Universidad del Rosario | Escuela de Ingeniería, Ciencia y Tecnología | DEPE

ESTRUCTURAS DE DATOS EN PYTHON

PARA MANIPULACIÓN DE DATOS

SÁBADO
2 DE MARZO | 10:00 AM A 12:00 PM

A lo largo de esta sesión, aprenderás sobre listas, tuplas, diccionarios y conjuntos en Python, y cómo utilizar estas estructuras para gestionar y analizar datos de manera eficiente

Taller Knuth

MODALIDAD ACCESO REMOTO

NO SE REQUIERE EXPERIENCIA PREVIA EN PROGRAMACIÓN

Universidad del Rosario | Escuela de Ingeniería, Ciencia y Tecnología | MACC

Bienvenida

Sobre el conferencista

Miguel Angel Orjuela Rocha

Ing. de Sistemas y Computación

Skills:

- Desarrollo Fullstack
- Analítica de datos
- Transferencia de conocimiento

Contacto:

miguela.orjuela@urosario.edu.co

https://www.linkedin.com/in/miguel-orjuela/

https://github.com/maorjuela73

Contenido

- Entorno de desarrollo
- Tuplas
- Listas
- Funciones de secuencia
- Diccionarios
- Conjuntos
- Listas, conjuntos y diccionarios por comprensión

Entorno de desarrollo

Herramientas a usar

Entorno de desarrollo

Intérprete de Python

- https://www.python.org/

Entornos Integrados de Desarrollo (**IDEs**)

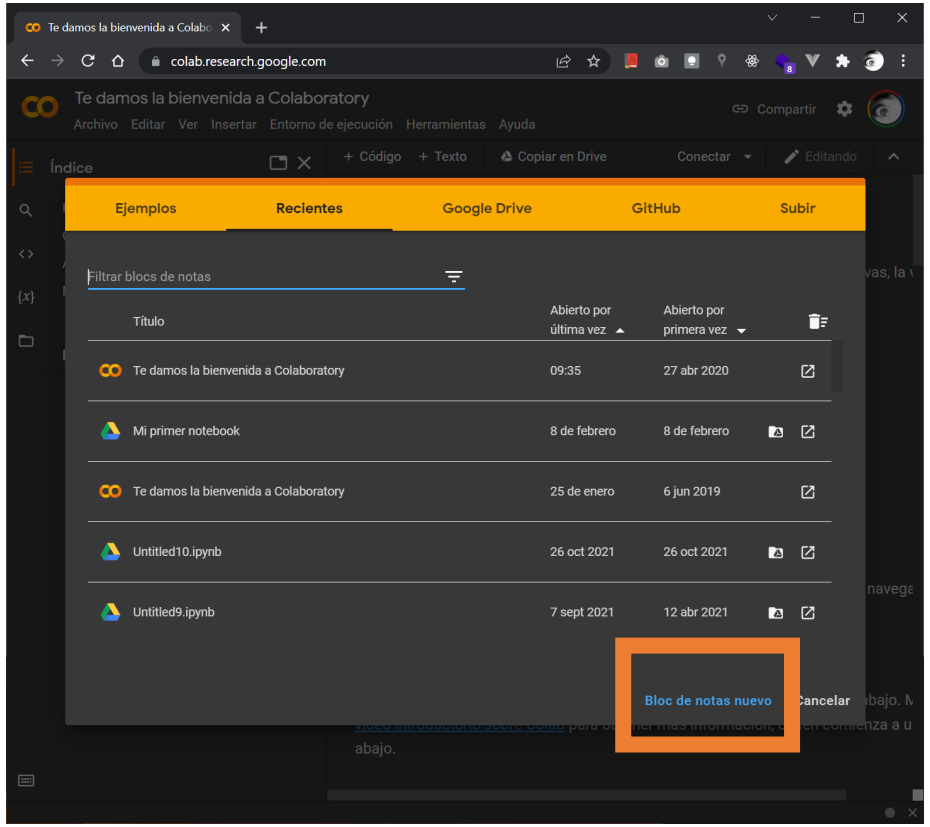
- https://www.spyder-ide.org/
- https://jupyter.org/try
- https://www.jetbrains.com/es-es/pycharm/
- https://code.visualstudio.com/
- https://www.sublimetext.com/
- …

