

**GUÍA DE LABORATORIO 05****Tipos de Datos de Colección Listas, Tuplas, Conjuntos y Diccionarios**

Asignatura	Datos del alumno	Fecha y Firma
Algoritmos y solución de problemas	Apellidos:	
	Nombre:	

**Instrucciones:**

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

**1. Objetivos:**

- Escribir algoritmos y codificar sobre los tipos de Datos de Colección.
- Solucionar los ejercicios propuestos

**2. Equipos, herramientas o materiales**

- Computador
- Software: Python
- Algoritmos

**3. Fundamento teórico****3.1. Conceptos Clave Tipos de Datos de Colección**

- **Lista:** es una colección que está ordenada y es modificable. Permite miembros duplicados.
- **Tupla:** es una colección que está ordenada y es inmutable. Permite miembros duplicados.
- **Conjunto:** es una colección que es desordenada, sin índice y no modificable, pero podemos añadir nuevos elementos al conjunto. No se permiten miembros duplicados.
- **Diccionario:** es una colección que es desordenada, modificable e indexada. No se permiten miembros duplicados.

**3.2. Listas**

Una lista es una colección de diferentes tipos de datos que está ordenada y es modificable (mutable). Una lista puede estar vacía o puede tener elementos de diferentes tipos de datos.

**Como Crear una Lista**

En Python podemos crear listas de dos maneras:

- Usando la función incorporada list.

```
1 lst = list()
2 empty_list = list() # esta es una lista vacía, no hay ningún elemento en la lista
3 print(len(empty_list)) # 0
```

- Usando corchetes, []

```
5 lst = []
6 empty_list = [] # esta es una lista vacía, no hay ningún elemento en la lista
7 print(len(empty_list)) # 0
```



Las listas pueden tener elementos de diferentes tipos de datos

```
19 lst = ['Asabeneh', 250, True, {'pais': 'Finlandia', 'ciudad': 'Helsinki'}] # Lista que contiene diferentes tipos de datos
```

Listas con valores iniciales. Usamos len() para encontrar la longitud de una lista.

```
1 frutas = ['banana', 'naranja', 'mango', 'limón'] # Lista de frutas
2 vegetales = ['Tomate', 'Papa', 'Col', 'Cebolla', 'Zanahoria'] # Lista de vegetales
3 productos_animales = ['leche', 'carne', 'mantequilla', 'yogur'] # Lista de productos animales
4 tecnologias_web = ['HTML', 'CSS', 'JS', 'React', 'Redux', 'Node', 'MongoDB'] # Lista de tecnologías web
5 paises = ['Finlandia', 'Estonia', 'Dinamarca', 'Suecia', 'Noruega']
6
7 #Imprime las listas y su longitud
8 print('Frutas:', frutas)
9 print('Número de frutas:', len(frutas))
10 print('Vegetales:', vegetales)
11 print('Número de vegetales:', len(vegetales))
12 print('Productos animales:', productos_animales)
13 print('Número de productos animales:', len(productos_animales))
14 print('Tecnologías web:', tecnologias_web)
15 print('Número de tecnologías web:', len(tecnologias_web))
16 print('Países:', paises)
17 print('Número de países:', len(paises))
```

### Accediendo a los Elementos de la Lista Mediante Indexación Positiva

Accedemos a cada elemento de una lista utilizando su índice. Un índice de lista comienza desde 0. La imagen a continuación muestra claramente donde comienza el índice de la lista

['banana',	'Naranja',	'Mango',	'Limón']
0	1	2	3

```
1 frutas = ['banana', 'naranja', 'mango', 'limón']
2 primera_fruta = frutas[0] # estamos accediendo al primer elemento usando su índice
3 print(primera_fruta) # banana
4 segunda_fruta = frutas[1]
5 print(segunda_fruta) # naranja
6 última_fruta = frutas[3]
7 print(última_fruta) # limón
8
9 #Último índice
10 último_índice = len(frutas) - 1
11 última_fruta = frutas[último_índice]
```

### Accediendo a los Elementos de la Lista Mediante Indexación Negativa

La indexación negativa significa comenzar desde el final, -1 se refiere al último elemento, -2 se refiere al penúltimo elemento.

