issubset\_a\_b = conjunto\_a.issubset(conjunto\_b)  print (f"{issubset\_a\_b=}")  issubset\_b\_a = conjunto\_b.issubset(conjunto\_a)  print (f"{issubset\_b\_a=}")  print ("\*"\*50)#----------------------------------------------  issubset\_b\_c = conjunto\_b.issubset(conjunto\_c)  print (f"{issubset\_b\_c=}")  issubset\_c\_b = conjunto\_c.issubset(conjunto\_b)  print (f"{issubset\_c\_b=}")  print ("\*"\*50)#----------------------------------------------  issubset\_b\_c = conjunto\_b.issubset(conjunto\_c)  print (f"{issubset\_b\_c=}")  issubset\_b\_a = conjunto\_c.issubset(conjunto\_b)  print (f"{issubset\_c\_b=}") |   Salida esperada por consola   |  | | --- | | issubset\_a\_b=False  issubset\_b\_a=True  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  issubset\_b\_c=True  issubset\_c\_b=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  issubset\_b\_c=True  issubset\_c\_b=False | |

·

|  |  |  |
| --- | --- | --- |
| **issuperset:** Booleana, True si el conjunto\_a es un subconjunto del conjunto\_b especificado .  False si no lo es.  Código Python   |  | | --- | | conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6}  conjunto\_c = { 4, 5, 6, 7, 8, 9,10}  issuperset\_a\_b = conjunto\_a.issuperset(conjunto\_b)  print (f"{issuperset\_a\_b=}")  issuperset\_b\_a = conjunto\_b.issuperset(conjunto\_a)  print (f"{issuperset\_b\_a=}")  print ("\*"\*50)#----------------------------------------------  issuperset\_b\_c = conjunto\_b.issuperset(conjunto\_c)  print (f"{issuperset\_b\_c=}")  issuperset\_c\_b = conjunto\_c.issuperset(conjunto\_b)  print (f"{issuperset\_c\_b=}")  print ("\*"\*50)#----------------------------------------------  issuperset\_b\_c = conjunto\_b.issuperset(conjunto\_c)  print (f"{issuperset\_b\_c=}")  issuperset\_b\_a = conjunto\_c.issuperset(conjunto\_b)  print (f"{issuperset\_c\_b=}") |   Salida esperada por consola   |  | | --- | | issuperset\_a\_b=True  issuperset\_b\_a=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  issuperset\_b\_c=False  issuperset\_c\_b=True  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  issuperset\_b\_c=False  issuperset\_c\_b=True |   · |

**·**

|  |  |  |
| --- | --- | --- |
| **isdisjoint:** Booleana, True si el conjunto\_a no tienen elementos en común con del conjunto\_b especificado .  False si no lo es.  Código Python   |  | | --- | | conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  conjunto\_c = {20, 30, 40, 50, 60, 70, 80, 90}  isdisjoint\_a\_b = conjunto\_a.isdisjoint(conjunto\_b)  print (f"{isdisjoint\_a\_b=}")  isdisjoint\_b\_a = conjunto\_b.isdisjoint(conjunto\_a)  print (f"{isdisjoint\_b\_a=}")  print ("\*"\*50)#----------------------------------------------  isdisjoint\_b\_c = conjunto\_b.isdisjoint(conjunto\_c)  print (f"{isdisjoint\_b\_c=}")  isdisjoint\_c\_b = conjunto\_c.isdisjoint(conjunto\_b)  print (f"{isdisjoint\_c\_b=}")  print ("\*"\*50)#----------------------------------------------  isdisjoint\_b\_c = conjunto\_b.isdisjoint(conjunto\_c)  print (f"{isdisjoint\_b\_c=}")  isdisjoint\_b\_a = conjunto\_c.isdisjoint(conjunto\_b)  print (f"{isdisjoint\_c\_b=}") |   Salida esperada por consola   |  | | --- | | isdisjoint\_a\_b=False  isdisjoint\_b\_a=False  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isdisjoint\_b\_c=True  isdisjoint\_c\_b=True  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  isdisjoint\_b\_c=True  isdisjoint\_c\_b=True |   · |

