**Módulo 6:**

|  |
| --- |
| Según la ley Argentina (SiMeLA - Sistema Métrico Legal Argentino) la parte entera se separa de la decimal por una coma (“,”) en programación esto no es válido el separador es el punto (“.”) |

|  |
| --- |
| objeto = 3,14159  ¿que tipo de dato es? |

|  |
| --- |
| objeto = 3,14159  print (f"{objeto=} es del tipo {type(objeto)}") |

Salida esperada por consola

|  |
| --- |
| objeto=(3, 14159) es del tipo <class 'tuple'> |

·

|  |
| --- |
| Es una tupla, una colección de dos (2) datos enteros.  objeto = 3,14159  |  La coma separa objetos en una colección  En una tupla los paréntesis son recomendados por la comunidad pero no obligatorios por el interprete Python  Una mejor manera de representar una tupla es la siguiente  objeto = (3,14159)  \/ \\_\_\_\_/  | |  | 2do dato 14159 tipo entero  1er dato 3 tipo entero |

**Colecciones :**

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.

Las colecciones más comunes son:

**tuple (Tuplas):**

|  |
| --- |
| **Son colecciones ordenadas , con index y inmutables de objetos separados por comas.**  Ordenadas  Con index  Inmutables  Separados por comas.  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:  **una tupla con objetos int**  nombre\_tupla = (1,2,3,4,5,6,7,8,9,0)  **una tupla con objetos float**  nombre\_tupla = (3.14159 , 2.99792458 , 4.5)  **una tupla con objetos str**  nombre\_tupla = ("Hola","Mundo","IT")  **una tupla con objetos bool**  nombre\_tupla = (True,False,True,False)  **una tupla con objetos mix**  nombre\_tupla = (1, 2, 3.14159, "Hola", True). |

·

**list (Listas):**

|  |
| --- |
| **Son colecciones ordenadas , con index y mutables de objetos separados por comas.**  Ordenadas  Con index  Mutables  Separados por comas.  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:  **una lista vacía**  nombre\_ lista = list()  o  nombre\_ lista = []  **una lista con objetos int**  nombre\_ lista = [1,2,3,4,5,6,7,8,9,0] |

·

**Set (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  Mutables  separados por comas.  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} |

·

**FrozenSet (Conjuntos):**

|  |
| --- |
| **Son colecciones desordenadas, sin index, únicos y inmutables de objetos separados por comas.**  Desordenadas  Sin index No tienen un index o indice.  Únicos No permiten duplicados  Inmutables  separados por comas.  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:  nombre\_frozenset = frozenset([])  **una frozenset con objetos int**  nombre\_frozenset = frozenset([1,2,3,4,5,6,7,8,9,0]) |

·

**Dictionary (Diccionarios):**

|  |
| --- |
| **Son colecciones de pares de objetos clave-valor // key - value**  **El primero es la clave o llave (key) - única**  **El segundo es el valor (value) - no única**  **ademas son desordenadas, sin index, únicos y inmutables de objetos separados por comas.**  Desordenadas  Sin index No tienen un index o indice.  keys  Únicos No permiten duplicados  Inmutables  Estos objetos que la componen pueden ser  (enteros (int), flotantes (float), tuplas (tuple) y frozenset.)  Separadores clave1 : valor , clave2 : valor , clave3 : valor  : dos puntos las claves se separan de los valores con dos puntos(:)  Values  No Únicos Si permiten duplicados  Mutables  Estos objetos que la componen pueden ser de cualquier tipo  (str, int, float, bool y…...colecciones listas,tuplas, diccionarios, sets, frozensets,etc)  Los pares clave-valor separados por comas.  **un dict vacío**  nombre\_dic = dict()  o  nombre\_dict = {}#<-------------------------ver mas adelante con set  **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  }  } |

·

|  |
| --- |
| Range (Rangos):  Si bien no es una colección, es una función que devuelve una secuencia inmutable de números. |

|  |
| --- |
| Range (Rangos):  Se utiliza principalmente en bucles para iterar sobre un rango de valores.  La sintaxis básica es la siguiente:  una range puede tener 3 parametros  nombre\_range = range(inicio, limite\_final, step)  nombre\_ range = range(5,50,2)  desde 5  hasta 50 no incluido 99 real  de 2 en 2  nombre\_ range = range(0,20,1)  nombre\_ range = range(0,20)  nombre\_ range = range(20)  **Devuelve una secuencia inmutable de números.** |