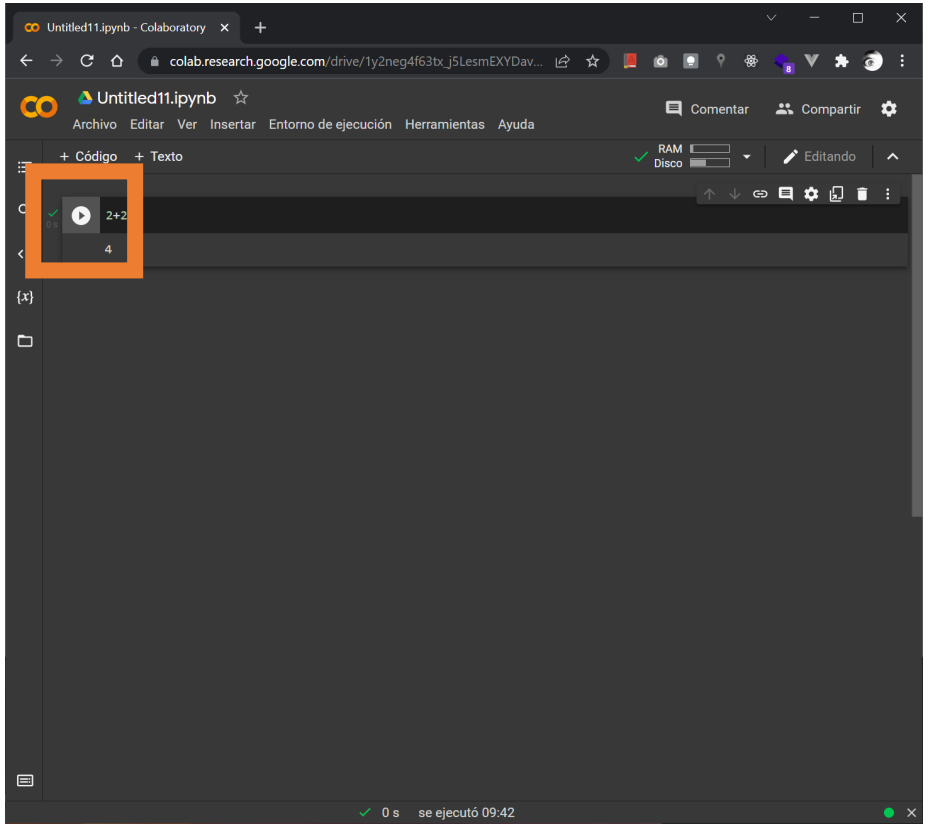
Entornos Cloud

- https://colab.research.google.com/

Ejecución de código en Colab

- Los blocs de notas funcionan con celdas
- Para ejecutar código de una celda se da “Play”





Imprimir datos

- Imprime en pantalla el valor de una expresión

```
mensaje = "Hola mundo"
print (mensaje)
print(type(mensaje))

n = 10
print(n)
print(f"El número es {n}")
print(type(n))

pi = 3.141592
print(pi)
print(type(pi))
```

- Es posible convertir un dato a otro tipo

```
int(pi)
float(15)
float("2.7182")
```

Tuplas

Tuplas

Es una secuencia de tamaño fijo

Es inmutable

Es una secuencia de valores separados por comas

```
tupla =(4, 5, 6)
```

```
tupla =4, 5, 6
```

Se pueden crear tuplas con tuplas adentro, hay que usar paréntesis para esto

```
tupla_anidada = (4, 5, 6), (7, 8)
```

Tuplas

```
tupla_str = tuple('string')
```

Para consultar los elementos de una tupla se usan los corchetes

Las secuencias están indexadas desde cero

```
tupla_str[0]
tupla_str[1]
tupla_str[2]
tupla_str[3]
tupla_str[4]
tupla_str[5]
```

Tuplas

Una vez la tupla se crea, no se pueden modificar los objetos de cada espacio

```
tupla4 = tuple(['Hola', [1,2], True])

tupla4[2] = False
```

Pero si un objeto dentro de una tupla es mutable, se puede modificar

```
tupla4[1].append(3)
```

Tuplas

Se pueden concatenar tuplas para hacer tuplas más largas

El operador de concatenar es el +

```
(4, None, 'Hola') + (6, 0) + ('Adios', )
```

Multiplicar una tuple por un entero es similar a concatenar varias veces una tupla

```
(6, 0) * 4
```

