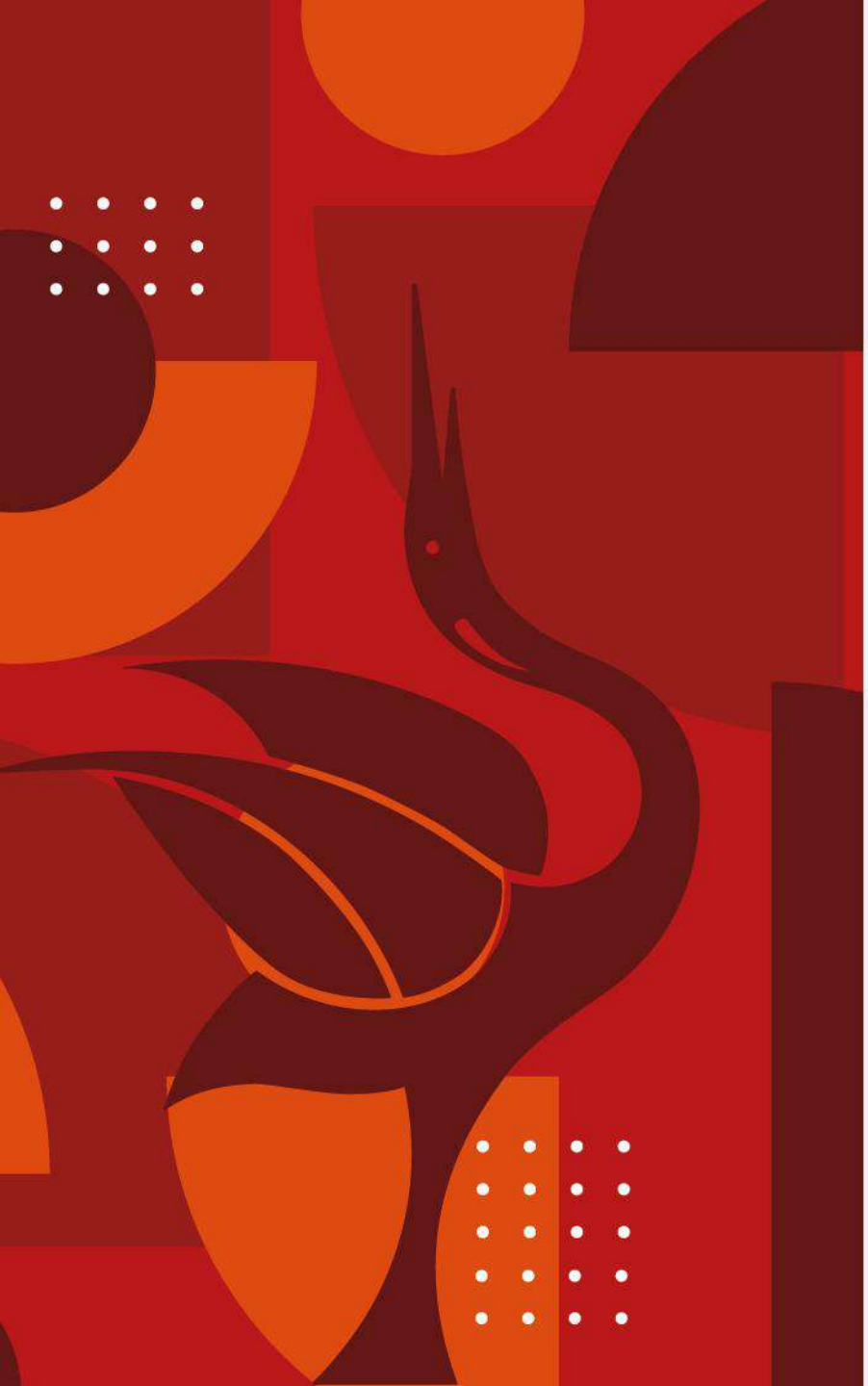


Área Académica de Computación y Electrónica
Licenciatura en Ciencias Computacionales
Matemáticas Discretas

Conjuntos en Python

Elaboró:
M. en C. Fabiola Martínez Juárez
M. en C. Ma. Judith Franco Torres

Enero – Junio 2023

Topic

Sets in Python

Abstract

This document addresses the concept of a set as a data structure. The definition or creation of sets in the Python programming language is exemplified, and the syntax to execute operations between sets such as union, intersection, difference and symmetric difference is exposed.