['banana',	'Naranja',	'Mango',	'Limón']
-4	-3	-2	-1

```
1 frutas = ['banana', 'naranja', 'mango', 'limón']
2 primera_fruta = frutas[-4]
3 última_fruta = frutas[-1]
4 segunda_última = frutas[-2]
5 print(primera_fruta) # banana
6 print(última_fruta) # limón
7 print(segunda_última) # mango
```



### Desempaquetar los Elementos de una Lista

```
1 lst = ['elemento1', 'elemento2', 'elemento3', 'elemento4', 'elemento5']
2 primer_elemento, segundo_elemento, tercer_elemento, *resto = lst
3 print(primer_elemento) # elemento1
4 print(segundo_elemento) # elemento2
5 print(tercer_elemento) # elemento3
6 print(resto) # ['elemento4', 'elemento5']
```

### Cortar Elementos de una Lista

Indexación positiva: Podemos especificar un rango de índices positivos especificando el inicio, el final y el paso, el valor de retorno será una nueva lista. (valores predeterminados para inicio = 0, final = len(lst) - 1 (último elemento), paso = 1)

```
1 frutas = ['banana', 'naranja', 'mango', 'limón']
2 todas_las_frutas = frutas[0:4] # devuelve todas las frutas
3
4 #esto también dará el mismo resultado que el anterior
5 todas_las_frutas = frutas[0:] # si no indicamos donde parar toma todos los demás
6 naranja_y_mango = frutas[1:3] # no incluye el primer índice
7 naranja_mango_limón = frutas[1:]
8 naranja_y_limón = frutas[::2] # aquí usamos un tercer argumento, paso. Tomará cada segundo elemento - ['banana', 'mango']
```

Indexación negativa:

```
1 frutas = ['banana', 'naranja', 'mango', 'limón']
2 todas_las_frutas = frutas[-4:] # devuelve todas las frutas
3 naranja_y_mango = frutas[-3:-1] #
4 naranja_mango_limón = frutas[-3:]
5 naranja_y_limón = frutas[::-1]
```

### Modificando Listas

La lista es una colección ordenada mutable o modificable. Vamos a modificar la lista de frutas.

```
1 frutas = ['banana', 'naranja', 'mango', 'limón']
2 frutas[0] = 'aguacate'
3 print(frutas) # ['aguacate', 'naranja', 'mango', 'limón']
4 frutas[1] = 'manzana'
5 print(frutas) # ['aguacate', 'manzana', 'mango', 'limón']
6 último_indice = len(frutas) - 1
7 frutas[último_indice] = 'lima'
8 print(frutas) # ['aguacate', 'manzana', 'mango', 'lima']
```

### Comprobando Elementos en una Lista

Comprobando un elemento si es miembro de una lista usando el operador in. Mira el ejemplo de abajo.

```
1 frutas = ['banana', 'naranja', 'mango', 'limón']
2 existe = 'banana' in frutas
3 print(existe) # True
4 existe = 'lima' in frutas
5 print(existe) # False
```

### Añadiendo Elementos a una Lista

Para agregar un elemento al final de una lista existente usamos el método append().



```
1 #sintaxis
2 lst = list()
3 lst.append('elemento')
4
5 frutas = ['banana', 'naranja', 'mango', 'limón']
6 frutas.append('manzana')
7 print(frutas) # ['banana', 'naranja', 'mango', 'limón', 'manzana']
8 frutas.append('lima') # ['banana', 'naranja', 'mango', 'limón', 'manzana', 'lima']
9 print(frutas)
```

### **Insertando Elementos en una Lista**

Podemos usar el método `insert()` para insertar un solo elemento en un índice especificado en una lista. Ten en cuenta que otros elementos se desplazan a la derecha. El método `insert()` toma dos argumentos: índice y un elemento a insertar.

```
1 #Sintaxis
2 lst = ['elemento1', 'elemento2']
3 lst.insert(indice, elemento)
4
5 frutas = ['banana', 'naranja', 'mango', 'limón']
6 frutas.insert(2, 'manzana') # inserta manzana entre naranja y mango
7 print(frutas) # ['banana', 'naranja', 'manzana', 'mango', 'limón']
8 frutas.insert(3, 'lima') # ['banana', 'naranja', 'manzana', 'lima', 'mango', 'limón']
9 print(frutas)
```