·

|  |
| --- |
| nombre\_range = range(5,50,2)  print (f"rango ={nombre\_range}")  print (f"lista ={list(nombre\_range)}") |

Salida esperada por consola

|  |
| --- |
| rango =range(5, 50, 2)  lista =[5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49] |

·

|  |
| --- |
| rango = range(0,20,1) # range(inicio, limite\_final, step)  print (f"rango (I,F,S) es {rango=}")  rango = range(0,20) # range(inicio, limite\_final,**~~step~~**)  # se omite el , step cuando es igual a 1 – None  print (f"rango (I,F) es {rango=}")  rango = range(20)# range(**~~inicio~~**, limite\_final, **~~step~~**)  # se omite el , inicio cuando es igual a 0 – None  print (f"rango (I) es {rango=}")  print (f"lista ={list(rango==}") |

Salida esperada por consola

|  |
| --- |
| rango (I,F,S) es rango=range(0, 20)  rango (I,F) es rango=range(0, 20)  rango (I) es rango=range(0, 20)  lista =[0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] |

·

|  |
| --- |
| Python también proporciona módulos adicionales en su biblioteca estándar que ofrecen colecciones más especializadas y eficientes.  Ejemplos:  **collections:** Este Módulo ofrece tipos de colecciones adicionales como deque (cola doblemente terminada), Counter (contador), OrderedDict (diccionario ordenado) y namedtuple (tupla con nombre), entre otros.  **array:** Este Módulo proporciona un tipo de colección llamado array, que es una estructura de datos eficiente para almacenar elementos del mismo tipo de datos en un arreglo.  **heapq:** Este Módulo implementa estructuras de datos de montículos (heaps) que se utilizan para mantener el orden en una colección de manera eficiente.  **queue:** Este Módulo proporciona implementaciones de diferentes tipos de colas, como Queue, LifoQueue (pila) y PriorityQueue (cola de prioridad).  Y debemos tener muy en cuenta los arrays de la librería **numpy (Num**erical **Py**thon**)** y los DataFrames de **pandas (Pan**el **Da**ta**s)** |

.

|  |
| --- |
| En física, existen varias constantes fundamentales que juegan un papel crucial en diversas teorías y ecuaciones. Aquí tienes algunos ejemplos de constantes importantes en física:  **¿que estructura de datos utilizarías?**  Velocidad de la luz en el vacío (c): Es la velocidad máxima a la que puede propagarse la luz en el vacío y es una constante fundamental en la teoría de la relatividad. Su valor aproximado es 299792458 metros por segundo.  Constante gravitacional (G): Es una constante que aparece en la ley de gravitación universal de Newton y determina la fuerza gravitatoria entre dos objetos masivos. Su valor aproximado es 6.67430 \* 10\*\*-11 m³/(kg·s²).  Carga elemental (e): Es la carga eléctrica fundamental de un electrón o un protón. Su valor es aproximadamente 1.602 \* 10\*\*-19 culombios.  Constante de Planck (h): Es una constante que está relacionada con la cuantización de la energía y se utiliza en la mecánica cuántica. Su valor es aproximadamente 6.62607015 \* 10\*\*-34 julios-segundo.  Constante de Boltzmann (k): Es una constante que relaciona la temperatura con la energía en la física estadística y termodinámica. Su valor es aproximadamente 1.380649 \* 10\*\*-23 julios por kelvin.  Número de Avogadro (NA): Es el número de átomos o moléculas en un mol de sustancia. Su valor aproximado es 6.022 \* 10\*\*23 moléculas por mol.  Masa del electrón (me): Es la masa de un electrón, una partícula subatómica con carga negativa. Su valor es aproximadamente 9.10938356 \* 10\*\*-31 kilogramos.  Masa del protón (mp): Es la masa de un protón, una partícula subatómica con carga positiva. Su valor es aproximadamente 1.67262192 \* 10\*\*-27 kilogramos.  Masa del neutrón (mn): Es la masa de un neutrón, una partícula subatómica sin carga eléctrica. Su valor es aproximadamente 1.674927471 \* 10\*\*-27 kilogramos.  Constante de la ley de Coulomb (k\_e): Es una constante que aparece en la ley de Coulomb, que describe la fuerza electrostática entre dos cargas eléctricas. Su valor es aproximadamente 8.9875517923 \* 10\*\*9 N m\*\*2/C\*\*2.  Constante de permeabilidad del vacío (μ0): Es una constante que está relacionada con la magnetostática y determina la fuerza magnética entre corrientes eléctricas. Su valor es aproximadamente 4π \* 10\*\*-7 T m/A.  Constante de la ley de Stefan-Boltzmann (σ): Es una constante que aparece en la ley de Stefan-Boltzmann, que relaciona la radiación emitida por un cuerpo negro con su temperatura. Su valor es aproximadamente 5.670374419 \* 10\*\*-8 W/(m\*\*2 K\*\*4). |