Keywords: set, union, intersection, difference, symmetric difference, Python, data structures.

Tema

Conjuntos en Python

Resumen

El presente documento aborda el concepto de conjunto como una estructura de datos. Se ejemplifica la definición o creación de conjuntos en el lenguaje de programación Python, y se expone la sintaxis para ejecutar operaciones entre conjuntos, tales como: unión, intersección, diferencia y diferencia simétrica.

Palabras Clave: conjunto, unión, intersección, diferencia, diferencia simétrica, Python, estructuras de datos.

Objetivo de Aprendizaje

Describir el concepto de conjunto como una estructura de datos que permite almacenar datos en ella, y en la cual se pueden añadir y eliminar elementos; así como aplicar las operaciones que han sido definidas por las matemáticas.

Estructuras de datos en Python

Los lenguajes de programación proveen de diferentes maneras para agrupar objetos. Arreglos, listas, vectores, diccionarios, son ejemplos de grupos de datos o estructuras de datos. Las estructuras de datos en programación se caracterizan porque se pueden añadir o eliminar elementos de ellas.

El lenguaje de programación **Python** provee de 4 tipos de datos incorporados (*built-in data types*) para crear colecciones de objetos: listas, tuplas, diccionarios y conjuntos.

Un conjunto en **Python** guarda las mismas características de un conjunto definido por las matemáticas:

- los elementos de un conjunto ***no tienen orden***,
- un conjunto ***no admite elementos duplicados***.

El ambiente Google Colaboratory

Para replicar los ejemplos expuestos en este material, se recomienda utilizar **Colab**. **Google Colaboratory**, o simplemente **Colab**, es un ambiente en línea que permite escribir y ejecutar código Python en documentos ejecutables que son almacenados dentro de Google Drive.

A estos documentos ejecutables se les denomina cuadernos. Cada cuaderno se compone de celdas, las cuales pueden contener código, texto, imágenes, entre otros.

Figura 1. Logo **Google Colaboratory**.
Fuente: https://colab.research.google.com.

El ambiente Google Colaboratory

Colab funciona completamente en la nube, es decir, no requiere de instalación previa para poder ejecutar código python. Solo es necesario verificar que el botón de la barra de herramientas indique **CONECTADO**. Para ejecutar el código de una celda bastará con dar click en el icono de **PLAY** que aparece a la izquierda de cada celda.

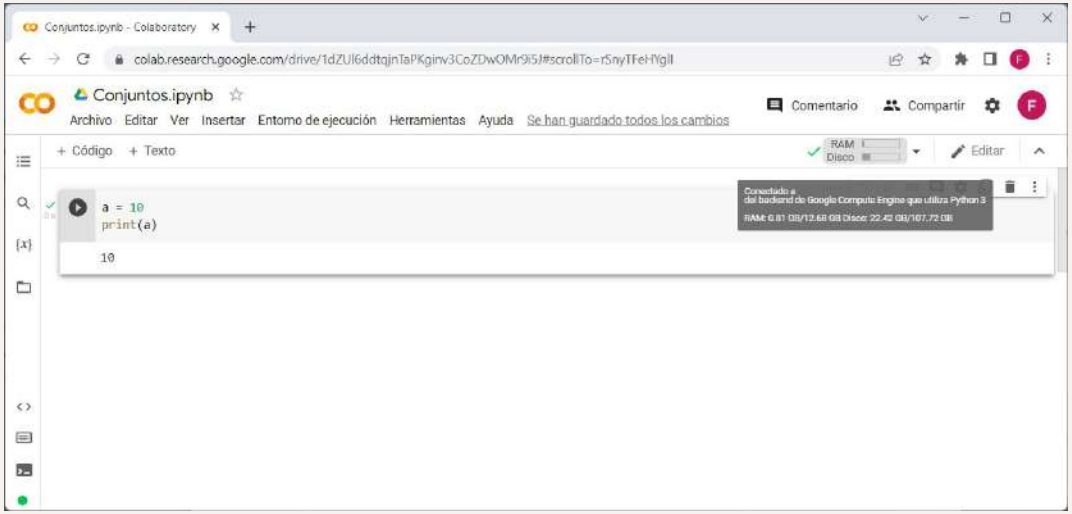


Figura 2. Ambiente **Google Colaboratory**.
Fuente: Elaboración propia.

