

A black and white photograph of a statue of a religious figure, likely a saint, wearing a long robe and holding a book. The statue is positioned on the left side of the slide, with its base visible at the bottom.

# INTRODUCCIÓN A LA PROGRAMACIÓN EN PYTHON

■ Miguel Orjuela



Universidad del  
**Rosario**

Educación  
Continua

Personas con  
*propósito*  
que ayudan a  
transformar vidas



Universidad del  
**Rosario**

Educación  
Continua

# **Introducción a la programación en Python**

2019

Sesión # 3

# Estructuras de datos



Miguel Angel Orjuela Rocha

Ingeniero de Sistemas y Computación

# Contenido

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

# Tuplas

# Tuplas

Es una secuencia de tamaño fijo

Es inmutable

Es una secuencia de valores separados por comas

```
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

Se puede convertir una secuencia o iterador en tupla

```
tupla2 = tuple([4, 0, 2])
```

```
tupla3 = tuple('string')
```

Para consultar los elementos de una tupla se usan los corchetes cuadrados

Las secuencias están indexadas desde cero

```
tupla3[0]  
tupla3[1]  
tupla3[2]  
tupla3[3]  
tupla3[4]  
tupla3[5]
```

# Tuplas

Una vez la tuple 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('a={0}, b={1}, c={2}'.format(a, b, c))
```

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 sintaxis `*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 y 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 extend

Más eficiente

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

# Listas

Ejercicio: Modelar un tablero de ajedrez con arreglos, de tal forma que podamos recuperar el color de la casilla dada su posición en  $x$  y  $y$



tablero[0][0] es negra

tablero[7][7] es negra

tablero[4][3] es blanca

# Listas

Solución:

```
tablero = []

for i in range(8):
    tablero.append([])
    for j in range(8):
        tablero[i].append('Negro' if (i + j) % 2 == 0 else 'Blanco')

tablero[0][0]
tablero[7][7]
tablero[4][3]
```

# Listas

## Ordenando

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

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

Se le puede pasar 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)
```

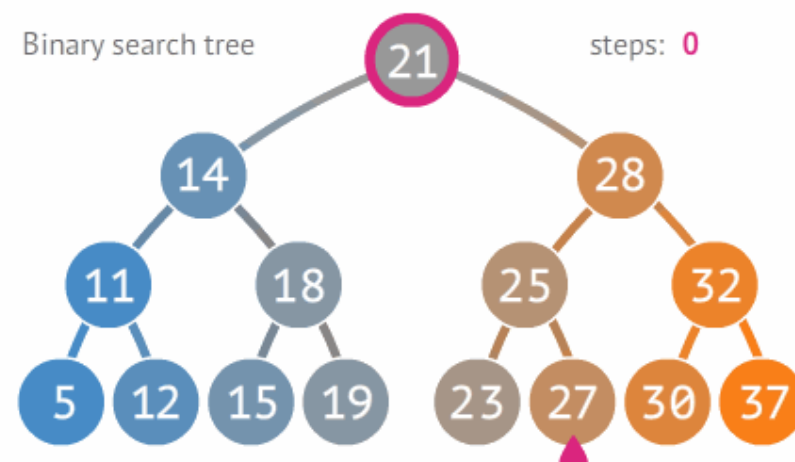
# Listas

El módulo `bisect` implementa búsqueda e inserción binaria

`bisect()` busca el espacio donde debería ir el element

`insort()` adiciona el elemento en la ubicación

**NOTA:** usar solo con arreglos ordenados



```
import bisect
```

```
c = [1, 2, 2, 2, 3, 4, 7]
```

```
bisect.bisect(c, 2)
```

```
bisect.bisect(c, 5)
```

```
bisect.insort(c, 6)
```

# 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]
```

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 3 | 7 | 5 | 6 | 0 | 1 |   |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