**Tuplas y listas:**

**Son colecciones ordenadas , con index y con objetos separados por comas.**

Ordenadas

Con index

Mutables **No tuplas**

**Si listas**

Separados por comas.

**Index**

En el contexto de una lista o tupla en Python, un índice se refiere a la posición numérica de un elemento dentro de la secuencia. Cada elemento en una lista o tupla tiene asignado un índice único que lo identifica iniciando en cero (0) y termina en el último N-1. El uso de len cuenta la cantidad de objetos desde 1 a N

En Python, los índices comienzan desde cero, lo que significa que el primer elemento de una lista o tupla tiene un índice de 0, el segundo elemento tiene un índice de 1, y así sucesivamente. Puedes acceder a un elemento específico de una lista o tupla utilizando su índice correspondiente.

Aquí tienes un ejemplo de cómo acceder a elementos individuales utilizando índices:

En el ejemplo anterior, se accede a elementos individuales de la lista mi\_lista y la tupla mi\_tupla utilizando corchetes [] seguidos del índice correspondiente.

Se debe tener en cuenta que los índices deben estar dentro del rango válido de la lista o tupla. Si intentas acceder a un índice que está fuera de ese rango, se producirá un error de índice fuera de rango (list index out of range).

·

|  |
| --- |
| lista = [1,2,3,"hola",99,3.14159,True, False]  print (f"el {lista =} es de la Módulo {type (lista)=}")  print (f"el {lista[0] =} es de la Módulo {type (lista[0])=}")  print (f"el {lista[1] =} es de la Módulo {type (lista[1])=}")  print (f"el {lista[2] =} es de la Módulo {type (lista[2])=}")  print (f"el {lista[3] =} es de la Módulo {type (lista[3])=}")  print (f"el {lista[4] =} es de la Módulo {type (lista[4])=}")  print (f"el {lista[5] =} es de la Módulo {type (lista[5])=}")  print (f"el {lista[6] =} es de la Módulo {type (lista[6])=}")  print (f"el {lista[7] =} es de la Módulo {type (lista[7])=}") |

Salida esperada por consola

|  |
| --- |
| el lista =[1, 2, 3, 'hola', 99, 3.14159, True, False] es de la Módulo type (lista)=<class 'list'>  el lista[0] =1 es de la Módulo type (lista[0])=<class 'int'>  el lista[1] =2 es de la Módulo type (lista[1])=<class 'int'>  el lista[2] =3 es de la Módulo type (lista[2])=<class 'int'>  el lista[3] ='hola' es de la Módulo type (lista[3])=<class 'str'>  el lista[4] =99 es de la Módulo type (lista[4])=<class 'int'>  el lista[5] =3.14159 es de la Módulo type (lista[5])=<class 'float'>  el lista[6] =True es de la Módulo type (lista[6])=<class 'bool'>  el lista[7] =False es de la Módulo type (lista[7])=<class 'bool'> |

·

**Métodos y atributos**

mi\_tupla

|  |
| --- |
| mi\_tupla = ("A","B","C","D","E","F")  print (f"el contenido de {mi\_tupla=}\t{type(mi\_tupla)}")  print (f"los métodos disponibles son {dir(mi\_tupla)=}") |

Salida esperada por consola

|  |
| --- |
| el contenido de mi\_tupla=('A', 'B', 'C', 'D', 'E', 'F') <class 'tuple'>  los métodos disponibles son dir(mi\_tupla)=['\_\_add\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'count', 'index'] |

mi\_ lista

|  |
| --- |
| mi\_lista = ["A","B","C","D","E","F"]  print (f"el contenido de {mi\_lista=}\t{type(mi\_lista)}")  print (f"los métodos disponibles son {dir(mi\_lista)=}") |

