Programación

Tema 2. Buenas Prácticas e Iterables

Javier González Villa
David Lázaro Urrutia
Departamento de Matemática Aplicada
y Ciencias de la Computación

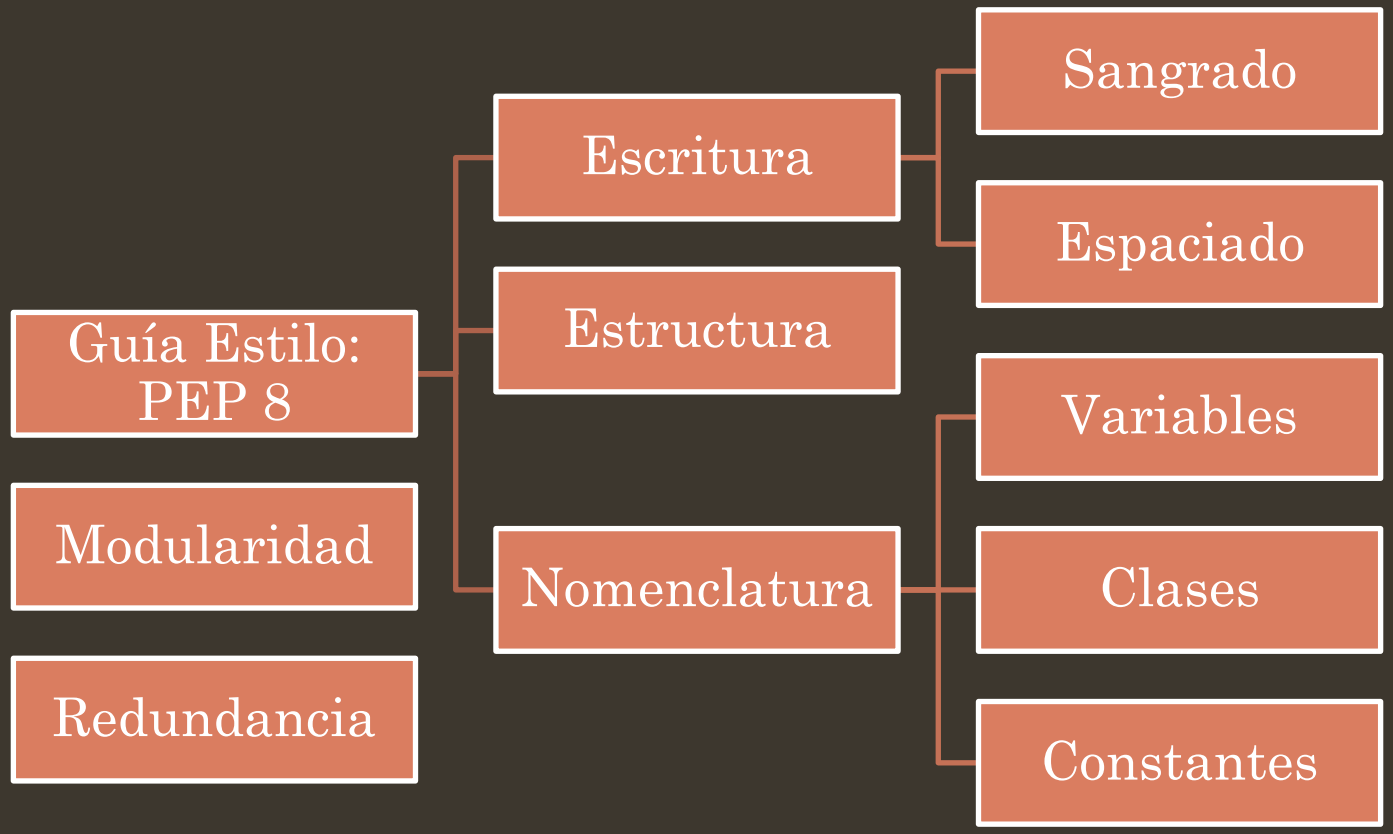
Contenidos

1. Buenas Prácticas
    1. Organización y legibilidad
    2. Comentarios y documentación
    3. Prueba y práctica
2. Iterables
    1. Cadenas
    2. Listas
    3. Tuplas
    4. Diccionarios
    5. Conjuntos