### **Eliminando Elementos de una Lista**

El método `remove` elimina un elemento especificado de una lista

```
1 #sintaxis
2 lst = ['elemento1', 'elemento2']
3 lst.remove(elemento)
4
5 frutas = ['banana', 'naranja', 'mango', 'limón', 'banana']
6 frutas.remove('banana')
7 print(frutas) # ['naranja', 'mango', 'limón', 'banana'] - este método elimina la primera ocurrencia del elemento
8 frutas.remove('limón')
9 print(frutas) # ['naranja', 'mango', 'banana']
```

### **Eliminando Elementos Usando Pop**

El método `pop()` elimina el índice especificado, (o el último elemento si no se especifica el índice):

```
1 #sintaxis
2 lst = ['item1', 'item2']
3 lst.pop() # último elemento
4 lst.pop(indice)
5
6 frutas = ['banana', 'naranja', 'mango', 'limón']
7 frutas.pop()
8 print(frutas) # ['banana', 'naranja', 'mango']
9
10 frutas.pop(0)
11 print(frutas) # ['naranja', 'mango']
```

### **Eliminando Elementos Usando Del**

La palabra clave `del` elimina el índice especificado y también puede usarse para eliminar elementos dentro de un rango de índices. También puede eliminar completamente la lista.





```
1 #sintaxis
2 lst = ['item1', 'item2']
3 del lst[indice] # sólo un elemento
4 del lst # para eliminar la lista completamente
5
6 frutas = ['banana', 'naranja', 'mango', 'limón', 'kiwi', 'lima']
7 del frutas[0]
8 print(frutas) # ['naranja', 'mango', 'limón', 'kiwi', 'lima']
9 del frutas[1]
10 print(frutas) # ['naranja', 'limón', 'kiwi', 'lima']
11 del frutas[1:3] # esto elimina los elementos entre los índices dados, así que no elimina el elemento con índice 3!
12 print(frutas) # ['naranja', 'lima']
13 del frutas
14 print(frutas) # Esto debería dar: NameError: name 'frutas' is not defined
```

### Limpiando Elementos de la Lista

El método `clear()` vacía la lista:

```
1 #sintaxis
2 lst = ['item1', 'item2']
3 lst.clear()
4
5 frutas = ['banana', 'naranja', 'mango', 'limón']
6 frutas.clear()
7 print(frutas) # []
```

### Copiando una Lista

Es posible copiar una lista reasignándola a una nueva variable de la siguiente manera: `lista2 = lista1`. Ahora, `lista2` es una referencia de `lista1`, cualquier cambio que hagamos en `lista2` también modificará la original, `lista1`. Pero hay muchos casos en los que no nos gusta modificar la original, en su lugar nos gusta tener una copia diferente. Una forma de evitar el problema anterior es usar `copy()`.


```
1 #sintaxis
2 lst = ['item1', 'item2']
3 lst_copia = lst.copy()
4
5 frutas = ['banana', 'naranja', 'mango', 'limón']
6 frutas_copia = frutas.copy()
7 print(frutas_copia) # ['banana', 'naranja', 'mango', 'limón']
```

### Uniendo Listas

Hay varias formas de unir, o concatenar, dos o más listas en Python.

 Operador Plus (+)

```
1 #sintaxis
2 lista3 = lista1 + lista2
3
4 numeros_positivos = [1, 2, 3, 4, 5]
5 cero = [0]
6 numeros_negativos = [-5, -4, -3, -2, -1]
7 enteros = numeros_negativos + cero + numeros_positivos
8 print(enteros) # [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
9 frutas = ['banana', 'naranja', 'mango', 'limón']
10 vegetales = ['Tomate', 'Patata', 'Repollo', 'Cebolla', 'Zanahoria']
11 frutas_y_vegetales = frutas + vegetales
12 print(frutas_y_vegetales) # ['banana', 'naranja', 'mango', 'limón', 'Tomate', 'Patata', 'Repollo', 'Cebolla', 'Zanahoria']
```

 Uniendo usando el método `extend()` El método `extend()` permite agregar una lista en una lista. Ver el ejemplo a continuación.



```
1 #sintaxis
2 lista1 = ['item1', 'item2']
3 lista2 = ['item3', 'item4', 'item5']
4 lista1.extend(lista2)
5
6 num1 = [0, 1, 2, 3]
7 num2 = [4, 5, 6]
8 num1.extend(num2)
9 print('Números:', num1) # Números: [0, 1, 2, 3, 4, 5, 6]
10 numeros_negativos = [-5, -4, -3, -2, -1]
11 numeros_positivos = [1, 2, 3, 4, 5]
12 cero = [0]
13
14 numeros_negativos.extend(cero)
15 numeros_negativos.extend(numeros_positivos)
16 print('Enteros:', numeros_negativos) # Enteros: [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
17 frutas = ['banana', 'naranja', 'mango', 'limón']
18 vegetales = ['Tomate', 'Patata', 'Repollo', 'Cebolla', 'Zanahoria']
19 frutas.extend(vegetales)
20 print('Frutas y vegetales:', frutas) # Frutas y vegetales: ['banana', 'naranja', 'mango', 'limón', 'Tomate', 'Patata', 'Repollo', 'Cebolla', 'Zanahoria']
```

