CONJUNTOS

Algoritmos y estructuras de datos I

CONJUNTOS

Un conjunto, llamado *set* en Python, es una colección de elementos desordenados que no admite duplicados. Se trata, por tanto, de una estructura de datos equivalente a los conjuntos en matemáticas y nos permite usar algunas de sus operaciones asociadas como: intersección, diferencia, unión y diferencia simétrica.

La principal característica de este tipo de datos es que es una colección cuyos elementos no guardan ningún orden y que además son únicos.

Estas características hacen que los principales usos asociados a los conjuntos sean, por ejemplo, conocer si un elemento pertenece o no a una colección y eliminar duplicados de un tipo iterable. Además, como sucede con los elementos de las listas, los elementos de un set no necesariamente han de ser del mismo tipo.

CREACIÓN DE CONJUNTOS

En Python los sets se pueden crear de dos formas distintas. La primera de ellas sería utilizando llaves ({}) y separando sus elementos con comas (,). Recordemos que los sets no admiten elementos duplicados. En el ejemplo a continuación el valor 1 al estar duplicado, se agrega solo una vez:

```
fibonacci = {0, 1, 1, 2, 3}
print(fibonacci) #{0, 1, 2, 3}
```

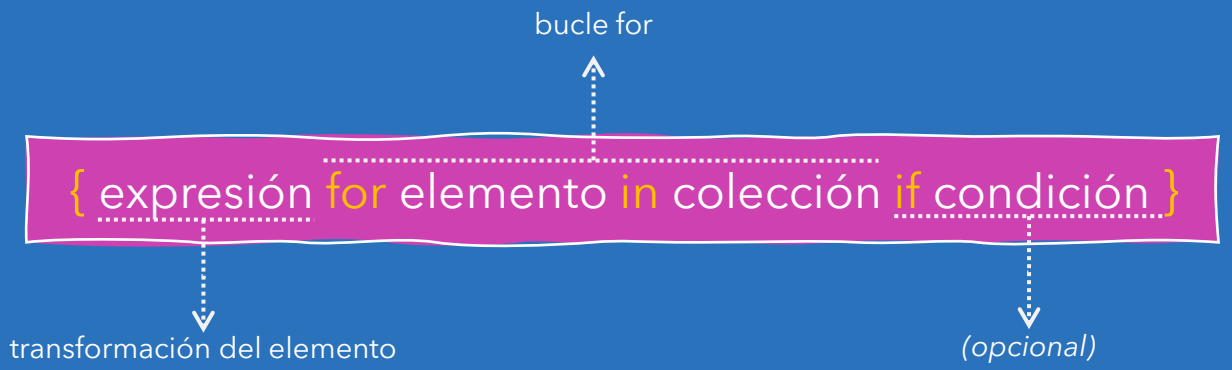
Incluso, podemos usar la función set para crear un conjunto en función de una lista:

```
fibonacci = set([0, 1, 1, 2, 3])
```

Para crear un conjunto vacío (sin elementos), lo podemos hacerlo únicamente mediante la función set. Utilizar las llaves ({}) no crea un conjunto vacío, sino un diccionario vacío:

```
fibonacci = set()
print(type(fibonacci)) #<class 'set'>

fibonacci = {}
print(type(fibonacci)) #<class 'dict'>
```




```
numeros = [1, 2, 2, 3, 4, 4, 5]
cuadrados = {i**2 for i in numeros}
#{1, 4, 9, 16, 25}

frase = "hola mundo hola universo"
palabras = {palabra for palabra in frase.split()}
#{'hola', 'mundo', 'universo'}

cadena = "susurrar"
letrasUnicas = {letra for letra in cadena}
#{'s', 'r', 'a', 'u'}
```

COMPRENSIÓN DE CONJUNTOS

Al igual que lo podemos hacer con las listas, por medio de la comprensión, también podremos generar conjuntos a partir de los elementos de otros elementos iterables de una forma rápida de escribir, muy legible y funcionalmente eficiente.