Tuplas

Desempacar una tupla

Asignar los valores dentro de una tuple a una variable

```
tupla = 4, 5, 6
a, b, c = tupla
```

Inclusive tuplas con tuplas anidadas se pueden desempacar

```
tupla5 = 4 ,5 (6, 7)
a, b, (c, d) = tupla5
```

Tuplas

En la mayoría de lenguajes, cuando se quiere intercambiar nombres de variables, hay que usar una variable temporal

```
a = 10; b = 5;
temp = a
a = b
b = temp
```

Usando el desempacado de tuplas se puede hacer fácilmente

```
a, b = 10, 5
b, a = a, b
```

Tuplas

Uso práctico: iterar sobre secuencias de tuplas o listas

```
seq = [(1, 2, 3),(4, 5, 6),(7, 8, 9)]

for a, b, c in seq:
    print(f'a={0}, b={1}, c={2}')
```

El arreglo seq tiene tres elementos (que son tuplas)

Cada elemento es despempacado en las variables a, b y c

Tuplas

Pluck: Sacar

En ocasiones queremos sacar elementos del comienzo de una tuple. Para eso usamos la síntaxis *rest

```
valores =  1, 2, 3, 4, 5
a, b, *rest = valores
```

```
a, b
```

```
rest
```

Como convención, las variables no deseadas se marcan con barra al piso (_)

```
a, b, *_ = valores
```

Tuplas

Métodos de tuplas:

Como son inmutables tienen pocos

count() cuenta el número de ocurrencias de un valor

```
a = 1,2,2,2,3,4,2

a.count(2)
```

¿Cuántas veces está el 2 en la tupla?

Listas

Listas

Son de longitud variable

Su contenido puede ser modificado

Se definen con corchete cuadrado [ ] o usando la función list

```
lista_a = [2, 3, 7, None]

tupla = ('uno', 'two', 'tres')
lista_b = list(tupla)
lista_b[1] = 'dos'
```

Se usa para convertir un iterador en lista

```
gen = range(10)
```

```
list(gen)
```

Listas

Se agregan elementos con append()

insert() permite insertar un elemento dado un índice

```
lista_b
lista_b.append('cuatro')
lista_b
lista_b.insert(1, 'uno y medio')
```

El inverso de insert() es pop(). Se quitan elementos con pop() usando el índice

```
lista_b.pop(3)
```

remove() busca el primer elemento cuyo valor es igual al pasado por parámetro uy lo borra

```
lista_b.remove('uno')
```

Listas

Para revisar si un elemento está en una lista

```
'dos' in lista_b
```

```
'dos' not in lista_b
```

Para concatenar y combinar listas, se hace de forma similar a las tuplas (operador +)

```
[4, None, 'foo'] + [7, 8, (2, 3)]
```

Si ya existe el arreglo, se pueden agregar elementos múltiples con el comando extent

```
x = [4, None, 'foo']
x.extend([7, 8, (2, 3)])
```

Más eficiente

Listas

Ejercicio: Listado de invitados

1. Si pudieras invitar a un grupo de personas cualquiera a cenar ¿a quien invitarías? Haz una lista llamada **invitados** que incluya por lo menos a tres personas. Luego, usa esa lista para imprimir el mensaje de invitación a la cena para cada uno.
2. Alguien de la lista no puede ir. Imprime el nombre de la persona que no puede ir y luego modifica la lista para reemplazar el nombre de quien no puede ir con el de un nuevo invitado.
3. Ahora hay una mesa más grande de reserva. Usa las funciones insert y append para agregar nuevos invitados, uno al comienzo, otro en la mitad y otro al final de la lista.
4. Mesa para dos. Remueve invitados hasta que queden únicamente dos en la lista.

Listas

Ordenando

Se puede llamar la función sort() para ordenar elementos del arreglo

```
a = [7, 2, 5, 1, 3]
a.sort()
```

Se le puede pasas a sort() una llave de ordenamiento. P.ej. Ordenar un arreglo de strings por longitud de cada string