1. Buenas Prácticas

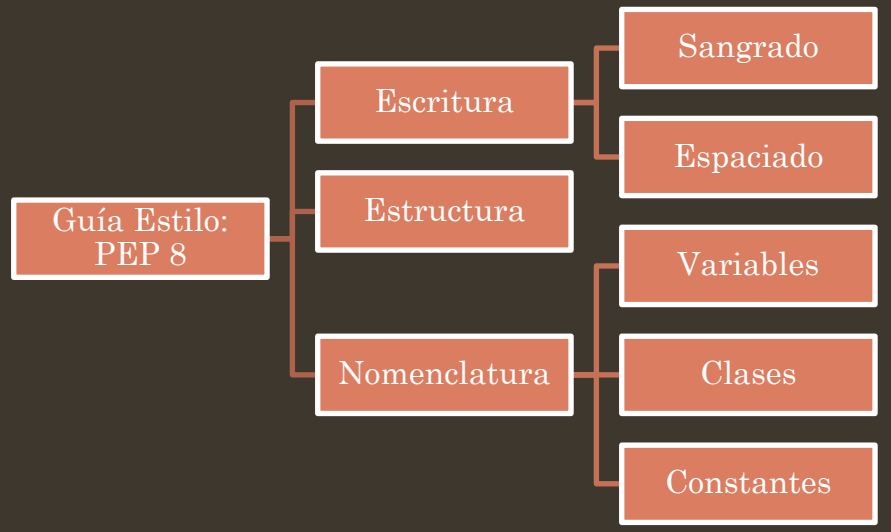
- Son un conjunto de normas, técnicas y principios que los desarrolladores aplican a la hora de escribir código para que este sea legible, mantenible y eficiente.
- Su objetivo es crear software de calidad que sea fácil de leer, comprender y modificar por cualquier persona durante un largo periodo de tiempo.
- Facilita el trabajo colaborativo, reducen la aparición de errores y mejoran la escalabilidad y reutilización del código.
- Ahorran tiempo y recursos a la hora de revisar el código y realizar modificaciones, sobre todo en fases iniciales de aprendizaje y familiarización.