Definición de conjuntos en Python

Existen dos maneras distintas de definir conjuntos en lenguaje Python:

- *pasándole una lista a la función* **`set`**, *o*
- *escribiendo los objetos encerrados entre llaves y separados por comas.*

En la figura, se define el conjunto **`A`** mediante la función **`set()`** pasándole la lista **`['w','e','j','c','p','r']`** como parámetro de la función. El comando **`type()`** retorna el tipo de dato del objeto parámetro, en este caso para **`A`** es de tipo **`set`**. Obsérvese que cuando se imprimen los elementos del conjunto, estos no guardan ningún orden.

Figura 3. Creación de un conjunto mediante la función **set()**.
Fuente: Elaboración propia.

Definición de conjuntos en Python

En el ejemplo de la figura se define el conjunto `B`, escribiendo los elementos entre llaves y separados por comas. Nótese que aunque el elemento `'m'` se escribe 2 veces, el conjunto final no lo considera como dos elementos diferentes, es decir, no se aceptan elementos duplicados.

```
[2]  B = {'x','p','m','j','w','m'}
     print(B)
     type(B)

     {'x', 'j', 'm', 'w', 'p'}
     set
```

Figura 4. Creación de un conjunto mediante llaves.
Fuente: Elaboración propia.

Cardinalidad de un conjunto

Para determinar la cardinalidad de un conjunto, es decir, el número de elementos que tiene, se utiliza la función `len()`. Para el ejemplo del conjunto `A` su cardinalidad es de **6**.

```
[3]  A = set(['w','e','j','c','p','r'])
     len(A)

     6
```

Figura 5. Ejemplo de aplicación del método **len()**.
Fuente: Elaboración propia.

Para el conjunto `B`, se tiene que su cardinalidad es de **5**, recordando que una estructura conjunto no acepta elementos duplicados.

```
[4]  B = {'x','p','m','j','w','m'}
     len(B)

     5
```

Figura 6. Ejemplo de aplicación del método **len()**.
Fuente: Elaboración propia.

Pertenencia a un conjunto

Si se desea conocer la pertenencia de un elemento a un conjunto, se utiliza el operador **in**, el cual retorna el valor de **True** o **False**. La figura ejemplifica que `'a' ∉ A`, es decir, que `'a'` no pertenece al conjunto **A**.

```
[5]  A = set(['w','e','j','c','p','r'])
     'a' in A

     False
```

Figura 7. Ejemplo de aplicación del operador **in**.
Fuente: Elaboración propia.

En el siguiente ejemplo el elemento `'j'` pertenece al conjunto **B**, es decir `'a' ∈ B,` tal y como se muestra en la figura.

```
[6]  B = {'x','p','m','j','w','m'}
     'j' in B

     True
```

Figura 8. Ejemplo de aplicación del operador **in**.
Fuente: Elaboración propia.

Operaciones entre conjuntos

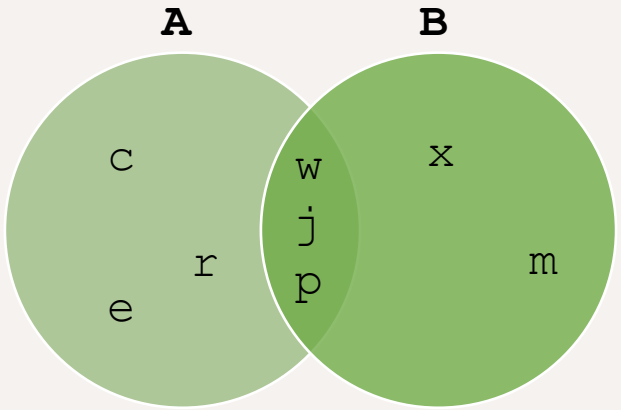
Python cuenta con operadores ya establecidos para realizar fácilmente las operaciones de conjuntos: unión, intersección, diferencia y diferencia simétrica.

El operador | (barra vertical) aplica la operación de unión, devolviendo un conjunto que incluye todos los elementos que están en el primer conjunto más los elementos que están en el segundo conjunto.

```
[7]  union = (A | B)
     print (union)

     {'e', 'r', 'x', 'j', 'c', 'm', 'w', 'p'}
```

Figura 9. Ejemplo de aplicación del operador unión.
Fuente: Elaboración propia.