¿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

|    |    |    |    |    |    |    |    |   |
|----|----|----|----|----|----|----|----|---|
| 7  | 2  | 3  | 7  | 5  | 6  | 0  | 1  |   |
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8 |
| -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |   |

```
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

```
seq[::2]
```

Saltos de 2 en 2

```
seq[::-1]
```

Reversando la lista

# Listas

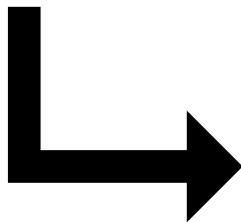
Funciones de secuencia

- `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 determinada 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' : 'algún valor', 'b' : [1, 2, 3, 4]}
```



# Diccionarios

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

```
d1 = {'a' : 'algún 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' : 'algún valor', 'b' : [1, 2, 3, 4]}  
  
d1[7] = 'un entero'  
d1[5] = 'algún 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

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

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

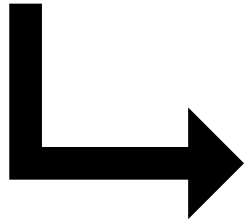
```
mapping_v2 = dict(zip(lista_llaves, lista_valores))
```

# Diccionarios

## Valores por defecto

Lógica

```
if llave in algun_diccionario:  
    valor = algun_diccionario[llave]  
else:  
    valor = valor_default
```



Implementación

```
valor = algun_diccionario.get(llave, valor_default)
```

NOTA: Si el get no tiene valor por defecto, retorna un None

# Diccionarios

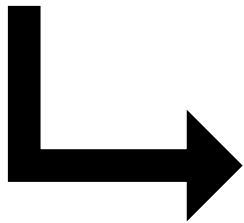
Poner valores por defecto puede ser una funcionalidad muy útil  
Ejemplo:

```
palabras = ['amarillo', 'bate', 'bosque', 'ahorro', 'burro']
```

Lógica

```
por_letra = {}  
for palabra in palabras:  
    inicial = palabra[0]  
    if inicial not in por_letra:  
        por_letra[inicial] = [palabra]  
    else:  
        por_letra[inicial].append(palabra)
```

Implementación



```
por_letra_v2 = {}  
for palabra in palabras:  
    inicial = palabra[0]  
    por_letra_v2.setdefault(inicial, []).append(palabra)
```

# Diccionarios

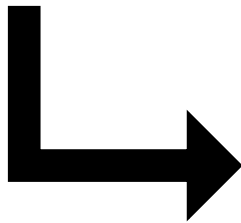
Poner valores por defecto puede ser una funcionalidad muy útil  
Ejemplo:

```
palabras = ['amarillo', 'bate', 'bosque', 'ahorro', 'burro']
```

Lógica

```
por_letra = {}  
for palabra in palabras:  
    inicial = palabra[0]  
    if inicial not in por_letra:  
        por_letra[inicial] = [palabra]  
    else:  
        por_letra[inicial].append(palabra)
```

Implementación



```
from collections import defaultdict  
por_letra_v3 = defaultdict(list)  
for palabra in palabras:  
    por_letra_v3[palabra[0]].append(palabra)
```

# Diccionarios

Hashabilidad: Las llaves del diccionario tienen que ser inmutables de tipo escalar (int, float, string) o tuplas (los objetos dentro de la tupla deben ser inmutables también)

```
hash('string')  
hash((1,2,(2,3)))  
hash((1,2,[2,3]))
```

¿cuál de estos tres no funciona y por qué?

Para usar una lista como llave, hay que convertirla en tupla, la cual será hashable si sus elementos lo son