### Contando Elementos en una Lista

El método `count()` devuelve el número de veces que un elemento aparece en una lista:

```
1 #sintaxis
2 lst = ['item1', 'item2']
3 lst.count(item)
4
5 frutas = ['banana', 'naranja', 'mango', 'limón']
6 print(frutas.count('naranja')) # 1
7 edades = [22, 19, 24, 25, 26, 24, 25, 24]
8 print(edades.count(24)) # 3
```

### Encontrando el Índice de un Elemento

El método `index()` devuelve el índice de un elemento en la lista:

```
1 # sintaxis
2 lst = ['item1', 'item2']
3 lst.index(item)
4
5 frutas = ['banana', 'naranja', 'mango', 'limón']
6 print(frutas.index('naranja')) # 1
7 edades = [22, 19, 24, 25, 26, 24, 25, 24]
8 print(edades.index(24)) # 2, la primera aparición
```

### Invertir una lista


El método `reverse()` invierte el orden de una lista.

```
1 #sintaxis
2 lst = ['elemento1', 'elemento2']
3 lst.reverse()
4
5 frutas = ['plátano', 'naranja', 'mango', 'limón']
6 frutas.reverse()
7 print(frutas) # ['Limón', 'mango', 'naranja', 'plátano']
8 edades = [22, 19, 24, 25, 26, 24, 25, 24]
9 edades.reverse()
10 print(edades) # [24, 25, 24, 26, 25, 24, 19, 22]
```

### Ordenando elementos de una lista

Para ordenar listas podemos usar el método `sort()` o la función incorporada `sorted()`. El método `sort()` reordena los elementos de la lista en orden ascendente y modifica la lista original. Si un argumento del método `sort()` es `reverse` igual a `true`, organizará la lista en orden descendente.



 `sort()`: este método modifica la lista original




```
1 #sintaxis
2 lst = ['elemento1', 'elemento2']
3 lst.sort() # ascendente
4 lst.sort(reverse=True) # descendente
5
6 #Ejemplo:
7
8 frutas = ['plátano', 'naranja', 'mango', 'limón']
9 frutas.sort()
10 print(frutas) # ordenado alfabéticamente, ['banana', 'limón', 'mango', 'naranja']
11 frutas.sort(reverse=True)
12 print(frutas) # ['naranja', 'mango', 'limón', 'plátano']
13 edades = [22, 19, 24, 25, 26, 24, 25, 24]
14 edades.sort()
15 print(edades) # [19, 22, 24, 24, 24, 25, 25, 26]
16
17 edades.sort(reverse=True)
18 print(edades) # [26, 25, 25, 24, 24, 24, 22, 19]
```

 `sorted()`: devuelve la lista ordenada sin modificar la lista original Ejemplo:

```
1 frutas = ['plátano', 'naranja', 'mango', 'limón']
2 print(sorted(frutas)) # ['banana', 'limón', 'mango', 'naranja']
3
4 #Orden inverso
5 frutas = ['plátano', 'naranja', 'mango', 'limón']
6 frutas = sorted(frutas, reverse=True)
7 print(frutas) # ['naranja', 'mango', 'limón', 'plátano']
```

### 3.3. Tuplas

Una tupla es una colección de diferentes tipos de datos que es ordenada e inmutable. Las tuplas se escriben con paréntesis redondos, (). Una vez que se crea una tupla, no podemos cambiar sus valores. No podemos usar métodos de agregar, insertar, remover en una tupla porque no es modificable (mutable). A diferencia de la lista, la tupla tiene pocos métodos. Métodos relacionados con tuplas:

-  `tuple()`: para crear una tupla vacía
-  `count()`: para contar el número de un ítem específico en una tupla
-  `index()`: para encontrar el índice de un ítem específico en una tupla
  - o operador: para unir dos o más tuplas y crear una nueva tupla