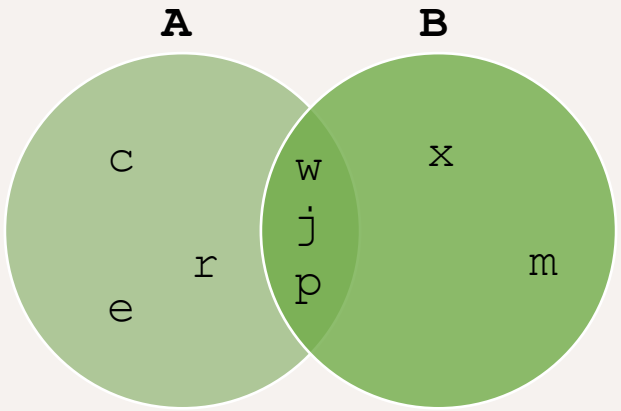
Operaciones entre conjuntos

Para la operación intersección, se tiene el operador **&** (ampersand), que al ser aplicado a dos conjuntos devuelve un tercer conjunto que incluye a los elementos que pertenecen a ambos conjuntos.

```
[8]  interseccion = (A & B)
     print (interseccion)

     {'w', 'j', 'p'}
```

Figura 10. Ejemplo de aplicación del operador intersección.
Fuente: Elaboración propia.



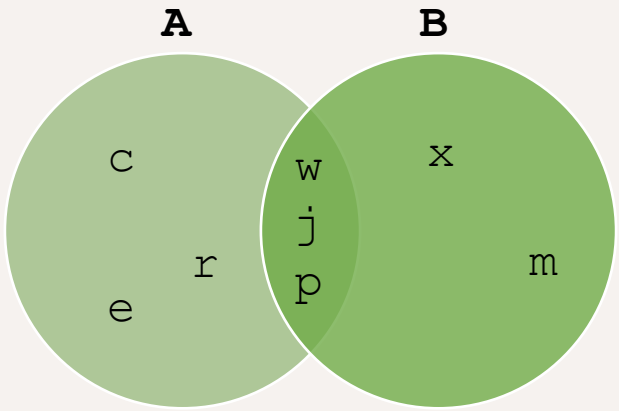
Operaciones entre conjuntos

La diferencia entre dos conjuntos devuelve los elementos del primer conjunto eliminando los elementos que comparte con el segundo conjunto. Se obtiene mediante el operador – (menos).

```
[9]  diferencia = (A - B)
     print (diferencia)

     {'c', 'r', 'e'}
```

Figura 11. Ejemplo de aplicación del operador diferencia.
Fuente: Elaboración propia.



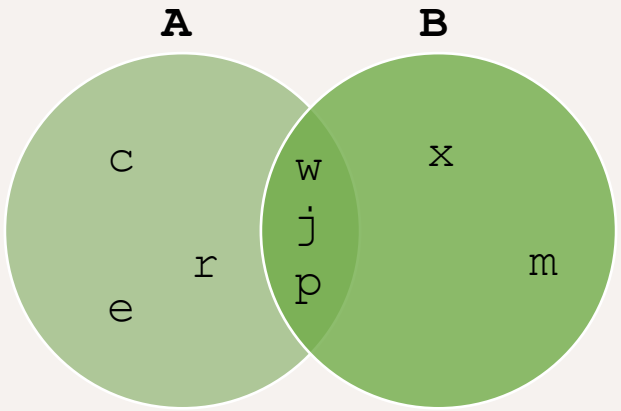
Operaciones entre conjuntos

Finalmente, la diferencia simétrica entre dos conjuntos, que se realiza con el operador ^ (acento circunflejo), devuelve los elementos que pertenecen al primer conjunto más los elementos que pertenecen al segundo conjunto, eliminando los elementos que pertenecen a ambos conjuntos.

```
[10] dif_simetrica = (A ^ B)
     print (dif_simetrica)

     {'r', 'e', 'm', 'c', 'x'}
```

Figura 12. Ejemplo de aplicación del operador diferencia simétrica.
Fuente: Elaboración propia.



Métodos de la estructura set

Visto al conjunto como una estructura de datos que puede manipularse, entonces es necesario contar con métodos para modificar dichas estructuras. El lenguaje de programación Python ofrece métodos para añadir o eliminar elementos para su tipo de dato específico **<set>**.

