El futuro digital es de todos

MinTIC

Conjuntos

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Conjuntos

Creación de un conjunto:

Loa conjuntos nos permiten identificar características del problema, que nos permitirán realizar procedimientos mas eficientes. Es una forma de identificar patrones y especificaciones de las variables de nuestro problema y nos permite conocerlo de una mejor forma.

Una vez identificamos un conjunto del problema, podemos crearlo en Python especificando sus elementos entre llaves:

```
In [1]: s = {1, 2, 3, 4}
```

Al igual que otras colecciones, sus miembros pueden ser de diversos tipos:

```
In [63]: s = {True, 3.14, None, False, "Hola mundo", (1, 2)}
```

No obstante, un conjunto no puede incluir objetos mutables como listas, diccionarios, e incluso otros conjuntos.

```
In [ ]: s = {[1, 2]}
```

Python distingue este tipo operación de la creación de un diccionario ya que no incluye dos puntos. Sin embargo, no puede dirimir el siguiente caso:

```
In [ ]: s = {}
```

Por defecto, la asignación anterior crea un diccionario. Para generar un conjunto vacío, directamente creamos una instancia de la clase set:

```
In [ ]: s = set()
```

De la misma forma podemos obtener un conjunto a partir de cualquier objeto iterable:

```
In [64]: s1 = set([1, 2, 3, 4])
         s2 = set(range(10))
         print(s1)
         print(s2)
```

```
{1, 2, 3, 4}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Un set puede ser convertido a una lista y viceversa. En este último caso, los elementos duplicados son unificados.

```
In [65]: list({1, 2, 3, 4})
Out[65]: [1, 2, 3, 4]

In [66]: set([1, 2, 2, 3, 4])
Out[66]: {1, 2, 3, 4}

In [67]: # Crea un conjunto con una serie de elementos entre llaves
         # Los elementos repetidos se eliminan
         c = {1, 3, 2, 9, 3, 1}
         print(c)

         {1, 2, 3, 9}

In [68]: # Crea un conjunto a partir de un string
         # Los caracteres repetidos se eliminan
         a = set('Hola Pythonista')
         print(a)

         {'t', 'n', 'o', 'l', 'a', 'y', ' ', 'P', 's', 'h', 'i', 'H'}

In [69]: # Crea un conjunto a partir de una lista
         # Los elementos repetidos de la lista se eliminan
         unicos = set([3, 5, 6, 1, 5])
         print(unicos)

         {1, 3, 5, 6}
```

Cómo acceder a los elementos de un conjunto en Python:

Dado que los conjuntos son colecciones desordenadas, en ellos no se guarda la posición en la que son insertados los elementos como ocurre en los tipos list o tuple. Es por ello que no se puede acceder a los elementos a través de un índice.
Sin embargo, sí se puede acceder y/o recorrer todos los elementos de un conjunto usando un bucle for:

```
In [70]: mi_conjunto = {1, 3, 2, 9, 3, 1}
         print(mi_conjunto)
         for numero in mi_conjunto:
             print(numero)
```

```
{1, 2, 3, 9}
1
2
3
9
```

Métodos

Los objetos de tipo conjunto mutable y conjunto inmutable integra una serie de métodos integrados a continuación:

add()

Este método agrega un elemento a un conjunto mutable. Esto no tiene efecto si el elemento ya esta presente.

```
In [71]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11}

In [72]: set_mutable1.add(22.000001)
         print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11, 22.000001}
```

clear()

Este método remueve todos los elementos desde este conjunto mutable.

```
In [73]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11}

In [74]: set_mutable1.clear()
         print(set_mutable1)

         set()
```

copy()

Este método devuelve una copia superficial del tipo conjunto mutable o conjunto inmutable:

```
In [75]: set_mutable = set([4.0, 'Carro', True])
         otro_set_mutable = set_mutable.copy()
         set_mutable == otro_set_mutable

Out[75]: True
```

difference()

Este método devuelve la diferencia entre dos conjunto mutable o conjunto inmutable: todos los elementos que están en el primero, pero no en el argumento.