1.1. Organización y legibilidad



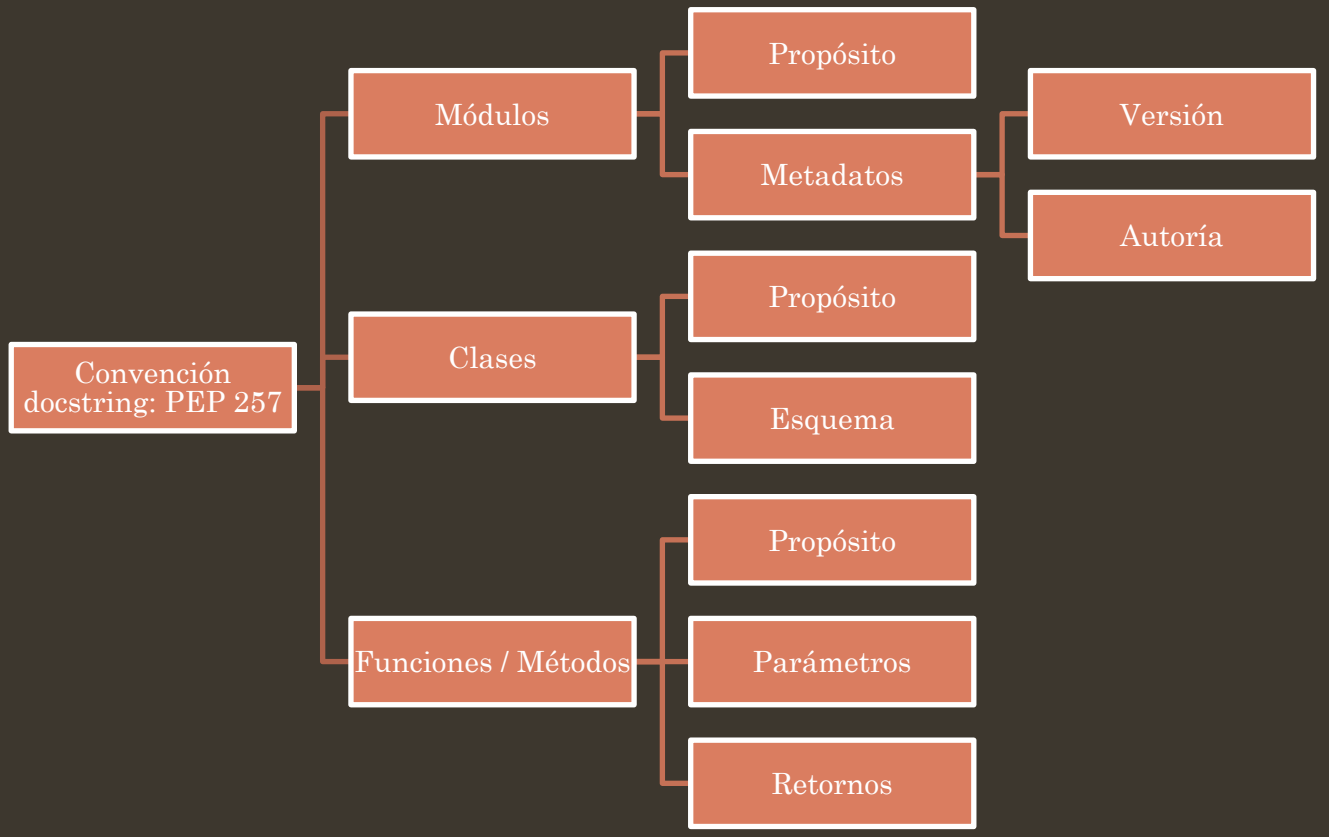
1.1. Organización y legibilidad

- El sangrado es de 4 espacios.
- Longitud máxima de línea de 79 caracteres.
- Usa líneas en blanco para separar bloques lógicos, funciones o métodos.
- La importación de librerías se realiza al comienzo del código.
- Evitar espacios en blanco innecesarios.
- Rodea operadores con espaciado sencillo (x = 2 + 2).
- Variables y funciones en minúsculas utilizando guiones bajos para separar las palabras (*mi_variable*).
- Nombres de clases comienzan con mayúsculas y se ponen de forma continua sin espacios (*MiClase*).
- Las constantes van en mayúsculas con guiones bajos para separar las palabras (*MI_CONSTANTE*).



1.2. Comentarios y documentación

- Agrega comentarios a tu código mediante al carácter # alineados con la línea de código a la que se refieren.
- Evita comentarios obvios o redundantes.
- Mantén los comentarios actualizados cuando realices cambios en el código.



1.3. Prueba y práctica

- La práctica es fundamental para aprender a programar.
- Realiza ejercicios sencillos desde el primer día, aunque no tengan un propósito específico.
- Aprende experimentando y equivocándote.
- Realiza pequeños proyectos personales o docentes relacionados con otras asignaturas.
- Utiliza recursos y comunidades online.
- Ejecutar el código es esencial.
- Edita, ejecuta y observa.

2. Iterables

- Son objetos que contienen secuencias de elementos de diverso tipo y que se pueden recorrer siguiendo unos criterios concretos.
- Puedes pensar que son como contenedores con múltiples compartimentos que pueden ser accedidos de forma ordenada y repetida.
- Principalmente están diseñados para ser utilizados con estructuras de iteración como son los bucles.
- Ten cuidado a la hora de recorrer listas con posibles accesos infinitos y nunca modifiques un iterable al mismo tiempo que lo recorres.

2.1. Cadenas

Cadena: *sucesión de caracteres.*

- Asignación:
  - variable = ‘cadena’
- Se puede operar como si fuera una lista de caracteres individuales o como un conjunto completo.
- Es un objeto **inmutable** ya que no se puede introducir, eliminar o modificar elementos de la cadena tan solo concatenar diferentes cadenas.

https://docs.python.org/es/3.13/library/stdtypes.html#textseq

```
[10]: cadena = 'Hola Mundo!'
      cadena

[10]: 'Hola Mundo!'

[11]: cadena[0]

[11]: 'H'

[12]: cadena[1]

[12]: 'o'

[16]: cadena[2] = 'N'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [16], line 1
----> 1 cadena[2] = 'N'

TypeError: 'str' object does not support item assignment

[17]: cadena+cadena

[17]: 'Hola Mundo!Hola Mundo!'
```