·

**El métodos es de los conjuntos set y no Frozenset**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Los métodos update modifican el conjunto\_1 (inicial) actualizándolo con los elementos del conjunto\_2 (secundario), su viceversa es igual.  Código Python   |  | | --- | | conjunto\_a = { 1, 2, 3, 4, 5, 6}  conjunto\_b = { 4, 5, 6, 7, 8, 9,10}  #----------------------------------------------  print (f"Antes")  print (f"{conjunto\_a=}")  print (f"{conjunto\_b=}")  print ("\*"\*50)#----------------------------------------------  print (f"Despues")  conjunto\_a.update(conjunto\_b)  print (f"{conjunto\_a=}")  conjunto\_b.update(conjunto\_a)  print (f"{conjunto\_b=}") |   Salida esperada por consola   |  | | --- | | Antes  conjunto\_a={1, 2, 3, 4, 5, 6}  conjunto\_b={4, 5, 6, 7, 8, 9, 10}  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Despues  conjunto\_a={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}  conjunto\_b={1, 2, 3, 4, 5, 6, 7, 8, 9, 10} |   ·   |  | | --- | | Los métodos terminados en \_update actualizan el conjunto inicial con el resultado del método especificados  conjunto\_r=conjunto\_1.difference(conjunto\_2)# se carga en conjunto\_r  conjunto\_1.difference\_update(conjunto\_2)# sobrescribe conjunto₁  conjunto\_r=conjunto\_1.intersection(conjunto\_2) # se carga en conjunto\_r  conjunto\_1. 'intersection\_update(conjunto\_2)# sobrescribe conjunto₁  conjunto\_r=conjunto\_1.symmetric\_difference(conjunto\_2) # se carga en conjunto\_r  conjunto\_1.symmetric\_difference\_update(conjunto\_2)# sobrescribe conjunto\_1 |   ·   |  | | --- | | **add(elemento):**  Agrega un elemento al conjunto si no está presente.  **copy():**  Crea y devuelve una copia superficial del conjunto.  **pop():**  Remueve y devuelve un elemento **aleatorio** del conjunto.  Si el conjunto está vacío, **genera un error KeyError.**  **El proposito de remover un elemento aleatorio despues de 20 años de python aun me es totalmente desconocido**  **remove(objeto):**  Elimina un elemento específico del conjunto.  Si el elemento no existe, **genera un error KeyError.**  **discard(objeto):**  Elimina un elemento específico del conjunto.  Si el elemento no existe, **no genera ningún error.**  **clear():**  Elimina todos los elementos del conjunto, dejándolo vacío.  **No elimina el objetos set** |   · |

·

**Diccionarios (dict):**

|  |  |
| --- | --- |
| **Son colecciones desordenadas, sin index, con claves únicos y mutables, con pares de objetos separados por comas.**  Desordenadas  Sin index No tienen un index o indice.  pares separados por comas.  **keys** Inmutables  únicos - No permiten duplicados  Estos objetos que la componen pueden ser objetos inmutables  (str, int, float, bool y tuplas, frozensets)  **separador :**  **values** Mutables  Permiten duplicados  Estos objetos que la componen pueden ser de objetos cualquier tipo  (str, int, float, bool y…...colecciones listas,tuplas, diccionarios, sets, frozensets,etc)  Estos objetos que la componen pueden ser de cualquier tipo (str, int, float, bool y…...colecciones listas,tuplas, diccionarios, sets, frozensets,etc)  La sintaxis básica es la siguiente:   |  | | --- | | **un diccionario vacío**  nombre\_ dic = dict()  o  nombre\_ dic = {}  -------ver diferencia con {} de sets-------  **un diccionario con claves strings y valores int**  nombre\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5} |   · |

·

|  |
| --- |
| Recuerda que para que los objetos puedan ser claves de un diccionario, deben ser inmutables, ya que si un objeto mutable (como una lista) se usara como clave, podría conducir a comportamientos inesperados ya que las claves deben ser únicas y no cambiantes |
| **Clave string**  Cadenas (Strings): Las cadenas son una de las opciones más comunes para las claves de diccionario, ya que son inmutables y proporcionan una forma legible de asociar valores.  my\_dict = {"nombre": "Juan", "edad": 25} |
| **Clave numérica entera o flotante**  grades = {98: "A+", 85: "B", 70: "C"}  Números (int, float): Los números enteros y de punto flotante también pueden ser claves de diccionario. |
| **Clave Tuplas**  Las tuplas también pueden usarse como claves de diccionario si solo contienen elementos inmutables.  point = (3, 5)  locations = {point: "Punto de interés"} |
| **Clave booleana (solo 2 posibilidades)**  Booleanos: Los valores booleanos True y False también pueden ser usados como claves.  status = {True: "Activo", False: "Inactivo"} |
| **Clave frozenset**  Los frozensets (conjuntos inmutables) también son válidos como claves de diccionario debido a su inmutabilidad.  mapping = {frozenset([1, 2]): "Conjunto 1-2"} |
| **una dict con claves-keys con objetos permitidos**  my\_dict = {  "nombre": "key str",  42 : "key int",  (1, 2, 3) : "key tuple",  3.14 : "key float",  frozenset([1,2, 3]) : "key frozenset"  }  **una dict con valores con objetos simples y colecciones**  my\_dict = {  "value str" : "Juan",  "value int" : 25,  "value float" : 3.14159,  "value complex" : 3j  "value tuple" : (1, 2, 3)  "value list" : [1, 2, 3]  "value set" : {1, 2, 3}  "value frozenset" : frozenset([1, 2, 3])  "value dict" : {  "1er sub key" : 1,  "2do sub key" : 2,  "3er sub key" : 3,  "4to sub key" : 4  }  } |