#### Creando una Tupla

 Tupla vacía: Creando una tupla vacía

```
1 #sintaxis
2 tupla_vacia = ()
3
4 #o usando el constructor de tupla
5 tupla_vacia = tuple()
```

 Tupla con valores iniciales

```
1 #sintaxis
2 tp1 = ('elemento1', 'elemento2', 'elemento3')
3 frutas = ('plátano', 'naranja', 'mango', 'limón')
```

 Longitud de una tupla

Usamos el método `len()` para obtener la longitud de una tupla.



```

1 # sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3')
3 len(tpl)

```

### Accediendo a los elementos de una tupla

- Indexación positiva Similar al tipo de datos de lista, usamos indexación positiva o negativa para acceder a los elementos de una tupla.

['Platano',	'Naranja',	'Mango',	'Limón']
0	1	2	3

Accediendo a los elementos de una tupla

```

1 #Sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3')
3 primer_elemento = tpl[0]
4 segundo_elemento = tpl[1]
5 frutas = ('plátano', 'naranja', 'mango', 'limón')
6 primera_fruta = frutas[0]
7 segunda_fruta = frutas[1]
8 ultimo_indice = len(frutas) - 1
9 ultima_fruta = frutas[ultimo_indice]

```

- Indexación negativa La indexación negativa significa comenzar desde el final, -1 se refiere al último elemento, -2 se refiere al penúltimo y el negativo de la longitud de la lista/tupla se refiere al primer elemento.

['Platano',	'Naranja',	'Mango',	'Limón']
-4	-3	-2	-1

Indexación negativa de una tupla

```

1 #Sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3', 'elemento4')
3 primer_elemento = tpl[-4]
4 segundo_elemento = tpl[-3]
5 frutas = ('plátano', 'naranja', 'mango', 'limón')
6 primera_fruta = frutas[-4]
7 segunda_fruta = frutas[-3]
8 ultima_fruta = frutas[-1]

```

### Cortando Tuplas

Podemos cortar una sub-tupla especificando un rango de índices donde comenzar y donde terminar en la tupla, el valor devuelto será una nueva tupla con los elementos especificados.

- Rango de índices positivos

```

1 #Sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3', 'elemento4')
3 todos_los_elementos = tpl[0:4] # todos los elementos
4 todos_los_elementos = tpl[0:] # todos los elementos
5 dos_elementos_intermedios = tpl[1:3] # no incluye el elemento en el índice 3
6 frutas = ('plátano', 'naranja', 'mango', 'limón')
7 todas_las_frutas = frutas[0:4] # todos los elementos
8 todas_las_frutas = frutas[0:] # todos los elementos
9 naranja_mango = frutas[1:3] # no incluye el elemento en el índice 3
10 naranja_y_resto = frutas[1:]

```





### Rango de índices negativos

```
1 #Sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3','elemento4')
3 todos_los_elementos = tpl[-4:] # todos los elementos
4 dos_elementos_intermedios = tpl[-3:-1] # no incluye el elemento en el índice -1
5 frutas = ('plátano', 'naranja', 'mango', 'limón')
6 todas_las_frutas = frutas[-4:] # todos los elementos
7 naranja_mango = frutas[-3:-1] # no incluye el elemento en el índice 3
8 naranja_y_resto = frutas[-3:]
```

### **Cambiando Tuplas a Listas**

Podemos cambiar tuplas a listas y listas a tuplas. La tupla es inmutable, si queremos modificar una tupla deberíamos cambiarla a una lista.

```
1 #Sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3','elemento4')
3 lst = list(tpl)
4 frutas = ('plátano', 'naranja', 'mango', 'limón')
5 frutas = list(frutas)
6 frutas[0] = 'manzana'
7 print(frutas) # ['manzana', 'naranja', 'mango', 'limón']
8 frutas = tuple(frutas)
9 print(frutas) # ('manzana', 'naranja', 'mango', 'limón')
```

### **Verificando un Elemento en una Tupla**

Podemos verificar si un elemento existe o no en una tupla usando `in`, devuelve un booleano.

```
1 # Sintaxis
2 tpl = ('elemento1', 'elemento2', 'elemento3','elemento4')
3 'elemento2' in tpl # Verdadero
4 frutas = ('plátano', 'naranja', 'mango', 'limón')
5 print('naranja' in frutas) # Verdadero
6 print('manzana' in frutas) # Falso
7 frutas[0] = 'manzana' # Error de tipo: el objeto 'tupla' no soporta la asignación de elementos
```