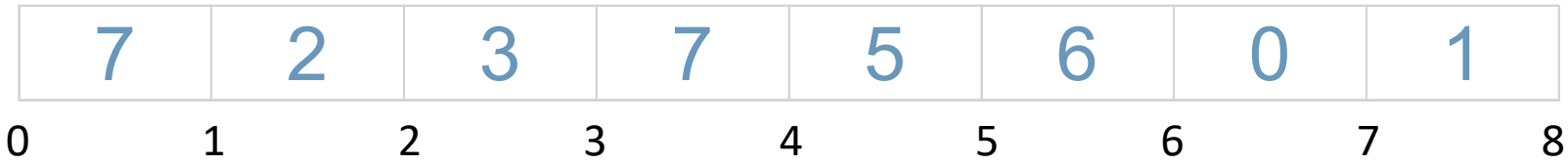
```
b = ['saw', 'small', 'He', 'foxes', 'six']
b.sort(key=len)
```

Listas

Slicing: Sacar tajadas

En el operador de índice, se introducen valores de inicio:fin

```
seq = [7, 2, 3, 7, 5, 6, 0, 1]
seq[1:5]
```



¿Qué hace la asignación ?

```
seq[3:4] = [6, 3]
```

Listas

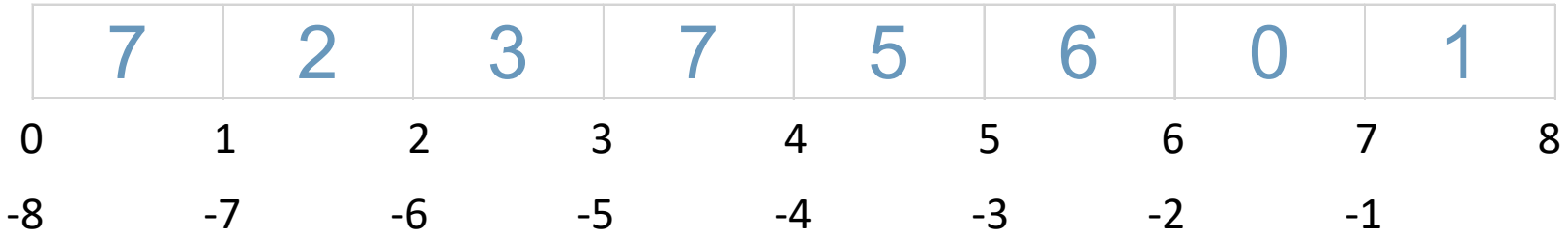
Slicing: Sacar tajadas

Se puede omitir el índice de inicio o de fin

```
seq[3:]
```

```
seq[:5]
```

Se pueden usar también índices negativos




```
seq = [7, 2, 3, 7, 5, 6, 0, 1]
seq[-6:-3]
```

Listas

Slicing: Sacar tajadas

Se puede agregar un tercer valor en el índice para indicar saltos

Listas

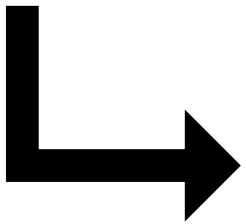
Funciones de secuencia

- enumerate()

Es para llevar información del índice de la iteración

```
coleccion = [2,3,6,44,12,9]
```

```
i = 0
for valor in coleccion:
    print('Este es el ciclo {0}'.format(i))
    i += 1
```



```
for i, valor in enumerate(coleccion):
    print('Este es el ciclo {0}'.format(i))
```

Listas

Mapear valores a un diccionario

Cuando está indexando datos, un patrón útil que utiliza enumerate es calcular un **dict** que asigna los valores de una secuencia (que se supone que son únicos) a sus ubicaciones en la secuencia

```
una_lista = ['Andres', 'Beatriz', 'Carlos']

mapping = {}

for indice, valor in enumerate(una_lista):
    mapping[valor] = indice
```

```
mapping['Andres']
```

Listas

sorted() retorna una nueva lista ordenada con los elementos de cualquier secuencia

```
sorted((1,4,8,2))
sorted([2,3,4,7,4,1,2,8,9])
sorted('buenas tardes')
```

zip() empareja elementos de listas o tuplas para crear una lista de tuplas

```
seq1 = ['uno', 'dos', 'tres']
seq2 = ['a' , 'be' , 'ce' ]

lista_de_tuplas = zip(seq1, seq2)
list(lista_de_tuplas)
```

Listas

zip() puede tomar muchas secuencias, y la lista que produce está deteminada por la secuencia más corta

```
seq1 = ['uno', 'dos', 'tres']
seq2 = ['a', 'be', 'ce']
seq3 = [False, True]

lista_de_tuplas = zip(seq1, seq2, seq3)
list(lista_de_tuplas)
```