```
In [76]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])

In [77]: print(set_mutable1)
         {1, 2, 3, 4, 5, 7, 11}

In [78]: print(set_mutable2)
         {2, 4, 5, 8, 9, 11}

In [79]: print(set_mutable1.difference(set_mutable2))
         {1, 3, 7}

In [80]: print(set_mutable2.difference(set_mutable1))
         {8, 9}
```

La diferencia también se puede generar de la siguiente manera:

```
In [81]: a = {1, 2, 3, 4}
         b = {2, 3}
         c = a - b
         print(c)
         {1, 4}
```

Dos conjuntos son iguales si y solo si contienen los mismos elementos:

```
In [82]: {1, 2, 3} == {3, 2, 1}
Out[82]: True

In [83]: {1, 2, 3} == {1, 2, 6}
Out[83]: False
```

difference_update():

Este método actualiza un tipo conjunto mutable llamando al método difference_update() con la diferencia de los conjuntos.

```
In [60]: proyecto1 = {'python', 'Zope2', 'ZODB3', 'pytz'}
         proyecto1

Out[60]: {'ZODB3', 'Zope2', 'python', 'pytz'}

In [61]: proyecto2 = {'python', 'Plone', 'diazo'}
         proyecto2

Out[61]: {'Plone', 'diazo', 'python'}

In [62]: proyecto1.difference_update(proyecto2)
         proyecto1

Out[62]: {'ZODB3', 'Zope2', 'pytz'}
```

Si proyecto1 y proyecto2 son dos conjuntos. La diferencia del conjunto proyecto1 y conjunto proyecto2 es un conjunto de elementos que existen solamente en el conjunto proyecto1 pero no en el conjunto proyecto2.

discard():

Este método remueve un elemento desde un conjunto mutable si esta presente.

```
In [57]: paquetes = {'python', 'zope', 'plone', 'django'}
         paquetes

Out[57]: {'django', 'plone', 'python', 'zope'}

In [58]: paquetes.discard('django')
         paquetes

Out[58]: {'plone', 'python', 'zope'}
```

El conjunto mutable permanece sin cambio si el elemento pasado como argumento al método discard() no existe.

```
In [59]: paquetes = {'python', 'zope', 'plone', 'django'}
         paquetes.discard('php')
         paquetes

Out[59]: {'django', 'plone', 'python', 'zope'}
```

intersection():

Este método devuelve la intersección entre los conjuntos mutables o conjuntos inmutables: todos los elementos que están en ambos.

```
In [51]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])

In [52]: print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11}

In [53]: print(set_mutable2)

         {2, 4, 5, 8, 9, 11}

In [54]: print(set_mutable1.intersection(set_mutable2))

         {2, 11, 4, 5}

In [55]: print(set_mutable2.intersection(set_mutable1))

         {2, 11, 4, 5}
```

También se puede expresar de la siguiente forma La intersección opera de forma análoga, pero con el operador **&**, y retorna un nuevo conjunto con los elementos que se encuentran en ambos.

```
In [56]: set_intersection = set_mutable1 & set_mutable2
         print(set_intersection)

         {2, 11, 4, 5}
```

intersection_update():

Este método actualiza un conjunto mutable con la intersección de ese mismo y otro conjunto mutable.

El método **intersection_update()** le permite arbitrariamente varios numero de argumentos (conjuntos).

La intersección de dos o mas conjuntos es el conjunto de elemento el cual es común a todos los conjuntos.