·

|  |  |  |
| --- | --- | --- |
| **Diccionarios - dict**  ·Código Python   |  | | --- | | mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"el contenido de {mi\_dic=}\t{type(mi\_dic)}")  print (f"los métodos disponibles son {dir(mi\_dic)=}") |   Salida esperada por consola   |  | | --- | | el el contenido de mi\_conjunto= (['A', 'D', 'B', 'C', 'F', 'E']) <class 'frozenset'>  los métodos disponibles son dir(mi\_conjunto)=['['\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_ior\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_or\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_ror\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']' |   · |

·**Métodos y atributos:**

|  |
| --- |
| Los diccionarios son objetos modernos de alto nivel, se refiere al valor por una clave que puede tener varios formatos,no un numero entero como es un index. |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"{mi\_dic=}") |   Salida esperada por consola   |  | | --- | | mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5} |   · |

·

|  |  |  |
| --- | --- | --- |
| **clear ():**  Este método elimina todos los elementos del diccionario, dejándolo vacío.  Código Python   |  | | --- | | # obj1.clear ()  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  mi\_dic.clear()  print (f"despues:{mi\_dic=}") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  despues:mi\_dic={} |   · |

·

|  |  |  |  |
| --- | --- | --- | --- |
| **copy ():**  Este método elimina todos los elementos del diccionario, dejándolo vacío.   |  | | --- | | Ver alias |   Código Python   |  | | --- | | # obj\_nuevo=obj\_original .copy ()  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  nuevo\_dic = mi\_dic.copy ()  print (f"despues:{nuevo\_dic=}") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  despues:nuevo\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5} |   · |

·

|  |  |  |
| --- | --- | --- |
| **fromkeys(iterable, value=None):**  Crea un nuevo diccionario con claves tomadas de un iterable dado y asigna un valor opcional a todas las claves. Si value tiene un dato (o None-vacio) este se repite en todas las claves  Código Python   |  | | --- | | # fromkeys(iterable, value=None):  mi\_lista = [1,2,3,4,5]  print (f"origen:{mi\_lista=} tipo:{type(mi\_lista)}")  mi\_dic=dict.fromkeys(mi\_lista)  print (f"resultado:{mi\_dic=}")  print ("\*"\*50)#----------------------------------------------------------  mi\_dic=dict.fromkeys(mi\_lista,"Python")  print (f"resultado:{mi\_dic=}") |   Salida esperada por consola   |  | | --- | | resultado:mi\_dic={1: None, 2: None, 3: None, 4: None, 5: None}  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  resultado:mi\_dic={1: 'Python', 2: 'Python', 3: 'Python', 4: 'Python', 5: 'Python'} |   · |

·

|  |  |  |
| --- | --- | --- |
| **get(key, default=None) :**  Devuelve el valor asociado con la clave especificada. Si la clave no está presente, devuelve el valor predeterminado o None.  Código Python   |  | | --- | | # obj1.get(key, None)  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"original:{mi\_dic=}")  clave = "uno"  error = "no existe esa clave, no hay valor"  valor\_esperado = mi\_dic.get(clave,error)  print (f"la clave:{clave} tiene asociado el valor:{valor\_esperado}")  print ("\*"\*50)#----------------------------------------------------------  clave = "ZZZ"  valor\_esperado = mi\_dic.get(clave,error)  print (f"la clave:{clave} tiene asociado el valor:{valor\_esperado}")  exit() |   Salida esperada por consola   |  | | --- | | original:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  la clave:uno tiene asociado el valor:1  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  la clave:ZZZ tiene asociado el valor:no existe esa clave, no hay valor |   · |

·

|  |  |  |
| --- | --- | --- |
| **items ():**  Devuelve una vista en forma de tuplas de todos los pares clave-valor del diccionario.  Código Python   |  | | --- | | # obj1.items ()  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  items=mi\_dic.items()  print (f"tuplas:{items=}") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  tuplas:items=dict\_items([('uno', 1), ('dos', 2), ('tres', 3), ('cuatro', 4), ('cinco', 5)]) |   · |

·

|  |  |  |
| --- | --- | --- |
| **keys ():**  Devuelve una vista de todas las claves en el diccionario.  Código Python   |  | | --- | | # obj1.keys ()  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  claves=mi\_dic.keys()  print (f"keys:{claves}") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  keys:dict\_keys(['uno', 'dos', 'tres', 'cuatro', 'cinco']) |   · |

·

|  |  |  |
| --- | --- | --- |
| **values ():**  Devuelve una vista de todos los valores en el diccionario.  Código Python   |  | | --- | | # obj1.values ()  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  valores=mi\_dic.values ()  print (f"values:{valores}") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  values:dict\_values([1, 2, 3, 4, 5]) |   · |