A menudo la expresión (lo que terminará insertado en el conjunto resultante) es igual al elemento que se recorre y la condición es opcional. La colección puede ser una lista o cualquier otro objeto iterable.

Entre llaves se escribe una expresión seguida de un bucle for sobre el que se itera, para finalmente escribir una condición de ser necesaria.

ACCEDER A LOS ELEMENTOS

Los conjuntos son colecciones de elementos desordenados que no están indexados. Por tanto, si intentamos acceder a sus elementos como lo haríamos en una lista (por medio de un índice) obtenemos un error de tipo (*TypeError*). En su lugar, para acceder a los elementos de un set tenemos que recurrir a un bucle *for*:

```
frutas = {"manzana", "pera", "manzana", "uva", "pera"}

for fruta in frutas:
    print(fruta)
```

También podemos comprobar si un elemento pertenece a un set mediante el uso de in:

```
animales = {"perro", "gato", "pájaro", "pez"}

if "tortuga" in animales:
    print("Existe el animal en el conjunto")
else:
    print("No existe el animal en el conjunto")
```

```
colores = {"azul", "blanco"}

colores.add("azul")  #Ya existe, no se agrega
colores.add("negro") #Agregar un nuevo color

print(colores)  #{'azul', 'blanco', 'negro'}

coloresAdicionales = ["rojo", "verde"]

colores.update(coloresAdicionales)

print(colores)  #{'rojo', 'negro', 'blanco', 'verde', 'azul'}
```

AÑADIR ELEMENTOS

Para añadir un nuevo elemento a un conjunto utilizaremos el método *add()* que sólo añade el elemento si éste no se encuentra dentro del *set*.

Podemos añadir varios elementos a la vez con el método *update()*, que recibe un iterable (conjunto, lista, tupla o rango) como parámetro de entrada.

COPIAR EL CONTENIDO DE UN CONJUNTO

Podemos encontrar un escenario en el que queramos hacer una copia de un conjunto. La forma más directa de realizar esta acción es a través del método *copy()*.

Como hemos visto anteriormente para las listas (y ocurre de igual manera para los conjuntos), el signo de igualdad = se puede utilizar para construir un duplicado de la misma, pero esto implica que el nuevo conjunto también se verá alterado si se actualiza la variable original o viceversa.

Entonces, una copia de los elementos existentes se puede lograr utilizando el método *copy()*, que no toma ningún parámetro en su llamado:

```
conjuntoA = {0, 1, 1, 2, 3, 5, 8}
conjuntoB = conjuntoA

#los identificadores son iguales
print(id(conjuntoA), id(conjuntoB))


conjuntoC = {13, 21, 34, 55}
conjuntoD = conjuntoC.copy()

#los identificadores son distintos
print(id(conjuntoC), id(conjuntoD))
```

FUNCIONES DE AGREGACIÓN

Tal como existen para otras estructuras de datos, también podemos aplicar funciones de agregación a los conjuntos. Las funciones de agregación son operaciones que toman múltiples elementos de una estructura de datos y las combinan para producir un único valor de resumen. Estas funciones nos permiten sintetizar y analizar datos de manera eficiente.

Por ejemplo, podemos calcular la cantidad total de elementos, encontrar el valor máximo o mínimo, y determinar la sumatoria de todos los elementos en un conjunto. Estas operaciones son útiles para obtener una visión general de los datos y extraer información significativa de grandes volúmenes de datos de manera rápida y precisa.

```
fibonacci = {0, 1, 1, 2, 3, 5, 8, 13}

#Impresión de la cantidad de elementos únicos en el conjunto
print("Cantidad de elementos únicos: ",
len(fibonacci))  #7

#Impresión del valor máximo en el conjunto
print("Máximo: ", max(fibonacci))  #13

#Impresión del valor mínimo en el conjunto
print("Mínimo: ", min(fibonacci))  #0

#Impresión de la sumatoria de los valores en el conjunto
print("Sumatoria: ", sum(fibonacci))  #32
```

ELIMINAR ELEMENTOS

Para eliminar elementos de un set existen dos métodos principales que son *remove()* y *discard()*.