Un uso muy común de zip() es iterar simultáneamente en múltiples secuencias, posiblemente también combinado con enumerate

```
seq1 = ['uno', 'dos', 'tres']
seq2 = ['a', 'be', 'ce']
for i, (a, b) in enumerate(zip(seq1, seq2)):
    print('i={0} a={1} b={2}'.format(i, a, b))
```

Listas

Para hacer unzip de una secuencia

```
nombres_completos = [('Andres', 'Agudelo'),
                     ('Brayan', 'Beltran'),
                     ('Camila','Cruz')]

nombres, apellidos = zip(*nombres_completos)
```

reverse se usa para invertir el orden de una lista

```
list(reversed(range(10)))
```

Diccionarios

Diccionarios

Diccionario, *hash map* o array asociativo

Estructura de datos más importante

Colección de tamaño flexible de parejas llave-valor, donde la llave y el valor son objetos de Python

Se crean con corchetes {}

Las llaves se separan de los valores con dos puntos

```
diccionario_vacio = {}

d1 = {'a' : 'algun valor', 'b' : [1, 2, 3, 4]}
```

Diccionarios

Se pueden acceder o insertar elementos usando la misma síntaxis con la que se acceden elementos de un arreglo o tupla:

```
d1 = {'a' : 'algun valor', 'b' : [1, 2, 3, 4]}

d1[7] = 'un entero'
d1['b']
```

Se pueden preguntar si una llave está en un diccionario:

```
'a' in d1
```

Diccionarios

Se borran valores con la palabra del o con el método pop()

```
d1 = {'a' : 'algun valor', 'b' : [1, 2, 3, 4]}

d1[7] = 'un entero'
d1[5] = 'algun valor'
d1['dummy'] = 'otro valor'
```

```
d1
del d1[5]
d1
```

```
a_borrar = d1.pop('dummy')
a_borrar
d1
```

Diccionarios

Es posible pedir una lista con las llaves y otra lista con los valores del diccionario

Las listas retornadas por ambos tiene el mismo orden

```
list(d1.keys())
list(d1.values())
```

Se pueden unir dos diccionarios con el método update

```
d1
d1.update({'b': 'un nuevo valor', 'c': 12})
d1
```

Diccionarios

```
lista_llaves = ['llave1','llave2','llave3']
lista_valores = ['valor1','valor2','valor3']
```

Creando diccionarios de secuencias

```
resultado = {}
for llave, valor in zip(lista_llaves, lista_valores):
 resultado[llave] = valor
```

Un diccionario es una colección de 2-tuplas, por eso la función dict acepta como parámetro una lista de 2-tuplas

```
resultado _v2 = dict(zip(lista_llaves, lista_valores))
```

Diccionarios

Ejercicio:

Usa un diccionario para almacenar los números favoritos de las personas. Piensa en cinco nombres y utilízalos como llaves en tu diccionario. Piensa en un número favorito para cada persona y guárdelo como un valor en su diccionario. Imprime el nombre de cada persona y su número favorito.