·

|  |  |  |
| --- | --- | --- |
| **pop(key, default=None):**  Elimina y devuelve el valor asociado con la clave especificada. Si la clave no está presente, devuelve el valor predeterminado o genera un error.  Código Python   |  | | --- | | # pop(key, default=None)  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  age = my\_dict.pop("tres", 0)  print (f"despues:{mi\_dic= }") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  despues:mi\_dic= {'uno': 1, 'dos': 2, 'cuatro': 4, 'cinco': 5} |   · |

·

|  |  |  |  |
| --- | --- | --- | --- |
| **popitem(key, default=None):**  Elimina y devuelve un par clave-valor arbitrario del diccionario como una tupla.   |  | | --- | | No encuentro el motivo de este metodo |   Código Python   |  | | --- | | # popitem()  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  borrado = mi\_dic.popitem()  print (f"despues:{mi\_dic= }")  print (f"borrado:{borrado }") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  despues:mi\_dic= {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4}  borrado:('cinco', 5) |   · |

·

|  |  |  |
| --- | --- | --- |
| **setdefault(key, default=None):**  Si la clave está presente, devuelve su valor. Si no está presente, agrega la clave con el valor predeterminado (o el valor proporcionado) y luego devuelve ese valor  Código Python   |  | | --- | | # setdefault(key, default=None)  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  print (f"antes:{mi\_dic=}")  clave="cuatro"  mi\_dic.setdefault(clave, "nuevo")  print (f"despues:{mi\_dic= }")  print ("\*"\*50)#----------------------------------------------------------  clave = "ZZZ"  mi\_dic.setdefault(clave, "nuevo")  print (f"despues:{mi\_dic= }") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  despues:mi\_dic= {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  despues:mi\_dic= {'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5, 'ZZZ': 'nuevo'} |   · |

|  |  |  |
| --- | --- | --- |
| **update(iterable\_or\_dict):**  Actualiza el diccionario con los pares clave-valor de otro diccionario o un iterable de pares clave-valor.  Código Python   |  | | --- | | # update(iterable\_or\_dict)  mi\_dic = {"uno":1,"dos":2,"tres":3,"cuatro":4,"cinco":5}  update\_dic = {"dos":2,"cuatro":4,"siete":6,"ocho":8,"diez":10}  print (f"antes:{mi\_dic=}")  borrado = mi\_dic.update(update\_dic)  print (f"despues:{update\_dic= }") |   Salida esperada por consola   |  | | --- | | antes:mi\_dic={'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4, 'cinco': 5}  despues:update\_dic= {'dos': 2, 'cuatro': 4, 'siete': 6, 'ocho': 8, 'diez': 10} |   · |

·

|  |
| --- |
| Funcion slice()  slice(start, stop, step)  Donde start , stop y step son argumentos opcionales que indican el índice inicial, el índice final  y el tamaño del paso de la porción deseada. Si se omite alguno de estos argumentos, se utiliza un valor predeterminado.  replace()  Método al que se le ingresan un patrón y un texto de sustitución. Regresará un objeto tipo str sustituyendo cada coincidencia. En caso de no encontrar coincidencias, regresará un objeto tipo str con el texto original. Permite elejir la cantidad de remplazos  -Tipo de Dato Bytes  ﻿﻿La principal diferencia entre los tipos de datos bytes y str en Python es que los bytes representan datos en formato binario en lugar de texto, mientras que los str representan datos de texto en formato Unicode.  Otra diferencia importante es la forma en que se almacenan los datos. Las cadenas (str) en Python se representan como una secuencia de caracteres Unicode, mientras que los bytes se representan como una secuencia de valores de bytes.  Por lo tanto, los datos binarios, como archivos de imagen, sonidos o vídeos, se almacenan típicamente como bytes en Python.  Además, mientras que las cadenas pueden ser diseñadas como cadenas literales utilizando comillas simples o dobles, los bytes se crean utilizando el prefijo `b` antes de la cadena literal. |