Salida esperada por consola

|  |
| --- |
| el contenido de mi\_lista=['A', 'B', 'C', 'D', 'E', 'F'] <class 'list'>  los métodos disponibles son dir(mi\_lista)=['\_\_add\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'] |

·

|  |
| --- |
| Métodos y atributos utilizables de forma simple son los que no poseen doble guion delante y detrás. Los dejaremos par mas adelante como los constructores de módulos |

.

|  |
| --- |
| Los métodos de las tuplas se incluyen dentro de los de las lista, por lo que solo es necesarios explicarlos una vez. |

.

En adelante veremos los metodos y atributos de las listas y tuplas

'append', 'clear', 'copy', **'count',** 'extend', **'index'**, 'insert', 'pop', 'remove', 'reverse', 'sort'

|  |
| --- |
| Len devuelve la cantidad de objetos que contienen en esta caso la coleccion |

**·**

**Separaremos los métodos por su acción:**

* **Agregar datos a la lista (las tuplas son inmutables)**
  + append() agrega un solo dato al final de la lista
  + insert() agrega un solo dato en el lugar que se indique por un index
  + extend() agrega una colección de datos al final de la lista original

|  |
| --- |
| lista = [1,2,3,"hola",99,3.14159,True, False]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  # agrego dato(s)  lista.append("chau")#al final  print (f"lista.append('chau'):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.insert(2,"dato insertado")# en el lugar solicitado  print (f"lista.insert(2,'dato insertado'):{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  # agrego una colección de datos  lista\_ext = ["A","B","C","D","E","F"]  lista.extend(lista\_ext)  print (f"lista.extend(lista\_ext):\n{lista}\n\tcantidad de elementos:{len(lista)}") |

Salida esperada por consola

|  |
| --- |
| el contenido de original lista=[1, 2, 3, 'hola', 99, 3.14159, True, False]  cantidad de elementos:8  lista.append('chau'):  [1, 2, 3, 'hola', 99, 3.14159, True, False,  **'chau'**]  cantidad de elementos:9  lista.insert(2,'dato insertado'):  [1, 2, **'dato insertado'**, 3, 'hola', 99, 3.14159, True, False,'chau']  cantidad de elementos:10  lista.extend(lista\_ext):  [1, 2, 'dato insertado', 3, 'hola', 99, 3.14159, True, False, 'chau', **'A', 'B', 'C', 'D', 'E', 'F'**]  cantidad de elementos:16 |

·

* **eliminar datos de una lista (las tuplas son inmutables)**
  + pop() elimina un solo dato ubicado en el index final de la lista
  + pop(I) elimina un solo dato ubicado en el index I de la lista
  + remove(D) elimina el primer valor D que encuentra desde index 0
  + clear() elimina todo el contenido de la colección. (La colección sigue existiendo)

|  |
| --- |
| lista = [1,2,3,"hola",99,3.14159,True, False]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  # elimino un dato  lista.pop()#por index () en el lugar solicitado ultimo  print (f"lista.pop():\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.pop(2)#por index (2) en el lugar solicitado  print (f"lista.pop(2):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.remove("hola")# por contenido  print (f"lista.remove('hola'):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.remove(2)# por contenido  print (f"lista.remove(2):\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.clear()# vació la lista  print (f"lista.clear ():\n{lista}\n\tcantidad de elementos:{len(lista)}") |

Salida esperada por consola

|  |
| --- |
| el contenido de original lista=[1, 2, 3, 'hola', 99, 3.14159, True, False]  cantidad de elementos:**8**  lista.pop():  [1, 2, 3, 'hola', 99, 3.14159, True]  cantidad de elementos:**7**  lista.pop(2):  [1, 2, 'hola', 99, 3.14159, True]  cantidad de elementos:**6**  lista.remove('hola'):  [1, 2, 99, 3.14159, True]  cantidad de elementos:**5**  lista.remove(2):  [1, 99, 3.14159, True]  cantidad de elementos:**4**  lista.clear ():  []  cantidad de elementos:**0** |