### **Uniendo Tuplas**

Podemos unir dos o más tuplas usando el operador `+`

```
1 # sintaxis
2 tpl1 = ('elemento1', 'elemento2', 'elemento3')
3 tpl2 = ('elemento4', 'elemento5','elemento6')
4 tpl3 = tpl1 + tpl2
5
6 frutas = ('plátano', 'naranja', 'mango', 'limón')
7 verduras = ('Tomate', 'Patata', 'Repollo','Cebolla', 'Zanahoria')
8 frutas_y_verduras = frutas + verduras
```

### **Borrando Tuplas**

No es posible eliminar un solo elemento en una tupla, pero es posible eliminar la tupla en sí usando `del`.

```
1 #sintaxis
2 tpl1 = ('elemento1', 'elemento2', 'elemento3')
3 del tpl1
4
5 frutas = ('plátano', 'naranja', 'mango', 'limón')
6 del frutas
```




### 3.4. Conjuntos


Un conjunto es una colección de elementos. Permíteme llevarte de vuelta a tu lección de matemáticas de la escuela primaria o secundaria. La definición matemática de un conjunto también se puede aplicar en Python. Un conjunto es una colección de elementos distintos, desordenados y no indexados. En Python, el conjunto se usa para almacenar elementos únicos, y es posible encontrar la unión, intersección, diferencia, diferencia simétrica, subconjunto, superconjunto y conjunto disjunto entre conjuntos.

#### Creando un Conjunto

Usamos la función incorporada `set()`.

 Creando un conjunto vacío

```
1 #sintaxis
2 st = set()
```

 Creando un conjunto con elementos iniciales

```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3
4 #sintaxis
5 frutas = {'plátano', 'naranja', 'mango', 'limón'}
```

#### Obteniendo la longitud de un Conjunto

Usamos el método `len()` para encontrar la longitud de un conjunto.

```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 len(st)
4
5 #Ejemplo:
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 len(frutas)
```

#### Verificando un elemento

Para verificar si un elemento existe en una lista, usamos el operador de pertenencia `in`.

```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 print("¿El conjunto st contiene elemento3? ", 'elemento3' in st) # ¿El conjunto st contiene elemento3? Verdadero
4
5 #Ejemplo:
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 print('mango' in frutas) # Verdadero
```

#### Añadiendo elementos a un Conjunto

Una vez creado un conjunto, no podemos cambiar ninguno de sus elementos, pero también podemos agregar elementos adicionales.

 Agregue un elemento usando `add()`



```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st.add('elemento5')
4
5 #Ejemplo:
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 frutas.add('lima')
```

- 🐍 Agregar múltiples elementos usando update() El método update() permite agregar múltiples elementos a un conjunto. El método update() toma una lista como argumento.

```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st.update(['elemento5', 'elemento6', 'elemento7'])
4
5 #Ejemplo:
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 verduras = ('tomate', 'patata', 'col', 'cebolla', 'zanahoria')
8 frutas.update(verduras)
```

### Eliminando elementos de un Conjunto

- 🐍 Podemos eliminar un elemento de un conjunto usando el método remove(). Si el elemento no se encuentra, el método remove() provocará errores, por lo que es bueno verificar si el elemento existe en el conjunto dado. Sin embargo, el método discard() no provoca ningún error.

```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st.remove('elemento2')
```

- 🐍 El método pop() elimina un elemento aleatorio de una lista y devuelve el elemento eliminado.

```
1 frutas = {'plátano', 'naranja', 'mango', 'limón'}
2 frutas.pop() # elimina un elemento aleatorio del conjunto
3
4 #Si estamos interesados en el elemento eliminado.
5
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 elemento_eliminado = frutas.pop()
8 print(elemento_eliminado)
```

### Limpiando elementos en un conjunto

Si queremos limpiar o vaciar el conjunto, usamos el método clear().

```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st.clear()
4
5 #Ejemplo:
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 frutas.clear()
8 print(frutas) # set()
```

### Borrando un Conjunto

Si queremos eliminar el conjunto en sí, usamos el operador del.



```
1 #sintaxis
2 st = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 del st
4
5 #Ejemplo:
6 frutas = {'plátano', 'naranja', 'mango', 'limón'}
7 del frutas
```