·

|  |
| --- |
| **Resumen de Diccionarios (dict)**  Los objetos de tipo dict son una colección de elementos en la que los índices en vez de ser numéricos corresponden a un "identificador".  Algunas de las características importantes de los diccionarios en Python son:  • Los diccionarios son una estructura de datos mutable que permite agregar, modificar y eliminar elementos después de haber sido creados.  • Los diccionarios son una colección de pares clave-valor, donde cada clave debe ser única y se utiliza para acceder al valor correspondiente almacenado en el diccionario.  • Los diccionarios son una estructura de datos sin orden, lo que significa que los elementos no se almacenan en una posición específica dentro del diccionario. Aunque a partir de python 3,7 pasaron a tener un orden interno pero no es indexable.  • Los diccionarios pueden contener cualquier tipo de dato como valor, incluyendo números, cadenas, listas, tuplas, diccionarios y objetos personalizados.  • Los diccionarios en Python utilizan una tabla hash interna para almacenar los pares clave-valor, lo que proporciona un acceso rápido y eficiente a los elementos del diccionario.  El identificador debe de ser un objeto inmutable tal como es el caso de:  • Objetos de tipo str.  • Objetos de tipo int.  • Objetos de tipo float.  • Objetos de tipo complex.  • Objetos de tipo bool.  • Objetos de tipo bytes.  • Objetos de tipo tuple.  Lo más común y recomendable es utilizar objetos de tipo str o int.  Es posible crear objetos de tipo dict mediante la función dict(). En este caso los identificadores deben de cumplir con las reglas para la definición de nombres y no llevan apóstrofes ni comillas. La relación entre el identificador y el valor se define mediante el operador de asignación (=).  **dict(<nombre 1>=<valor 1>, <nombre2 >=<valor 2>, ... , <nombre n>=<valor n>)**  o literalmente con las llaves :  e hace una secuencia de pares clave: valor, separados por comas y encerrados entre llaves ({ }).  a sintaxis es la siguiente:  **<identificador 1>: <objeto 1>, <identificador 2>: <objeto 2>, ..., <identificador n>: <objeto n>}**  Modificación o adición de un elemento contenido en un objeto de tipo dict. Para modificar un elemento de un objeto tipo dict se utiliza el operador de asignación (\_ =\_ ).  **<objeto tipo dict>[<identificador>] = <valor>**  En caso de que no exista un elemento con ese identificador, el elemento se añadirá al objeto tipo dict.  Para obtener el valor llamamos al diccionario por la clave la sintaxis es dicionario["clave"] . y los valores puedes ser cualquier objeto  **get()** es un método incorporado en Python para el tipo de datos de diccionario. Este método se utiliza para obtener un valor correspondiente a una clave en un diccionario. La sintaxis básica del método es la siguiente:  diccionario.get(clave, valor\_por\_defecto)`  Donde `clave` es el elemento que se desea obtener del diccionario y `valor\_por\_defecto` es el valor que se devuelve si la clave no se encuentra en el diccionario. Si `valor\_por\_defecto` no se proporciona, el valor predeterminado es `None`.  Una de las principales ventajas de este método es que, si se usa para obtener un valor de un diccionario, si la clave no existe en el diccionario, en lugar de generar un error, simplemente devuelve el valor por defecto (si se proporcionó uno) o `None`.  Otra ventaja de usar el método `get()` es que se puede usar para evitar la duplicación de código innecesaria. Por ejemplo, en lugar de escribir un bloque de código para verificar si una clave existe en un diccionario y luego obtener el valor correspondiente a esa clave, se puede usar una sola línea de código con el método `get()`.  **update()** sustituye y añade los elementos que se ingresan como argumento. El argumento puede ser un objeto de tipo dict o un objeto iterable que contenga pares compatibles con el fomato clave:valor.  **setdefault().** ingresan un identificador y un valor como argumentos. En caso de no ingresar un valor, el valor por defecto es None. Si ya existe un elemento cuyo identificador coincide con el ingresado, regresa el valor correspondiente del elemento existente. Si no existe un elemento con el nombre del identificador, entonces se añade un elemento conteniendo el identificador y valor ingresados, y regresa dicho valor.  **keys()** Regresa un objeto iterable que genera una secuencia que corresponde al identificador de cada elemento del objeto tipo dict.  **values()** Regresa un objeto iterable que genera una secuencia que corresponde al valor de cada elemento del objeto tipo dict.  **items().**Regresa un objeto iterable que genera una secuencia de objetos de tipo tuple con el indicador y el valor de cada elemento.  **pop()** Regresa el valor ligado al identificador que se ingresa como argumento. En caso de no encontrar un identificador igual al argumento, se genera un error de tipo KeyError. En caso de no ingresar un argumento, se generará un error de tipo TypeError.  **popitems()**Regresa de forma aleatoria una tupla conteniendo el identificado y el valor de un elemento y lo elimina del objeto tipo dict. En caso de que el objeto tipo dict esté vació, se generará un errorde tipo KeyError.  **lear().**Este método elimina todos los elementos de un diccionario.  **copy().**Regresa un nuevo objeto de tipo dict con los mismos elementos que contiene el objeto  En definitiva los diccionario de python son las estructura que se utilizan mucho en la proporciona orientada a objetos y también al ser muy similares a los json como intercambio de información entre aplicaciones |