·

* **Organización de datos de una lista (las tuplas son inmutables)**
  + reverse() invierte el orden de la lista
  + sort() ordena la lista de mayor a menor
  + lista.sort(reverse=True) elimina el primer valor D que encuentra desde index 0

·

|  |
| --- |
| Tengan en cuenta que sort ordena de mayor a menor objetos numéricos y por cantidad de caracteres en objetos string.  Si se utiliza listas con objetos de ambos tipos se obtendrá un error |

·

|  |
| --- |
| lista = [1,2,3,4,5,6,7,8,9,8,7,6,5,6,7,8,9,11,2]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.reverse()  print (f"lista.reverse:\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.sort()  print (f"lista sort:\n{lista}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  lista.sort(reverse=True)  print (f"lista sort(reverse=True):\n{lista}\n\tcantidad de elementos:{len(lista)}") |

Salida esperada por consola

|  |
| --- |
| el contenido de original lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 6, 7, 8, 9, 11, 2]  cantidad de elementos:19  lista.reverse:  [2, 11, 9, 8, 7, 6, 5, 6, 7, 8, 9, 8, 7, 6, 5, 4, 3, 2, 1]  cantidad de elementos:19  lista sort:  [1, 2, 2, 3, 4, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 11]  cantidad de elementos:19  lista sort(reverse=True):  [11, 9, 9, 8, 8, 8, 7, 7, 7, 6, 6, 6, 5, 5, 4, 3, 2, 2, 1]  cantidad de elementos:19 |

·

* **Información sobre el contenido de datos de una lista o una tupla.**

**Estos Atributos no modifican el contenido y son usados en ambas colecciones**

* + count() invierte el orden de la lista
  + index() ordena la lista de mayor a menor

|  |
| --- |
| lista = [1,2,3,4,5,6,7,8,9,8,7,6,5,6,7,8,9,11,2]  print (f"el contenido de original {lista=}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  print (f"lista.count(9) esta {lista.count(9)} veces en la lista.\n\tcantidad de elementos:{len(lista)}")  print (f"lista.count(11) esta {lista.count(11)} veces en la lista.\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------  print (f"lista.index(3) esta en el index :{lista.index(3)}\n\tcantidad de elementos:{len(lista)}")  #------------------------------------------ |

Salida esperada por consola

|  |
| --- |
| el contenido de original lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 8, 7, 6, 5, 6, 7, 8, 9, 11, 2]  cantidad de elementos:19  lista.count(9) esta 2 veces en la lista.  cantidad de elementos:19  lista.count(11) esta 1 veces en la lista.  cantidad de elementos:19  lista.index(3) esta en el index :2  cantidad de elementos:19 |

·

**Slicing**

El slicing en Python es una técnica que te permite extraer una porción (subsecuencia) de una secuencia, como una cadena, una lista o una tupla. Permite seleccionar un rango de elementos utilizando un inicio, un fin y un paso.

La sintaxis básica del slicing es la siguiente:

secuencia[inicio:limite\_final]

inicio: El índice donde comienza el slice que se necesitar (este inice si se incluye).

limite\_final: El índice limite final donde termina la porción que deseas extraer.

Al ser un limite, este valor nunca se llega, por lo que no se incluye en el slice que se necesita.

secuencia[inicio:limite\_final:step]

Step o paso (opcional): El incremento entre los índices seleccionados. Por defecto es 1.

Aquí tienes algunos ejemplos para ilustrar cómo funciona el slicing:

|  |
| --- |
| cadena = "Python-es-Genial"  print(f"|{cadena[0:6]}| Extrae los caracteres desde el índice 0 al 6 (5)") # |Python|  print(f"|{cadena[ :6]}| Extrae los caracteres desde el inicio al 6 (5)")# |Python|  print(f"|{cadena[ :-10]}| Extrae los caracteres desde el inicio al -10")# |Python|  print(f"|{cadena[6:16]}| Extrae los caracteres desde el índice 7 hasta el 16")# |-es-Genial|  print(f"|{cadena[6:]}| Extrae los caracteres desde el índice 7 hasta el final")# |-es-Genial|  print(f"|{cadena[-10:]}| Extrae los caracteres desde el índice -10 hasta el final")# |-es-Genial|  print(f"|{cadena[6:10]}| Extrae los caracteres desde el inicio 6 hasta el 10 (9)")# |-es-|  print(f"|{cadena[-10:-6]}| Extrae los caracteres desde el inicio -10 hasta el-6(-7)")# |-es-|  print(f"|{cadena[::2]}| Extrae los caracteres con un paso de 2")# |Pto-sGna|  print(f"|{cadena[::-1]}| Invierte el orden") # | laineG-se-nohtyP| |