La diferencia entre estos dos métodos radica en que el método *remove()* lanza un error de tipo *KeyError* si el elemento que intentamos eliminar no se encuentra el set:

Por el contrario, el método *discard()* no hace nada si intentamos eliminar un elemento que no pertenece al *set*.

Para eliminar todos los elementos del conjunto podemos usar el método *clear()*.

```
elementos = {"agua", "tierra", "fuego", "aire"}

elementos.discard("tierra")
elementos.discard("metal")
print(elementos)  #{'agua', 'fuego', 'aire'}

elementos.remove("fuego")
print(elementos)  #{'agua', 'aire'}

try:
    elementos.remove("metal")
except KeyError:
    print("No existe el elemento a eliminar.")


elementos.clear()
print(elementos)  #set()
```

```
productosCarrito = ["Camiseta", "Pantalones",
"Zapatos", "Camiseta", "Bufanda"]

productosUnicos = list(set(productosCarrito))

print(productosUnicos)


barajaDeCartas = (("A", "Corazones"), ("Q",
"Corazones"), ("A", "Corazones"), ("K", "Corazones"))

cartasUnicas = tuple(set(barajaDeCartas))

print(cartasUnicas)
```

ELIMINAR ELEMENTOS DUPLICADOS DE UNA LISTA O TUPLA

Otra operación que podemos realizar utilizando sets es remover los elementos duplicados de una lista.

Si no usamos conjuntos, probablemente recorreríamos la lista en un bucle, e iríamos añadiendo cada elemento a otra lista después de haber comprobado su pertenencia. Sin embargo, esta operación es mucho más sencilla si primero convertimos nuestra lista en un set y luego reconvertimos el resultado a una lista como se observa en el ejemplo.

CONVERTIR UNA CADENA DE CARACTERES A UN CONJUNTO

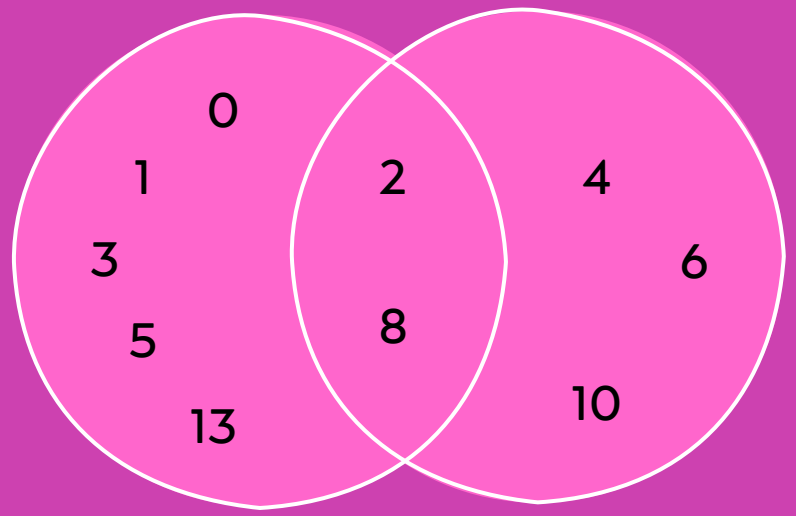
Es posible convertir una cadena de caracteres (str) en un conjunto (set) en Python utilizando la función set().

Se debe tener en cuenta que cada carácter será considerado como un elemento al realizarse la conversión.

Los conjuntos en Python son colecciones desordenadas, lo que significa que el orden de los elementos no está garantizado. El conjunto resultante puede tener los elementos en un orden diferente al de la cadena original.

```
cadena = "Hola mundo"
conjunto = set(cadena)

print(conjunto) #{'l', 'H', 'a', ' ', 'm', 'd', 'u', 'o', 'n'}
```