·

|  |
| --- |
| **Resumen de conjuntos (set)**  Los conjuntos en Python son una estructura de datos que nos permiten almacenar elementos de manera única e inmutable. En otras palabras, no se puede duplicar elementos en un conjunto.  Los conjuntos se definen colocando sus elementos entre llaves `{}` o utilizando la función `set()`. Ejemplo:  mi\_conjunto = {1, 2, 3, 4}  otro\_conjunto = set([3, 4, 5, 6])  Los conjuntos son útiles cuando necesitamos almacenar elementos sin preocuparnos por el orden o la duplicidad. Además, proporcionan operaciones matemáticas básicas como unión, intersección, diferencia, comprobación de pertenencia, entre otras.  Para ilustrar de mejor manera los conceptos relacionados con conjuntos se utilizarán diagramas de Venn y su relación entre 2 conjuntos genéricos.  Métodos de manipulacion  Los conjuntos en Python son una estructura de datos que nos permiten almacenar elementos de manera única e inmutable. En otras palabras, no se puede duplicar elementos en un conjunto.  Los conjuntos se definen colocando sus elementos entre llaves `{}` o utilizando la función `set()`. Ejemplo:  mi\_conjunto = {1, 2, 3, 4}  otro\_conjunto = set([3, 4, 5, 6])  Los conjuntos son útiles cuando necesitamos almacenar elementos sin preocuparnos por el orden o la duplicidad. Además, proporcionan operaciones matemáticas básicas como unión, intersección, diferencia, comprobación de pertenencia, entre otras.  Para ilustrar de mejor manera los conceptos relacionados con conjuntos se utilizarán diagramas de Venn y su relación entre 2 conjuntos genéricos.  **add().**Añade un elemento que se ingresa como argumento al objeto tipo set. Si ya hay un objeto equivalente, no se agrega uno nuevo  **remove().** Busca y elimina al elemento que se ingresa como argumento. En caso de no encontrarlo, envía un mensaje de error.  **Discard().**Busca y elimina al elemento que se ingresa como argumento. En caso de no encontrarlo, no regresa nada.  **Pop().**Regresas y elimina un elemento del objeto tipo set. Cuando ya no existen elementos, se generan un error de tipo KeyError  **clear().**Este método elimina todos los elementos de un diccionario.  Regresa un nuevo objeto de tipo dict con los mismos elementos que contiene el objeto  **isdisjoint.** Evalúa si dos conjuntos no se interceptan. Si no existen elementos compartidos entre el objeto tipo set al que pertenece el método y el objeto set que se ingresa como argumento, se regresa el valor True. De lo contrario se regresa False.  **Issubset﻿﻿** Si el objeto tipo set al que pertenece el método es subconjunto del objeto set que se ingresa como argumento, se regresa el valor True. De lo contrario se regresa False.  ﻿﻿**issuperset** Si el objeto tipo set que se ingresa como argumento es subconjunto del objeto tipo set al que pertenece el método, se regresa el valor True. De lo contrario se regresa False.  **Union﻿** ﻿﻿Regresa el resultado de la unión del objeto tipo set al que pertenece el método y el objeto tipo set que se ingresa como argumento.  ﻿**difference** ﻿﻿Regresa el resultado de la diferencia entre el objeto tipo set al que pertenece el método y el objeto tipo set que se ingresa como argumento.  intersection  Regresa el resultado de la intersección entre el objeto tipo set al que pertenece el método y el objeto tipo set que se ingresa como argumento.  **symmetric\_difference﻿﻿** ﻿﻿Regresa el resultado de la diferencia simétrica entre el objeto tipo set al que pertenece el método y el objeto tipo set que se ingresa como argumento. |

**Operadores ternarios:**

Los operadores ternarios u operadores condicionales, permiten expresar una condición ( if-else en una sola línea) y realizar una asignación o retorno de valores basados en esa condición.

Permite en una forma concisa de expresar una estructura condicional en una sola línea. A diferencia de una estructura if-else tradicional, los operadores ternarios permiten evaluar una expresión y retornar un resultado basado en una condición.

|  |
| --- |
| Operadores ternarios solo se trabaja con   * + if   + else |

·

|  |
| --- |
| Solo se trabaja con if else |

La sintaxis básica del operador ternario en Python es la siguiente:

**salida ="**valor o expresión verdadero**" if condición else "**valor o expresión falsa**"**

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | edad = 18  comprar\_alcohol = True if edad >= 18 else False  print(f"puedes comprar alcohol? {comprar\_alcohol}") |   Salida esperada por consola   |  | | --- | | puedes comprar alcohol? True |   · |

·

salida = ("valor o expresión verdadera","valor o expresión falsa")[booleana]

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | anda = True  estado = ("si: el codigo corre", "bug: no, el código no corre")[anda]  print("estado del programa ", estado) |   Salida esperada por consola   |  | | --- | | estado del programa bug: no, el código no corre |   · |

·

|  |
| --- |
| Los operadores ternarios son útiles para expresar de manera concisa estructuras condicionales simples en una sola línea de código.  Según la comunidad, no son métodos Python estándar , es importante utilizarlos con moderación y asegurarse de que la expresión sea legible y comprensible para otros programadores que lean el código.  Como en varios elementos (el mas conocido operador walrus :=) la comunidad asegura que confunde el código y que resta mas de lo que suma |

**Colecciones por comprensión (comprehension).**

Python permite construir de con una sintaxis compacta y de forma simple y rápida listas, conjuntos y diccionarios (list,str & dict). Son una forma elegante y eficiente de crear colecciones en base a iteraciones y condiciones.