Salida esperada por consola

|  |
| --- |
| |Python| Extrae los caracteres desde el índice 0 al 6 (5)  |Python| Extrae los caracteres desde el inicio al 6 (5)  |Python| Extrae los caracteres desde el inicio al -10  |-es-Genial| Extrae los caracteres desde el índice 7 hasta el 16  |-es-Genial| Extrae los caracteres desde el índice 7 hasta el final  |-es-Genial| Extrae los caracteres desde el índice -10 hasta el final  |-es-| Extrae los caracteres desde el inicio 6 hasta el 10 (9)  |-es-| Extrae los caracteres desde el inicio -10 hasta el-6(-7)  |Pto-sGna| Extrae los caracteres con un paso de 2  |laineG-se-nohtyP| Invierte el orden |

·

Recuerda que el índice de inicio y fin pueden ser números negativos, lo que indica que se cuentan desde el final de la secuencia.

El slicing en Python es una herramienta muy útil para manipular secuencias y extraer partes específicas según tus necesidades.

* **Pilas y colas**

|  |
| --- |
| FIFO First In > Fist Out  LIFO Last In > First Out –o- First In > Last Out |

·

|  |
| --- |
| lista = [1,2,3,4,5,6,7,8,9,10,11,112,13,14,15]  print ("append()")  lista.append(16)  print(f"{lista=}")  lista.append(17)  print(f"{lista=}")  lista.append(18)  print(f"{lista=}")  lista.append(19)  print(f"{lista=}")  lista.append(20)  print(f"{lista=}")  #-----------------------------  print ("pop() fin")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  lista.pop()  print(f"{lista=}")  #-----------------------------  print ("pop(0) inicio")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}")  lista.pop(0)  print(f"{lista=}") |

Salida esperada por consola

|  |
| --- |
| append()  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18, 19]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18, 19, 20]  pop() fin  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18, 19]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17, 18]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16, 17]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15, 16]  lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  pop(0) inicio  lista=[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[3, 4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[4, 5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[5, 6, 7, 8, 9, 10, 11, 112, 13, 14, 15]  lista=[6, 7, 8, 9, 10, 11, 112, 13, 14, 15] |

**Modificación de flujo Bucles: for:**

for dato in coleccion :

bloque

de

iteración

* + break
  + continue

anidación

* + enumerate
  + zip

|  |
| --- |
| En Python la función solo recorre objetos iterables.  Es un for each en otros lenguajes |

·

**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.

·

|  |
| --- |
| lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  for numero in lista:  print (f"el valor de {numero =}") |

Salida esperada por consola

|  |
| --- |
| el valor de numero =1  el valor de numero =2  el valor de numero =3  el valor de numero =4  el valor de numero =5  el valor de numero =6  el valor de numero =7  el valor de numero =8  el valor de numero =9  el valor de numero =10 |

* **break**

|  |
| --- |
| lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  for numero in lista:  print (f"el valor de {numero =}")  if numero == 6:  print("break")  break |

Salida esperada por consola

|  |
| --- |
| el valor de numero =1  el valor de numero =2  el valor de numero =3  el valor de numero =4  el valor de numero =5  el valor de numero =6  break |

* **continue**·

|  |
| --- |
| lista=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  for numero in lista:  print (f"el valor de {numero =}")  if numero%2 == 0:  print("\t\tcontinue x par")  continue  print("\timpar") |

Salida esperada por consola

|  |
| --- |
| el valor de numero =1  impar  el valor de numero =2  continue x par  el valor de numero =3  impar  el valor de numero =4  continue x par  el valor de numero =5  impar  el valor de numero =6  continue x par  el valor de numero =7  impar  el valor de numero =8  continue x par  el valor de numero =9  impar  el valor de numero =10  continue x par |

* **Anidación**

|  |
| --- |
| rango\_ext=range(0,10,1)  rango\_int=range(0,10,1)  for externo in rango\_ext:  for interno in rango\_int:  print (f"{externo} - {interno}", end=" | ")  print() |