OPERACIONES: UNIÓN

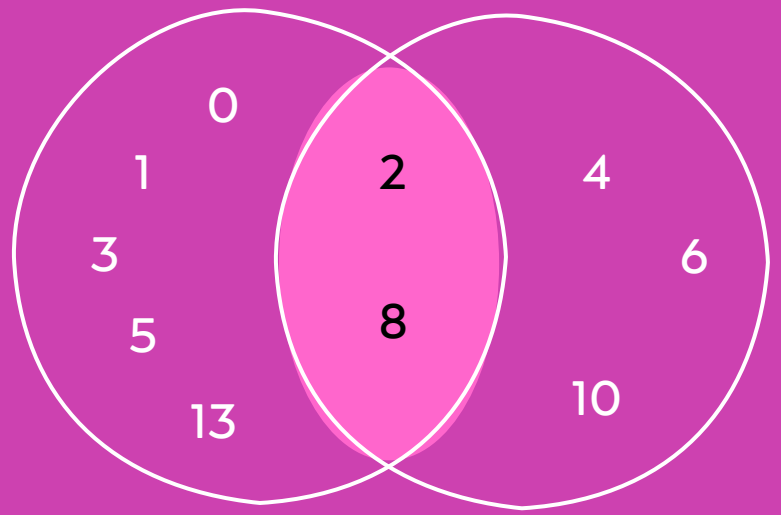
La unión de dos conjuntos retorna un conjunto que contiene todos los elementos pertenecientes a alguno de los dos conjuntos.

Esta operación se puede realizar mediante la barra vertical (|) o el método *union()*:

```
fibonacci = {0, 1, 1, 2, 3, 5, 8, 13}
pares = {2, 4, 6, 8, 10}

#{0, 1, 2, 3, 4, 5, 6, 8, 10, 13}
print(fibonacci | pares)

#{0, 1, 2, 3, 4, 5, 6, 8, 10, 13}
print(fibonacci.union(pares))
```

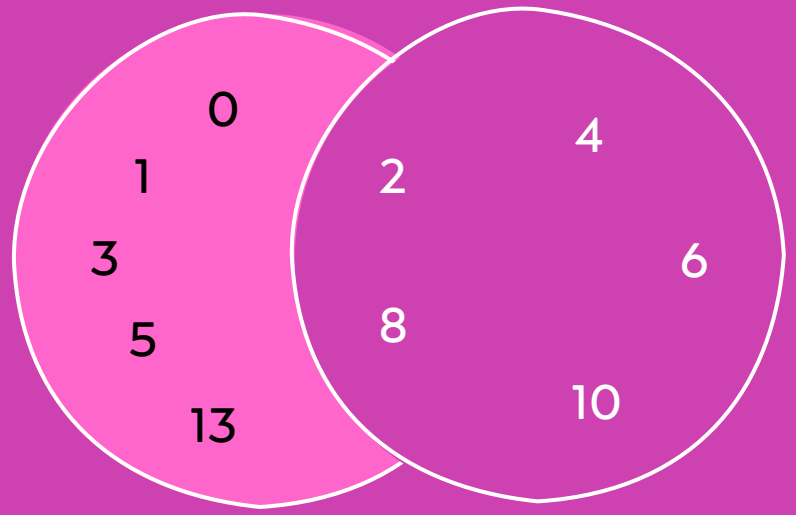


OPERACIONES: INTERSECCIÓN

La intersección de dos conjuntos es un conjunto que contiene los elementos que pertenecen a ambos conjuntos. Esta operación se realiza con el símbolo *ampersand (&)* o el método *intersection()*:

```
fibonacci = {0, 1, 1, 2, 3, 5, 8, 13}
pares = {2, 4, 6, 8, 10}

print(fibonacci & pares) #{8, 2}
print(fibonacci.intersection(pares)) #{8, 2}
```