·

|  |
| --- |
| Colecciones por comprensión solo se trabaja con   * + for   + if   + else   (aunque pueden ser varios anidados) |

La sintaxis básica de comprensiones de colecciones en Python es la siguiente:

iterables: un iterable es un objeto que se puede recorrer o iterar elemento por elemento.

Básicamente, cualquier objeto que pueda devolver uno a la vez sus elementos cuando se utiliza en un bucle for se considera un iterable.

Los iterables pueden ser objetos como listas, tuplas, conjuntos, diccionarios, cadenas de texto y otros.

Cuando se utiliza un bucle for para iterar sobre un iterable, se obtiene acceso a cada elemento del iterable en sucesión. Esto permite realizar operaciones en cada elemento o realizar algún tipo de procesamiento secuencial.

En todos estos ejemplos, se utiliza un bucle for para iterar sobre los elementos de los objetos iterables. En cada iteración, se obtiene un elemento del iterable y se realiza alguna operación con él.

Es importante tener en cuenta que un objeto iterable se puede recorrer una vez. Después de que se haya agotado su secuencia de elementos, no se puede volver a recorrer a menos que se cree un nuevo iterable o se reinicie de alguna manera.

En resumen, un iterable en Python es cualquier objeto que se puede recorrer o iterar elemento por elemento, lo que permite realizar operaciones en cada elemento de forma secuencial.

La sintaxis básica de un objeto por comprensión en Python es la siguiente:

**listas y sets:**

**[**"valor o expresión" for elemento in iterable **]**

**[**"valor o expresión verdadero" for elem in iterable **if condición ]**

**[**"valor/expresión verdadero" **if condición else** "**valor/expresión falso**" for elem in iterable**]**

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | numeros = [2, 9, 6, 25, 8, 49, 10, 81, 12 ]  salidas = [cada\_numero \*\* 2 for cada\_numero in numeros]  print (f"{salidas=}")  #--------------------------------------------------------  salidas = [cada\_numero \*\* 2 for cada\_numero in numeros if cada\_numero % 2 == 0]  print (f"{salidas=}")  #--------------------------------------------------------  salidas = [cada\_numero \*\* 2 if cada\_numero % 2 == 0 else cada\_numero \*\*(1/2) for cada\_numero in numeros]  print (f"{salidas=}")  #-------------------------------------------------------- |   Salida esperada por consola   |  | | --- | | salidas=[4, 81, 36, 625, 64, 2401, 100, 6561, 144]  salidas=[4, 36, 64, 100, 144]  salidas=[4, 3.0, 36, 5.0, 64, 7.0, 100, 9.0, 144] |   · |

·

**diccionario:**

**{**clave: valor **for** clave, valor **in** diccionario.items() }

**{**clave: valor **for** clave, valor **in** diccionario.items() if condición }

**{**clave: valor True **if condición else** "**valor falso**" **for** clave,valor **in** diccionario.items()}

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | numeros\_dic={"1ro" : 6 , "2do" : 49 , "3ro" : 25 , "4to" : 4, "5to" : 9}  salidas = {cada\_clave : cada\_numero \*\* 2 for cada\_clave,cada\_numero in numeros\_dic.items()}  print (f"{salidas=}")  #--------------------------------------------------------  salidas = {cada\_clave : cada\_numero \*\* 2 for cada\_clave,cada\_numero in numeros\_dic.items() if cada\_numero % 2 == 0}  print (f"{salidas=}")  #--------------------------------------------------------  salidas = {cada\_clave : cada\_numero \*\* 2 if cada\_numero % 2 == 0 else cada\_numero \*\*(1/2) for cada\_clave,cada\_numero in numeros\_dic.items() }  print (f"{salidas=}")  #-------------------------------------------------------- |   Salida esperada por consola   |  | | --- | | salidas={'1ro': 36, '2do': 2401, '3ro': 625, '4to': 16, '5to': 81}  salidas={'1ro': 36, '4to': 16}  salidas={'1ro': 36, '2do': 7.0, '3ro': 5.0, '4to': 16, '5to': 3.0} |   · |

·

En este ejemplo, se utiliza una comprensión de diccionario para generar un diccionario donde las claves son los nombres y los valores son números enteros.

**Funciones all y any.**

La función all y any son funciones integradas en Python que permiten verificar si todos o alguno de los elementos de una secuencia (lista, tupla, conjunto, generador.) son verdaderos (True) o falsos (False). Estas funciones pueden ser útiles cuando se trabaja con expresiones booleanas y se desea evaluar la veracidad de un conjunto de valores.