### Convirtiendo una Lista a Conjunto

Podemos convertir una lista en un conjunto y un conjunto en una lista. Convertir una lista a un conjunto elimina los duplicados y solo se reservarán los elementos únicos.


```
1 #sintaxis
2 lst = ['elemento1', 'elemento2', 'elemento3', 'elemento4', 'elemento1']
3 st = set(lst) # {'elemento2', 'elemento4', 'elemento1', 'elemento3'} - el orden es aleatorio, porque los conjuntos
4
5 #Ejemplo:
6 frutas = ['plátano', 'naranja', 'mango', 'limón', 'naranja', 'plátano']
7 frutas = set(frutas) # {'mango', 'limón', 'plátano', 'naranja'}
```

### Uniendo Conjuntos (Union)

Podemos unir dos conjuntos usando el método union() o update().

 Unión Este método devuelve un nuevo conjunto

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento5', 'elemento6', 'elemento7', 'elemento8'}
4 st3 = st1.union(st2)
5
6 #Ejemplo:
7 frutas = {'plátano', 'naranja', 'mango', 'limón'}
8 verduras = {'tomate', 'patata', 'col', 'cebolla', 'zanahoria'}
9 print(frutas.union(verduras)) # {'limón', 'zanahoria', 'tomate', 'plátano', 'mango', 'naranja', 'col', 'patata',
```

 Update Este método inserta un conjunto en un conjunto dado

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento5', 'elemento6', 'elemento7', 'elemento8'}
4 st1.update(st2) # el contenido de st2 se añade a st1
5
6 #Ejemplo:
7 frutas = {'plátano', 'naranja', 'mango', 'limón'}
8 verduras = {'tomate', 'patata', 'col', 'cebolla', 'zanahoria'}
9 frutas.update(verduras)
10 print(frutas) # {'limón', 'zanahoria', 'tomate', 'plátano', 'mango', 'naranja', 'col', 'patata', 'cebolla'}
```

### Encontrando Elementos (Intersección)

La intersección devuelve un conjunto de elementos que están en ambos conjuntos.

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento3', 'elemento2'}
4 st1.intersection(st2) # {'elemento3', 'elemento2'}
5
6 #Ejemplo:
7 numeros_enteros = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
8 numeros_pares = {0, 2, 4, 6, 8, 10}
9 numeros_enteros.intersection(numeros_pares) # {0, 2, 4, 6, 8, 10}
10
11 python = {'p', 'y', 't', 'h', 'o', 'n'}
12 dragon = {'d', 'r', 'a', 'g', 'o', 'n'}
13 python.intersection(dragon) # {'o', 'n'}
```





### Comprobando Subconjunto y Superconjunto

Un conjunto puede ser un subconjunto o un superconjunto de otros conjuntos:

🐍 Subconjunto: `issubset()`

🐍 Superconjunto: `issuperset()`

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento2', 'elemento3'}
4 st2.issubset(st1) # Verdadero
5 st1.issuperset(st2) # Verdadero
6
7 #Ejemplo:
8 numeros_enteros = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
9 numeros_pares = {0, 2, 4, 6, 8, 10}
10 numeros_enteros.issubset(numeros_pares) # Falso, porque es un superconjunto
11 numeros_enteros.issuperset(numeros_pares) # Verdadero
12
13 python = {'p', 'y', 't', 'h', 'o', 'n'}
14 dragon = {'d', 'r', 'a', 'g', 'o', 'n'}
15 python.issubset(dragon) # Falso
```

### Verificando la Diferencia entre dos Conjuntos

Devuelve la diferencia entre dos conjuntos.