```
d = {}  
d[tuple([1, 2, 3])] = 5
```



# Conjuntos

# Conjuntos

Conjunto: Colección sin orden de elementos únicos

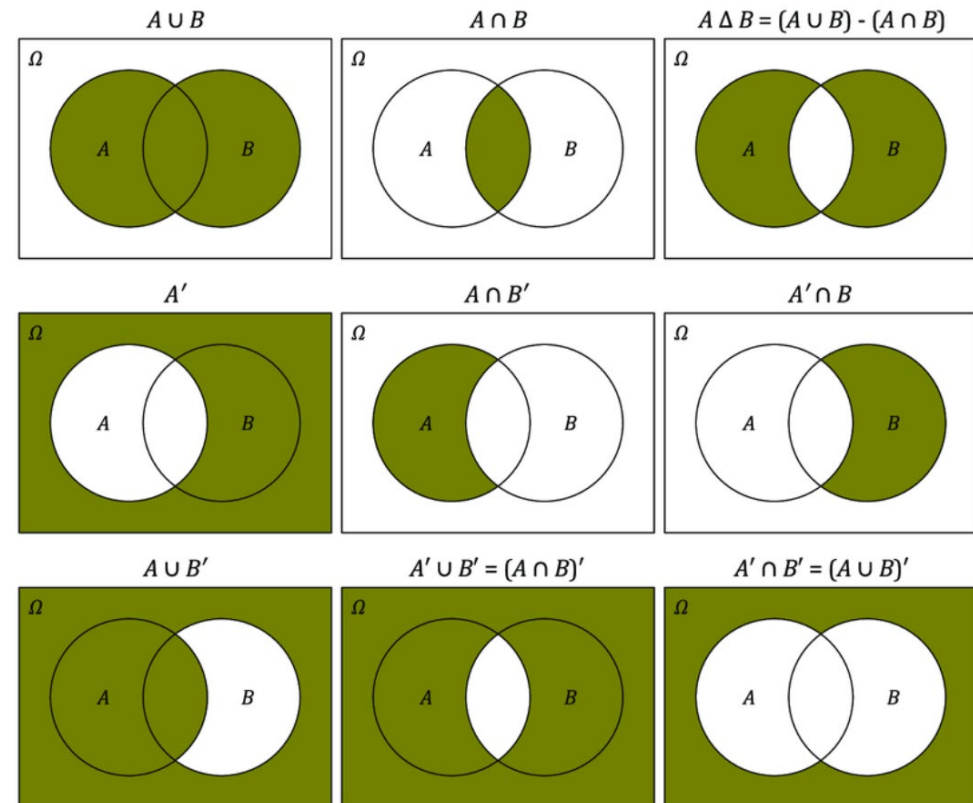
Son como **dicts** pero solo llaves, sin valores

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

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

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

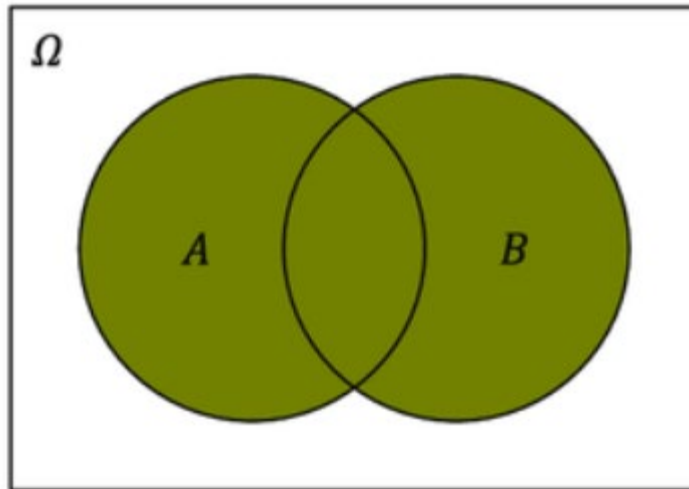
Soportan operaciones matemáticas de conjuntos como union, intersección, diferencia y diferencia simétrica



# Conjuntos

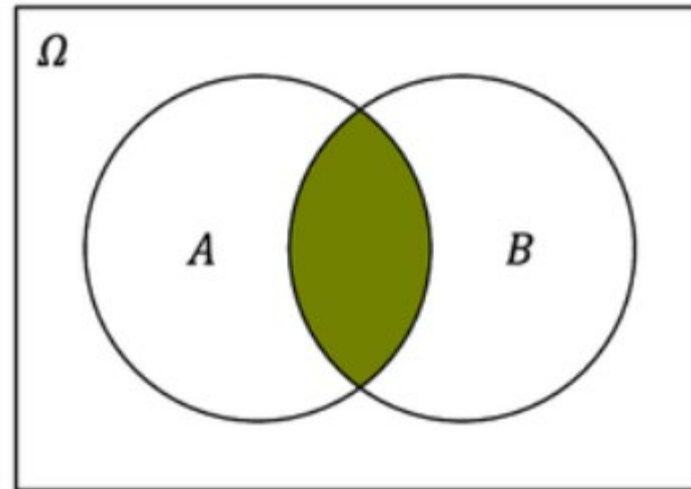
```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}
```

$A \cup B$



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

$A \cap B$



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

# Conjuntos

| Function                                      | Alternative syntax      | Description  |
|---|-------------------------|--|
| <code>a.add(x)</code>                         | N/A                     | Add element <code>x</code> to the set <code>a</code>   |
| <code>a.clear()</code>                        | N/A                     | Reset the set <code>a</code> to an empty state, discarding all of its elements                                 |
| <code>a.remove(x)</code>                      | N/A                     | Remove element <code>x</code> from the set <code>a</code>  |
| <code>a.pop()</code>                          | N/A                     | Remove an arbitrary element from the set <code>a</code> , raising <code>KeyError</code> if the set is empty    |
| <code>a.union(b)</code>                       | <code>a   b</code>      | All of the unique elements in <code>a</code> and <code>b</code>  |
| <code>a.update(b)</code>                      | <code>a  = b</code>     | Set the contents of <code>a</code> to be the union of the elements in <code>a</code> and <code>b</code>        |