```
In [47]: proyecto1 = {'python', 'Zope2', 'pytz'}
         proyecto1

Out[47]: {'Zope2', 'python', 'pytz'}

In [48]: proyecto2 = {'python', 'Plone', 'diazo', 'z3c.form'}
         proyecto2

Out[48]: {'Plone', 'diazo', 'python', 'z3c.form'}

In [49]: proyecto3 = {'python', 'django', 'django-filter'}
         proyecto3

Out[49]: {'django', 'django-filter', 'python'}

In [50]: proyecto3.intersection_update(proyecto1, proyecto2)
         proyecto3

Out[50]: {'python'}
```

isdisjoint():

Este método devuelve el valor True si no hay elementos comunes entre los conjuntos mutables o conjuntos inmutables.

```
In [42]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])

In [43]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])

In [44]: print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11}

In [45]: print(set_mutable2)

         {2, 4, 5, 8, 9, 11}

In [46]: print(set_mutable1.isdisjoint(set_mutable2))

         False
```

issubset():

Este método devuelve el valor True si el conjunto mutable es un subconjunto del conjunto mutable o del conjunto inmutable argumento.

```
In [36]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])
         set_mutable3 = set([11, 5, 2, 4])

In [37]: print( set_mutable1)
         {1, 2, 3, 4, 5, 7, 11}

In [38]: print(set_mutable2)
         {2, 4, 5, 8, 9, 11}

In [39]: print(set_mutable3)
         {2, 11, 4, 5}

In [40]: print(set_mutable2.issubset(set_mutable1))
         False

In [41]: print(set_mutable3.issubset(set_mutable1))
         True
```

issuperset():

Este método devuelve el valor True si el conjunto mutable o el conjunto inmutable es un superset del conjunto mutable argumento.

```
In [30]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])
         set_mutable3 = set([11, 5, 2, 4])

In [31]: print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11}

In [32]: print(set_mutable2)

         {2, 4, 5, 8, 9, 11}

In [33]: print(set_mutable3)

         {2, 11, 4, 5}

In [34]: print(set_mutable1.issuperset(set_mutable2))

         False

In [35]: print(set_mutable1.issuperset(set_mutable3))

         True
```

pop():

Este método remueve arbitrariamente y devuelve un elemento de conjunto mutable. El método **pop()** no toma ningún argumento. Si el conjunto mutable esta vacío se lanza una excepción **KeyError**.

Debemos tener en cuenta que usted podría obtener diferente salida devueltas usando el método **pop()** por que remueve aleatoriamente un elemento.

```
In [21]: paquetes = {'python', 'zope', 'plone', 'django'}
         paquetes
Out[21]: {'django', 'plone', 'python', 'zope'}

In [22]: print("Valor aleatorio devuelto es:", paquetes.pop())
         Valor aleatorio devuelto es: zope

In [23]: print(paquetes)
         {'python', 'plone', 'django'}

In [24]: print("Valor aleatorio devuelto es:", paquetes.pop())
         Valor aleatorio devuelto es: python

In [25]: paquetes
Out[25]: {'django', 'plone'}

In [26]: print("Valor aleatorio devuelto es:", paquetes.pop())
         Valor aleatorio devuelto es: plone

In [27]: paquetes
Out[27]: {'django'}

In [28]: print( "Valor aleatorio devuelto es:", paquetes.pop())
         Valor aleatorio devuelto es: django
```

Remove():

Este método busca y remueve un elemento de un conjunto mutable, si debe ser un miembro.

```
In [19]: paquetes = {'python', 'zope', 'plone', 'django'}
         paquetes

Out[19]: {'django', 'plone', 'python', 'zope'}

In [20]: paquetes.remove('django')
         paquetes

Out[20]: {'plone', 'python', 'zope'}
```

Si el elemento no es existe en el conjunto mutable, lanza una excepción **KeyError**. Usted puede usar el método **discard()** si usted no quiere este error. El conjunto mutable permanece sin cambio si el elemento pasado al método **discard()** no existe.

Un conjunto es una colección desordenada de elementos. Si usted quiere remover arbitrariamente elemento un conjunto, usted puede usar el método **pop()**.

Symmetric_difference():

Este método devuelve todos los elementos que están en un conjunto mutable e conjunto inmutable u otro, pero no en ambos.