Conjuntos

Conjuntos

Conjunto: Colección sin orden de elementos únicos

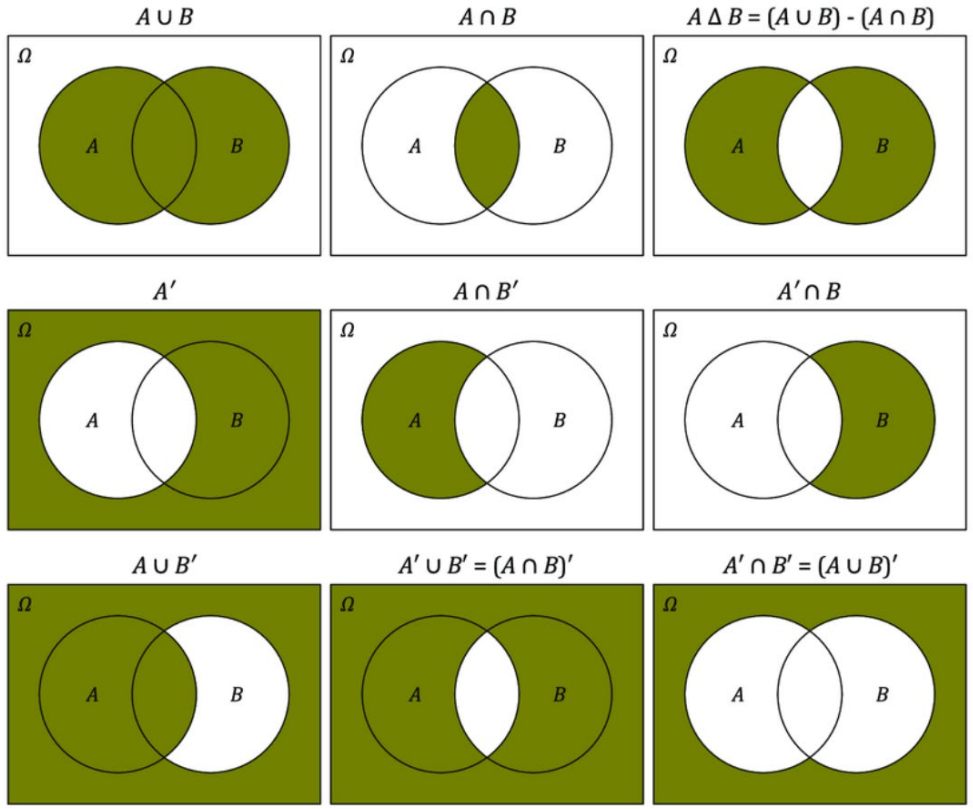
Son como dicts pero solo llaves, sin valores

Se pueden declara con la función set o con corchetes {}

```
set([2, 2, 2, 1, 3, 3])
```

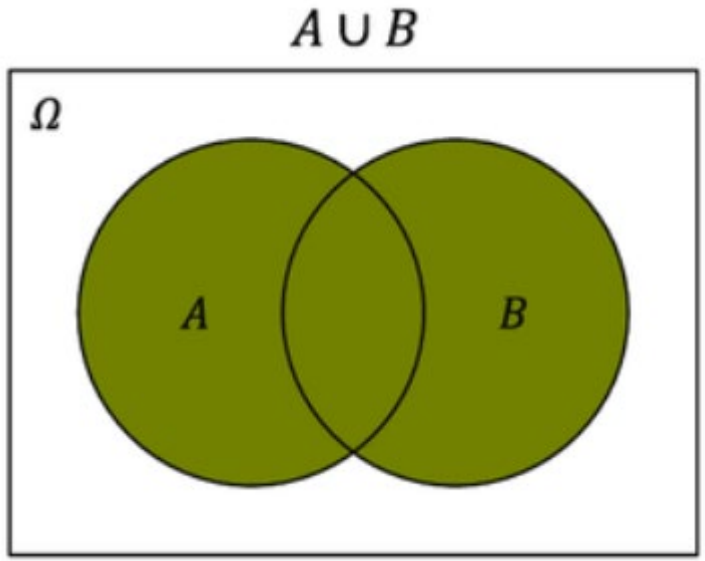
```
{2, 2, 2, 1, 3, 3}
```

Soportan operaciones matemáticas de conjuntos como union, intersección, diferencia y diferencia simétrica