| <code>a.intersection(b)</code>                | <code>a &amp; b</code>  | All of the elements in <i>both</i> <code>a</code> and <code>b</code>   |
| <code>a.intersection_update(b)</code>         | <code>a &amp;= b</code> | Set the contents of <code>a</code> to be the intersection of the elements in <code>a</code> and <code>b</code> |
| <code>a.difference(b)</code>                  | <code>a - b</code>      | The elements in <code>a</code> that are not in <code>b</code>  |
| <code>a.difference_update(b)</code>           | <code>a -= b</code>     | Set <code>a</code> to the elements in <code>a</code> that are not in <code>b</code>                            |
| <code>a.symmetric_difference(b)</code>        | <code>a ^ b</code>      | All of the elements in either <code>a</code> or <code>b</code> but <i>not both</i>                             |
| <code>a.symmetric_difference_update(b)</code> | <code>a ^= b</code>     | Set <code>a</code> to contain the elements in either <code>a</code> or <code>b</code> but <i>not both</i>      |
| <code>a.issubset(b)</code>                    | N/A                     | True if the elements of <code>a</code> are all contained in <code>b</code>                                     |
| <code>a.issuperset(b)</code>                  | N/A                     | True if the elements of <code>b</code> are all contained in <code>a</code>                                     |
| <code>a.isdisjoint(b)</code>                  | N/A                     | True if <code>a</code> and <code>b</code> 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

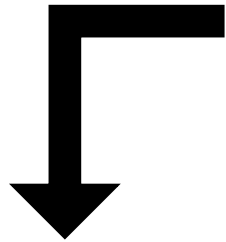
Una de las características más queridas de Python

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