Salida esperada por consola

|  |
| --- |
| 0 - 0 | 0 - 1 | 0 - 2 | 0 - 3 | 0 - 4 | 0 - 5 | 0 - 6 | 0 - 7 | 0 - 8 | 0 - 9 |  1 - 0 | 1 - 1 | 1 - 2 | 1 - 3 | 1 - 4 | 1 - 5 | 1 - 6 | 1 - 7 | 1 - 8 | 1 - 9 |  2 - 0 | 2 - 1 | 2 - 2 | 2 - 3 | 2 - 4 | 2 - 5 | 2 - 6 | 2 - 7 | 2 - 8 | 2 - 9 |  3 - 0 | 3 - 1 | 3 - 2 | 3 - 3 | 3 - 4 | 3 - 5 | 3 - 6 | 3 - 7 | 3 - 8 | 3 - 9 |  4 - 0 | 4 - 1 | 4 - 2 | 4 - 3 | 4 - 4 | 4 - 5 | 4 - 6 | 4 - 7 | 4 - 8 | 4 - 9 |  5 - 0 | 5 - 1 | 5 - 2 | 5 - 3 | 5 - 4 | 5 - 5 | 5 - 6 | 5 - 7 | 5 - 8 | 5 - 9 |  6 - 0 | 6 - 1 | 6 - 2 | 6 - 3 | 6 - 4 | 6 - 5 | 6 - 6 | 6 - 7 | 6 - 8 | 6 - 9 |  7 - 0 | 7 - 1 | 7 - 2 | 7 - 3 | 7 - 4 | 7 - 5 | 7 - 6 | 7 - 7 | 7 - 8 | 7 - 9 |  8 - 0 | 8 - 1 | 8 - 2 | 8 - 3 | 8 - 4 | 8 - 5 | 8 - 6 | 8 - 7 | 8 - 8 | 8 - 9 |  9 - 0 | 9 - 1 | 9 - 2 | 9 - 3 | 9 - 4 | 9 - 5 | 9 - 6 | 9 - 7 | 9 - 8 | 9 - 9 | |

* **enumerate**

Enumerate es una función que se utiliza para iterar sobre una secuencia (como una lista, una tupla o una cadena) al mismo tiempo que se obtienen tanto el valor como su índice correspondiente. Proporciona una forma conveniente de realizar un seguimiento del índice mientras se recorre una secuencia.

|  |
| --- |
| lista = ["A","B","C","D","E","F"]  for index, caracter in enumerate(lista):  print (f"el valor de {caracter =} se encuentra en el index {index}") |

Salida esperada por consola

|  |
| --- |
| el valor de caracter ='A' se encuentra en el index 0  el valor de caracter ='B' se encuentra en el index 1  el valor de caracter ='C' se encuentra en el index 2  el valor de caracter ='D' se encuentra en el index 3  el valor de caracter ='E' se encuentra en el index 4  el valor de caracter ='F' se encuentra en el index 5 |

* **zip**

Zip es una función que se utiliza para combinar elementos de varias secuencias en una sola secuencia de tuplas. Toma como argumentos una o más secuencias y devuelve un iterador que produce tuplas donde cada tupla contiene los elementos correspondientes de las secuencias de entrada.

|  |
| --- |
| lista\_c= ["A","B","C","D","E","F"]  lista\_n= [1, 2, 3, 4, 5, 6]  for caracter, numero in zip(lista\_c,lista\_n,):  print (f"{caracter =} en la 'lista\_c' se encuentra en la misma posición (index) que {numero} en la 'lista\_n'") |

Salida esperada por consola

|  |
| --- |
| caracter ='A' en la 'lista\_c' se encuentra en la misma posición (index) que 1 en la 'lista\_n'  caracter ='B' en la 'lista\_c' se encuentra en la misma posición (index) que 2 en la 'lista\_n'  caracter ='C' en la 'lista\_c' se encuentra en la misma posición (index) que 3 en la 'lista\_n'  caracter ='D' en la 'lista\_c' se encuentra en la misma posición (index) que 4 en la 'lista\_n'  caracter ='E' en la 'lista\_c' se encuentra en la misma posición (index) que 5 en la 'lista\_n'  caracter ='F' en la 'lista\_c' se encuentra en la misma posición (index) que 6 en la 'lista\_n' |

·