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento2', 'elemento3'}
4 st2.difference(st1) # set()
5 st1.difference(st2) # {'elemento1', 'elemento4'} => st1\st2
6
7 #Ejemplo:
8 numeros_enteros = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
9 numeros_pares = {0, 2, 4, 6, 8, 10}
10 numeros_enteros.difference(numeros_pares) # {1, 3, 5, 7, 9}
11
12 python = {'p', 'y', 't', 'o', 'n'}
13 dragon = {'d', 'r', 'a', 'g', 'o', 'n'}
14 python.difference(dragon) # {'p', 'y', 't'} - el resultado no está ordenado (característica de los conjuntos)
15 dragon.difference(python) # {'d', 'r', 'a', 'g'}
```

### Encontrando la Diferencia Simétrica entre dos Conjuntos

Devuelve la diferencia simétrica entre dos conjuntos. Esto significa que devuelve un conjunto que contiene todos los elementos de ambos conjuntos, excepto los elementos que están presentes en ambos conjuntos,

matemáticamente:  $(A \setminus B) \cup (B \setminus A)$

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento2', 'elemento3'}
4
5 #significa (A\B)U(B\A)
6 st2.symmetric_difference(st1) # {'elemento1', 'elemento4'}
7
8 #Ejemplo:
9 numeros_enteros = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
10 algunos_numeros = {1, 2, 3, 4, 5}
11 numeros_enteros.symmetric_difference(algunos_numeros) # {0, 6, 7, 8, 9, 10}
12
13 python = {'p', 'y', 't', 'h', 'o', 'n'}
14 dragon = {'d', 'r', 'a', 'g', 'o', 'n'}
15 python.symmetric_difference(dragon) # {'r', 't', 'p', 'y', 'g', 'a', 'd', 'h'}
```



### Uniendo Conjuntos Disjuntos

Si dos conjuntos no tienen un elemento o elementos comunes, los llamamos conjuntos disjuntos. Podemos comprobar si dos conjuntos son conjuntos o disjuntos usando el método `isdisjoint()`.

```
1 #sintaxis
2 st1 = {'elemento1', 'elemento2', 'elemento3', 'elemento4'}
3 st2 = {'elemento2', 'elemento3'}
4 st2.isdisjoint(st1) # Falso
5
6 #Ejemplo:
7 numeros_pares = {0, 2, 4, 6, 8}
8 numeros_impares = {1, 3, 5, 7, 9}
9 numeros_pares.isdisjoint(numeros_impares) # Verdadero, porque no hay elementos comunes
10
11 python = {'p', 'y', 't', 'h', 'o', 'n'}
12 dragon = {'d', 'r', 'a', 'g', 'o', 'n'}
13 python.isdisjoint(dragon) # Falso, hay elementos comunes {'o', 'n'}
```

## 4. Desarrollo y Actividades

### Ejercicio parte 01:

- 1) La siguiente es una lista de las edades de 10 estudiantes:  
edades = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24] (4ptos)
  - a. Ordena la lista y encuentra la edad mínima y máxima
  - b. Añade de nuevo la edad mínima y máxima a la lista
  - c. Encuentra la edad mediana (un elemento del medio o dos elementos del medio divididos por dos)
  - d. Encuentra la edad promedio (suma de todos los elementos divididos por su número)
  - e. Encuentra el rango de las edades (máximo menos mínimo)
  - f. Compara el valor de (mínimo - promedio) y (máximo - promedio), usa el método `abs()`
- 2) "Soy profesor y me encanta inspirar y enseñar a la gente". ¿Cuántas palabras únicas se han utilizado en la frase? Usa los métodos de `split` y `sets` para obtener las palabras únicas. (4pto)
- 3) Considera tres listas de números enteros, una con números del 1 al 10, otra con números del 5 al 15, y la última con números del 10 al 20. (5ptos)
  - a. Convierte las listas en conjuntos.
  - b. Encuentra el conjunto de todos los números que están presentes en las tres listas.
  - c. Encuentra el conjunto de todos los números que están presentes en al menos una de las listas.
  - d. Encuentra el conjunto de todos los números que están presentes en la primera lista, pero no en la última.
  - e. Convierte los conjuntos obtenidos en tuplas y suma el primer y último elemento de cada tupla.
- 4) Considera dos listas, `l1` y `l2`, que contienen tuplas. Cada tupla consta de una cadena de texto y un número entero. La lista `l1` tiene 5 elementos y la lista `l2` tiene 3 elementos. (7ptos)  
`l1 = [("one", 1), ("two", 2), ("three", 3), ("four", 4), ("five", 5)]`  
`l2 = [("one", 1), ("two", 2), ("six", 6)]`  
Tu tarea es:
  - a. Crear conjuntos a partir de estas listas, `s1` y `s2`.
  - b. Encontrar los elementos que son comunes a `s1` y `s2` y almacenarlos en un conjunto `s3`.
  - c. Encontrar los elementos que son únicos para `s1` y `s2` y almacenarlos en un conjunto `s4`.



- d. Crear una nueva lista l3 que contenga los elementos de s3 y s4 ordenados por el número entero de cada tupla.