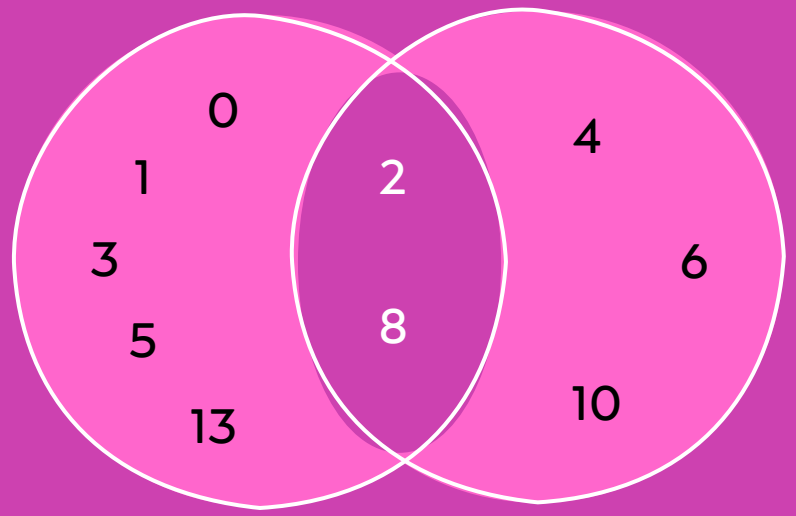
OPERACIONES: DIFERENCIA

La diferencia entre un conjunto A y un conjunto B es el resultado de eliminar del conjunto A cualquier elemento presente en B. Por tanto, el orden en el que se realiza la operación es significativo.

Podemos calcular la diferencia entre dos conjuntos mediante el signo menos (-) o el método *difference()*:

```
fibonacci = {0, 1, 1, 2, 3, 5, 8, 13}
pares = {2, 4, 6, 8, 10}

#{0, 1, 3, 5, 13}
print(fibonacci - pares)
#{0, 1, 3, 5, 13}
print(fibonacci.difference(pares))
```



OPERACIONES: DIFERENCIA SIMÉTRICA

La diferencia simétrica entre dos conjuntos comprende todos los elementos que pertenecen a uno de los dos conjuntos, pero no a ambos a la vez. Esta operación se puede realizar en Python mediante el acento circunflejo (^) o el método *symmetric_difference()*:

```
fibonacci = {0, 1, 1, 2, 3, 5, 8, 13}
pares = {2, 4, 6, 8, 10}

#{0, 1, 3, 4, 5, 6, 10, 13}
print(fibonacci ^ pares)
#{0, 1, 3, 4, 5, 6, 10, 13}
print(fibonacci.symmetric_difference(pares))
```

OPERACIONES: SUBCONJUNTO

En términos sencillos, un subconjunto es un conjunto que contiene solo elementos que también están presentes en otro conjunto más grande. En Python, se utiliza el operador <= o el método *issubset()* para verificar si un conjunto es subconjunto de otro.

En este caso, el conjunto *ingresantes* contiene los elementos "Oliver" y "Gabriela", y el conjunto *empleados* contiene los elementos "Irina", "Gabriela", "Oliver" y "Giuliana".

Como todos los elementos de *ingresantes* están presentes en empleados, se considera que *accesoBasico* es un subconjunto de empleados. Por lo tanto, la salida será *True*.

```
#Conjuntos de empleados y empleados con acceso a un sistema específico
ingresantes = {"Oliver", "Gabriela"}
empleados = {"Irina", "Gabriela", "Oliver", "Giuliana"}

#Verificación si todos los empleados con acceso básico también están en la lista de todos los empleados
print(ingresantes <= empleados)  #True
print(ingresantes.issubset(empleados))  #True
```

EJEMPLO DE IMPLEMENTACIÓN

Supongamos que tenemos un estudiante que ha aprobado ciertas materias, y queremos ver cuáles le faltan para completar su carrera. Además, compararemos estas materias con las de otra carrera para ver cuáles son comunes y el porcentaje de avance en su carrera.

```
#Materias aprobadas por el estudiante
materiasAprobadas = {"Fundamentos de informática", "Álgebra",
"Matemáticas"}

#Materias requeridas para completar primer año
materiasIngenieriaInformatica = {"Fundamentos de informática",
"Álgebra", "Matemáticas", "Programación I", "Química"}
materiasIngenieriaIndustrial = {"Introducción a la programación", "Física",
"Matemáticas", "Álgebra", "Química"}

#Materias que le faltan al estudiante para completar la Carrera Ingeniería Informática
materiasFaltantes =
materiasIngenieriaInformatica.difference(materiasAprobadas)

#Calcular progreso
porcentajeProgreso = (len(materiasAprobadas) * 100) /
len(materiasIngenieriaInformatica)

#Materias en común entre ambas carreras
materiasEnComunEntreCarreras =
materiasIngenieriaInformatica.intersection(materiasIngenieriaIndustrial)

print(f"Materias que le faltan al estudiante para completar Ing. Infomática:
{materiasFaltantes}")
print(f"Materias en común entre la Ing. Industrial e Ing. Informática:
{materiasEnComunEntreCarreras}")
print(f"Progreso del estudiante en su primer año:
{porcentajeProgreso:.2f}%")
```