```
[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

# Listas, conjuntos y diccionarios por comprensión

Listas anidadas por comprensión

Supongamos dos listas de nombres

```
all_data = [['John', 'Emily', 'Michael', 'Mary', 'Steven'],  
            ['Maria', 'Juan', 'Javier', 'Natalia', 'Pilar']]
```

Se creará una lista con los nombres que tengan dos e o más

```
nombres_sel = []  
  
for nombres in all_data:  
    a_agregar = [nombre for nombre in nombres if nombre.count('e') >= 2]  
    nombres_sel.extend(a_agregar)
```

Se puede hacer en una sola línea

```
nombres_sel_v2 = [nombre for nombres in all_data for nombre in nombres if nombre.count('e') >= 2]
```

# Reto

John trabaja en una tienda de ropa. Tiene una gran pila de medias que debe combinar por color para la venta. Dado un arreglo de enteros que representan el color de cada media, determine cuántos pares de medias con colores coincidentes hay.

Por ejemplo, hay  $n = 7$  medias de colores  $ar = [1, 2, 1, 2, 1, 3, 2]$ . Hay un par de color 1 y uno de color 2. Quedan tres medias impares, uno de cada color. El número de pares es 2.

**Actividad:** Complete la función `sockMerchant`. Debe devolver un número entero que represente el número de pares de medias coincidentes que están disponibles.

`sockMerchant` tiene los siguientes **parámetros**:

```
def sockMerchant(n, ar):
```

- $n$ : el número de medias en el arreglo  $ar$
- $ar$ : arreglo con los colores de cada media. Cada  $ar[i]$  es un color

## Formato de entrada

Arreglo Python con el listado de números enteros que representa cada color. Ej:

## Restricciones

$$1 \leq n \leq 100$$

$$1 \leq ar[i] \leq 100 \text{ donde } 0 \leq i \leq n$$

## Formato de salida

Devuelve el número total de pares de medias que John puede vender.

<https://www.hackerrank.com/>

```
medias = [10, 20, 20, 10, 10, 30, 50, 10, 20]
```

```
sockMerchant(9, medias)
```

# Reto

Gary es un ávido excursionista. Sigue sus caminatas meticulosamente, prestando mucha atención a pequeños detalles como la topografía. Durante su última caminata, tomó exactamente los pasos. Por cada paso que dio, notó si era un ascenso o un descenso. Las caminatas de Gary comienzan y terminan a nivel del mar y cada paso hacia arriba o hacia abajo representa un cambio de unidad en la altitud. Definimos los siguientes términos:

- Una montaña es una secuencia de pasos consecutivos sobre el nivel del mar, comenzando con un paso hacia arriba desde el nivel del mar y terminando con un paso hacia el nivel del mar.
- Un valle es una secuencia de pasos consecutivos debajo del nivel del mar, comenzando con un paso hacia abajo desde el nivel del mar y terminando con un paso hacia el nivel del mar.

Dada la secuencia de pasos hacia arriba y hacia abajo de Gary durante su última caminata, encuentre e imprima la cantidad de valles por los que caminó.

Por ejemplo, si el camino de Gary es  $S = [DDUUUUDD]$ , primero ingresa a un valle de profundidad 2. Luego se sube y sube a una montaña de 2 unidades de altura. Finalmente, regresa al nivel del mar y termina su caminata.

**Actividad:** Complete la función `countingValleys`. Debe devolver un número entero que denote el número de valles que Gary atravesó. `sockMerchant` tiene los siguientes **parámetros**:

- $n$ : el número de pasos en la caminata de Gary
- $ar$ : arreglo describiendo la travesía

```
def countingValleys(n, ar):
```

## Formato de entrada

Arreglo Python con el listado de pasos de la travesía. Ej:

```
recorrido = 'UDDDUDUU'
```

## Restricciones

$$2 \leq n \leq 10^6$$

$$ar[i] \in \{U, D\}$$

## Formato de salida

Devuelve el número total de pares de medias que John puede vender.

```
countingValleys(8, recorrido)
```

<https://www.hackerrank.com/>

## Resumen

- Tuplas () inmutables. Se consultan con corchete cuadrado
- Listas [] mutables. Se consultan con corchete cuadrado o con list
- Tuplas y listas se concatenan con +
- Diccionarios son arreglos de 2-tuplas. Llave hash
- Conjuntos son diccionarios, únicamente con llaves

## A continuación

Funciones