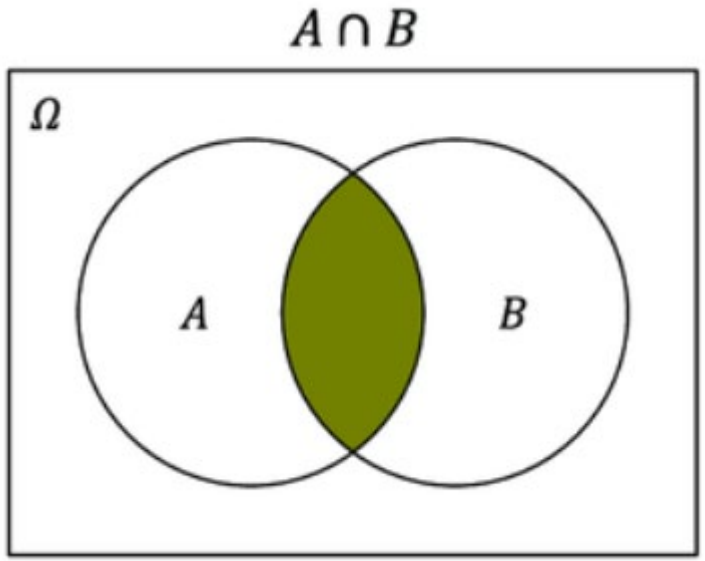


Conjuntos

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}
```




```
a.union(b)
a | b
```




```
a.intersection(b)
a & b
```

Conjuntos

Function	Alternative syntax	Description
`a.add(x)`	N/A	Add element `x` to the set `a`
`a.clear()`	N/A	Reset the set `a` to an empty state, discarding all of its elements
`a.remove(x)`	N/A	Remove element `x` from the set `a`
`a.pop()`	N/A	Remove an arbitrary element from the set `a`, raising `KeyError` if the set is empty
`a.union(b)`	`a | b`	All of the unique elements in `a` and `b`
`a.update(b)`	`a |= b`	Set the contents of `a` to be the union of the elements in `a` and `b`
`a.intersection(b)`	`a & b`	All of the elements in *both* `a` and `b`
`a.intersection_update(b)`	`a &= b`	Set the contents of `a` to be the intersection of the elements in `a` and `b`
`a.difference(b)`	`a - b`	The elements in `a` that are not in `b`
`a.difference_update(b)`	`a -= b`	Set `a` to the elements in `a` that are not in `b`
`a.symmetric_difference(b)`	`a ^ b`	All of the elements in either `a` or `b` but *not both*
`a.symmetric_difference_update(b)`	`a ^= b`	Set `a` to contain the elements in either `a` or `b` but *not both*
`a.issubset(b)`	N/A	`True` if the elements of `a` are all contained in `b`
`a.issuperset(b)`	N/A	`True` if the elements of `b` are all contained in `a`
`a.isdisjoint(b)`	N/A	`True` if `a` and `b` have no elements in common

```
c= a.copy()
c |= b
c
```

```
d = a.copy()
d &= b
d
```

Conjuntos

```
a_set = {1, 2, 3, 4, 5}
```

Un conjunto está contenido en (es subset)

```
{1, 2, 3}.issubset(a_set)
```

Un conjunto contiene todos los elementos de (es superset)

```
a_set.issuperset({1, 2, 3})
```

Los conjuntos son iguales solo si el contenido de ambos es igual

```
{1, 2, 3} == {3, 2, 1}
```

Listas, conjuntos y diccionarios por comprensión

Listas, conjuntos y diccionarios por comprensión

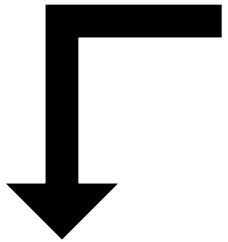
Una de las características más queridas de Python

Permite formar una nueva lista de forma concisa al filtrar los elementos de una colección, transformando los elementos que pasan el filtro en una expresión concisa.

Forma de uso:

Lógica

```
result = []
for val in collection:
    if condition:
        result.append(expr)
```



Implementación

```
[expr for val in collection if condition]
```

Listas, conjuntos y diccionarios por comprensión

Ejemplo

```
squares = [value**2 for value in range(1,
11)]
print(squares)
```

Listas, conjuntos y diccionarios por comprensión

```
[expr for val in collection if condition]
```

Ejemplo:

```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
```

¿Qué lista se produce con el siguiente comando?

```
[x.upper() for x in strings if len(x) > 2]
```

Listas, conjuntos y diccionarios por comprensión

Los conjuntos y diccionarios se crean de forma similar

Listas

```
[expr for val in collection if condition]
```

Diccionarios

```
{key-expr:value-expr for val in collection if condition}
```

Conjuntos

```
{expr for val in collection if condition}
```

Listas, conjuntos y diccionarios por comprensión

```
strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
```

Suponga que queremos conocer la longitud de los strings que se encuentran en la lista strings

```
long_unicas = {len(x) for x in strings}
```

Se puede expresar de forma más funcional con la función map()

```
set(map(len, strings))
```

Listas, conjuntos y diccionarios por comprensión

Creando un diccionario por comprensión

```
loc_mapping = {val : index for index, val in enumerate(strings)}
```

Actividad:

Construya una lista que contenga todos los números primos entre 1 y 100

- Usando ciclos for y declaraciones condicionales
- Usando listas por comprensión

Gracias por tu asistencia y participación ☺

Contacto

miguela.orjuela@urosario.edu.co
https://www.linkedin.com/in/miguel-orjuela/
https://github.com/maorjuela73

Links de interés

- *https://nostarch.com/pythoncrashcourse2e*