EJEMPLO DE IMPLEMENTACIÓN

Este ejemplo muestra cómo utilizar conjuntos para gestionar personal asignado a dos proyectos, identificando todas las personas involucradas mediante la unión, aquellas dedicadas exclusivamente a un proyecto con la diferencia simétrica, y verificando si un conjunto de empleados está completamente asignado a ambos proyectos usando *issubset*.

Estas operaciones ayudan a optimizar la asignación de recursos, evitar sobrecargas de trabajo, y garantizar la disponibilidad adecuada del equipo en cada proyecto.

```
#Personas asignadas a los proyectos
personasProyectoX = {"Oliver", "Gabriela", "Giulina", "Irina"}
personasProyectoY = {"Gabriela", "Oliver"}

#Unión de personas asignadas al menos a uno de los dos proyectos
personasUnion = personasProyectoX.union(personasProyectoY)

#Personas asignadas solo a uno de los dos proyectos
personasDiferenciaSimetrica =
personasProyectoX.symmetric_difference(personasProyectoY)

#Verificar si todas las personas del Proyecto Y están también asignadas al
Proyecto X
esSubset = personasProyectoY.issubset(personasProyectoX)

print("Personas asignadas al menos a uno de los dos proyectos
(Unión):", personasUnion)
print("Personas asignadas solo a uno de los dos proyectos
(Diferencia Simétrica):", personasDiferenciaSimetrica)
print("¿Todas las personas del Proyecto X están también asignadas al
Proyecto Y? (Subset):", esSubset)
```

Lic. Yaps David P.

EJEMPLO DE IMPLEMENTACIÓN

En este ejemplo, se realizan operaciones sobre tres conjuntos de ingredientes para distintas versiones de una pizza: básica, vegetariana y sin gluten.

```
#Ingredientes para la pizza básica
ingredientesBasica = {"harina", "agua", "levadura", "sal", "queso mozzarella"}

#Ingredientes para la pizza vegetariana
ingredientesVegetariana = {"harina", "agua", "levadura", "sal", "queso mozzarella",
"pimientos", "cebolla"}

#Ingredientes para la pizza sin gluten
ingredientesSinGluten = {"harina sin gluten", "agua", "levadura", "sal", "queso
mozzarella"}

#Ingredientes que son comunes entre las tres versiones (intersección)
ingredientesComunes =
ingredientesBasica.intersection(ingredientesVegetariana).intersection(ingredientes
SinGluten)

#Ingredientes en la pizza vegetariana que no están en la unión de la básica y la sin gluten
soloVegetariana = ingredientesVegetariana -
(ingredientesBasica.union(ingredientesSinGluten))

#Verificar si todos los ingredientes de una versión (sin gluten) están en cualquiera de las otras versiones
(subset combinado)
esSubreceta =
ingredientesSinGluten.issubset(ingredientesBasica.union(ingredientesVegetariana))

print("Ingredientes comunes entre todas las versiones:", ingredientesComunes)
print("Ingredientes solo en la pizza vegetariana:", soloVegetariana)
print("¿Los ingredientes de la pizza sin gluten están en cualquiera de las otras dos
versiones? (Subset):", esSubreceta)
```

Lic. Yaps David P.

AUTOEVALUACIÓN

La autoevaluación es crucial para identificar mejoras y fortalezas de forma objetiva, facilitando la fijación de metas y la toma de acciones para alcanzarlas.