**all()**

**Devuelve verdadero:**si todas las condiciones o los elementos son verdaderos (o si el iterable está vacío).

**Devuelve False**: si está vacío o si todos son falsos.

Todo se puede considerar como una secuencia de operaciones AND en los iterables proporcionados.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | todo= all([True, True, True, True])  print (f"{todo=}")  #--------------------------------------------------------  todo= all([False, True, True, False])  print (f"{todo=}")  #--------------------------------------------------------  todo= all([False, False, False])  print (f"{todo=}")  #-------------------------------------------------------- |   Salida esperada por consola   |  | | --- | | todo=True  todo=False  todo=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | a=1  b=2  c=3  d=4  e=5  f=0  if all ([(a<b) ,( b<c) ,( c<d ) ,(d<e)]):  print("todas son True")  print ([(a<b) ,( b<c) ,( c<d ) ,(d<e)])  else:  print ("al menos una condicion no es True")  print([(a<b) ,( b<c) ,( c<d ) ,( d<e )]) |   Salida esperada por consola   |  | | --- | | todas son True  [True, True, True, True] |   · |

**any()**

**Devuelve verdadero** si todas las condiciones o los elementos son verdaderos (o si el iterable está vacío).

**Devuelve False** si está vacío o si todos son falsos.

Todo se puede considerar como una secuencia de operaciones AND en los iterables proporcionados.

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | algun= any([True, True, True, True])  print (f"{algun=}")  #--------------------------------------------------------  algun= any([False, True, True, False])  print (f"{algun=}")  #--------------------------------------------------------  algun= any([False, False, False])  print (f"{algun=}")  #-------------------------------------------------------- |   Salida esperada por consola   |  | | --- | | algun=True  algun=True  algun=False |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | a=1  b=2  c=3  d=4  e=5  f=0  if any ([(a>b) ,( b<c) ,( c>d ) ,(d<e)]):  print("Alguno es True")  print ([(a>b) ,( b<c) ,( c>d ) ,(d<e)])  else:  print ("ninguno True")  print([(a>b) ,( b<c) ,( c>d ) ,( d<e )]) |   Salida esperada por consola   |  | | --- | | Alguno es True  [False, True, False, True] |   · |

·

|  |  |  |
| --- | --- | --- |
| Código Python   |  | | --- | | algun= any([True, True, True, True])  print (f"{algun=}")  a=1  b=2  c=3  d=4  e=5  f=0  if any ([(a>b) ,( b>c) ,( c>d ) ,( d>e ) ,(e<f)]):  print("Alguno es True")  print([(a>b) ,( b>c) ,( c>d ) ,( d>e ) ,(e<f)])  else:  print ("ninguno True")  print([(a>b) ,( b>c) ,( c>d ) ,( d>e ) ,(e<f)]) |   Salida esperada por consola   |  | | --- | | ninguno True  [False, False, False, False, False] |   · |

·

Estas funciones son especialmente útiles cuando necesitas verificar rápidamente si todos o al menos uno de los elementos de una secuencia cumplen una determinada condición.

Puedes utilizarlas en combinación con expresiones booleanas o con funciones lambda para realizar comprobaciones más complejas.

|  |
| --- |
| nota\_matematica = int(input ("que nota tenes en matemática:"))  nota\_lengua = int(input ("que nota tenes en lengua:"))  nota\_fisica = int(input ("que nota tenes en física:"))  nota\_quimica = int(input ("que nota tenes en química:"))  nota\_gimnasia = int(input ("que nota tenes en gimnasia:"))  nota\_dibujo = int(input ("que nota tenes en dibujo:"))  lista\_materias = ["nota\_matemática", "nota\_lengua", "nota\_física", "nota\_química",  "nota\_gimnasia", "nota\_dibujo"]  estado\_toda\_las\_materias = [ nota\_matematica >= 7,  nota\_lengua >= 7,  nota\_fisica >= 7,  nota\_quimica >= 7,  nota\_gimnasia >= 7,  nota\_dibujo >= 7 ]  for estado\_materia in estado\_toda\_las\_materias:  print (f"Estado materia aprobado {estado\_materia}")  for materia, estado\_materia in zip(lista\_materias, estado\_toda\_las\_materias):# zip  print (f"Estado de {materia} aprobado {estado\_materia}")  if False in estado\_toda\_las\_materias:  print ('"IN" al menos desaprobada una')  if False not in estado\_toda\_las\_materias:  print ('"NOT IN" ni una desaprobada una')  if all (estado\_toda\_las\_materias):  print ('"ALL" felicitaciones todo aprobado')  if any (estado\_toda\_las\_materias):  print ('"ANY" al menos aprobada una')  if not all (estado\_toda\_las\_materias):  print ('"NOT ALL" al menos desaprobada una')  print ('"NOT ALL" te quedo algo colgado para rendir')  if not any (estado\_toda\_las\_materias):  print ('"NOT ANY"todo mallllllll') |