El método **add()** añade un elemento al conjunto, el cual se sitúa entre paréntesis (parámetro de la función). En la figura se añade el elemento **'b'** al conjunto **C**.

Figura 13. Ejemplo de aplicación del método **add()**.
Fuente: Elaboración propia.

Métodos de la estructura set

Para eliminar un elemento de un conjunto, se tiene el método **`remove()`**. En la figura se muestra la eliminación del elemento `'s'` del conjunto `C`, el cual fue situado entre paréntesis como parámetro de la función.

```
[12] C = {'h', 's', 'q'}
     print(C)
     C.remove('s')
     print(C)

     {'h', 'q', 's'}
     {'h', 'q'}
```

Figura 14. Ejemplo de aplicación del método **remove()**.
Fuente: Elaboración propia.

Métodos de la estructura set

En caso de que el elemento que se desea eliminar mediante el método **`remove()`** no pertenezca al conjunto, entonces se lanza una excepción. Obsérvese en la figura que al tratar de eliminar el elemento `'r'`, el cual no pertenece al conjunto `C`, se lanza una excepción de tipo **`KeyError`**.

Figura 15. Ejemplo de lanzamiento de excepción en la aplicación del método **remove()**.
Fuente: Elaboración propia.

Métodos de la estructura set

A diferencia del método **`remove()`**, el método **`discard()`** elimina también el elemento indicado como parámetro de la función, pero sin lanzar ninguna excepción en caso de que el elemento no pertenezca al conjunto.

En el ejemplo de la figura, se intenta eliminar el elemento **`'r'`**, el cual no pertenece al conjunto **`C`**, pero no se lanza ninguna excepción.

Figura 16. Ejemplo de aplicación del método **discard()**.
Fuente: Elaboración propia.

Métodos de la estructura set

El método `pop()` elimina aleatoriamente un elemento del conjunto. En la figura, se observa que el elemento eliminado del conjunto `C` fue `'h'`.

```
[15] C = {'h', 's', 'q'}
     print(C)
     C.pop()
     print(C)

     {'h', 'q', 's'}
     {'q', 's'}
```

Figura 17. Ejemplo de aplicación del método **pop()**.
Fuente: Elaboración propia.

Métodos de la estructura set

Finalmente, si se desea eliminar todos los elementos de cierto conjunto, debe aplicarse el método **clear()**. En la figura, puede observarse que después de aplicarse este método al conjunto **C**, todos sus elementos han sido removidos.

Figura 18. Ejemplo de aplicación del método **clear()**.
Fuente: Elaboración propia.

Conclusiones

El concepto de conjunto es muy importante tanto en matemáticas como en programación. Las matemáticas establecen las características de un conjunto: colección de objetos sin orden alguno y sin duplicados. Establecen de igual manera la pertenencia de un objeto, su cardinalidad y sus operaciones.

Este concepto ha sido retomado por los lenguajes de programación y desarrollado como una estructura de datos ya incorporada, de la cual, solo es necesario familiarizarse con su sintaxis para aplicarse en contextos de desarrollo de aplicaciones donde sea necesario una estructura de datos que permita añadir y eliminar elementos de ella, y donde no importe su orden y donde sea necesario no contar con elementos duplicados.

Referencias

[1] Caballero, L. (Sep, 2022). Entrenamiento de programación en Python 3 - Nivel básico. 3.12. Tipo conjuntos. Recuperado de: https://entrenamiento-python-basico.readthedocs.io/es/3.7/leccion3/tipo_conjuntos.html

[2] Gersting, J. L. (2014) Mathematical Structures for Computer Science. Discrete Mathematics and Its Applications (7th ed.) USA: W. H. Freeman and Company

[3] Teoh, T. T., Rong, Z. (2022) Artificial Intelligence with Python. Series Title: Machine Learning: Foundations, Methodologies, and Applications. Editorial: Springer Singapore DOI: https://doi.org/10.1007/978-981-16-8615-3

Datos de contacto

Universidad Autónoma del Estado de Hidalgo
Instituto de Ciencias Básicas e Ingeniería
Área Académica de Computación y Electrónica
Licenciatura en Ciencias Computacionales

M. en C. Fabiola Martínez Juárez
Correo: mjfabiola@uaeh.edu.mx

M. en C. Ma. Judith Franco Torres
Correo: mjfranco@uaeh.edu.mx


UAEH