```
In [15]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
         set_mutable2 = set([11, 5, 9, 2, 4, 8])

In [16]: print(set_mutable1)

         {1, 2, 3, 4, 5, 7, 11}

In [17]: print(set_mutable2)

         {2, 4, 5, 8, 9, 11}

In [18]: print(set_mutable1.symmetric_difference(set_mutable2))

         {1, 3, 7, 8, 9}
```

Symmetric_difference_update():

Este método actualiza un conjunto mutable llamando al método **symmetric_difference_update()** con los conjuntos de diferencia simétrica.

La diferencia simétrica de dos conjuntos es el conjunto de elementos que están en cualquiera de los conjuntos pero no en ambos.

```
In [12]: proyecto1 = {'python', 'plone', 'django'}
         proyecto1

Out[12]: {'django', 'plone', 'python'}

In [13]: proyecto2 = {'django', 'zope', 'pyramid'}
         proyecto2

Out[13]: {'django', 'pyramid', 'zope'}

In [14]: proyecto1.symmetric_difference_update(proyecto2)
         proyecto1

Out[14]: {'plone', 'pyramid', 'python', 'zope'}
```

union():

Este método devuelve un conjunto mutable y conjunto inmutable con todos los elementos que están en alguno de los conjuntos mutables y conjuntos inmutables.

```
In [2]: set_mutable1 = set([4, 3, 11, 7, 5, 2, 1, 4])
        set_mutable2 = set([11, 5, 9, 2, 4, 8])
        print(set_mutable1)
```

```
{1, 2, 3, 4, 5, 7, 11}
```

```
In [3]: print(set_mutable2)
        print(set_mutable1.union(set_mutable2))
```

```
{2, 4, 5, 8, 9, 11}
{1, 2, 3, 4, 5, 7, 8, 9, 11}
```

La unión también se puede realizar con el caracter | y retorna un conjunto que contiene los elementos que se encuentran en al menos uno de los dos conjuntos involucrados en la operación.

```
In [4]: a = {1, 2, 3, 4}
        b = {3, 4, 5, 6}
        a | b

Out[4]: {1, 2, 3, 4, 5, 6}
```

update():

Este método agrega elementos desde un conjunto mutable (pasando como un argumento) un tipo tupla, un tipo lista, un tipo diccionario o un tipo conjunto mutable llamado con el método update().

A continuación un ejemplo de agregar nuevos elementos un tipo conjunto mutable usando otro tipo conjunto mutable:

```
In [5]: version_plone_dev = set([5.1, 6])
        version_plone_dev

Out[5]: {5.1, 6}

In [6]: versiones_plone = set([2.1, 2.5, 3.6, 4])
        versiones_plone

Out[6]: {2.1, 2.5, 3.6, 4}

In [7]: versiones_plone.update(version_plone_dev)
        versiones_plone

Out[7]: {2.1, 2.5, 3.6, 4, 5.1, 6}
```

A continuación un ejemplo de agregar nuevos elementos un tipo conjunto mutable usando otro tipo cadena de caracteres:

```
In [8]: cadena = 'abc'
        cadena

Out[8]: 'abc'

In [9]: conjunto = {1, 2}
        conjunto.update(cadena)
        conjunto

Out[9]: {1, 2, 'a', 'b', 'c'}
```

A continuación un ejemplo de agregar nuevos elementos un tipo conjunto mutable usando otro tipo diccionario:

```
In [10]: diccionario = {'key': 1, 2:'lock'}
         diccionario.items()

Out[10]: dict_items([('key', 1), (2, 'lock')])

In [11]: conjunto = {'a', 'b'}
         conjunto.update(diccionario)
         conjunto

Out[11]: {2, 'a', 'b', 'key'}
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022