2.1. Cadenas

- Indexación: comienza en 0 y puede ser negativa para extraer valores de la cola.
- Extracción:
  - Índice : cadena[índice].
  - Rangos : cadena[inicio : final (no incluido) : paso].
- Operadores:
  - len(cadena) : retorna la longitud de la cadena.
  - in : comprueba si existe el valor (retorna bool).
  - is : retorna True/False si la cadena es la misma.
  - == : compara el contenido de las cadenas (retorna bool).

```
cadena1 = 'Hola mi nombre es Juan'
cadena2 = 'Adios'
print(cadena[5])
print(cadena[0:10:2])
print(len(cadena))
print('mi' in cadena)
print(cadena1 == cadena2)
```

```
m
Hl in
22
True
False
```

2.1. Cadenas

- Concatenación:
  - Cadena1 + Cadena2 : concatena las dos cadenas.
- Replicación:
  - Cadena * n : repite la cadena n veces.
- Conversión de tipos:
  - str(valor) : convierte el valor en una cadena.
  - float(cadena) : convierte una cadena en un real (si es posible).
- Búsqueda:
  - cadena.find(cadena) : retorna el índice de la primera aparición de la cadena buscada.

```
[28]: cadena1 = 'Hola '
      cadena2 = 'Mundo!'
      print(cadena1 + cadena2)
      print(cadena1*2 + cadena2*3)
      print(type(3.0))
      print(type(str(3.0)))
      print(cadena1.find('la'))

      Hola Mundo!
      Hola Hola Mundo!Mundo!Mundo!
      <class 'float'>
      <class 'str'>
      2
```

2.2. Listas

Lista: *secuencia* ***mutable*** *indexada de objetos.*

- Asignación:
  - *Lista = [obj1, obj2, obj3, …,objN]*
- Puede estar compuesta por objetos de diferente clase y estos pueden ser modificados.
- Es un objeto **mutable** pudiendo introducir, eliminar o modificar elementos de la lista.

https://docs.python.org/es/3.13/tutorial/datastructures.html#more-on-lists

```
[19]: lista=[1,2.0,'Tres',4+5j,True]
      lista

[19]: [1, 2.0, 'Tres', (4+5j), True]

[20]: lista[0] = 'Nuevo Valor'
      lista

[20]: ['Nuevo Valor', 2.0, 'Tres', (4+5j), True]

[22]: lista.append('Valor Adicional')
      lista

[22]: ['Nuevo Valor', 2.0, 'Tres', (4+5j), True, 'Valor Adicional']

[24]: lista.remove(2)
      lista

[24]: ['Nuevo Valor', 'Tres', (4+5j), 'Valor Adicional']
```

2.2. Listas

- Indexación: comienza en 0 y puede ser negativa para extraer valores de la cola.
- Extracción:
  - Índice : lista[índice].
  - Rangos : lista[inicio : final (no incluido) : paso].
- Modificación:
  - lista[índice] = valor
- Operadores:
  - len(lista) : retorna el número de elementos de la lista.
  - in : comprueba si existe el valor (retorna bool).
  - is : retorna True/False si la lista es la misma.
  - == : compara el contenido de la lista (retorna bool).

```
[34]: lista = [1,2,3,4,5,6,7,8,9,10]
      print(len(lista))
      print(lista[0])
      print(lista[-1])
      print(lista[2:6:2])
      print(3 in lista)
      lista.append(11)
      print(lista)
      lista.remove(4)
      print(lista)
      lista2 = lista
      print(lista2 is lista)
      lista2[1] = 6
      print(lista == lista2)
      print(lista)
      print(lista2)

      10
      1
      10
      [3, 5]
      True
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
      [1, 2, 3, 5, 6, 7, 8, 9, 10, 11]
      True
      True
      [1, 6, 3, 5, 6, 7, 8, 9, 10, 11]
      [1, 6, 3, 5, 6, 7, 8, 9, 10, 11]
```

2.2. Listas

- Concatenación:
  - Lista1 + Lista2 : concatena las dos listas.
  - Lista1.extend(Lista2) : guarda los valores de Lista2 en Lista1.
  - Lista.append(valor) : añade el valor al final de la lista.
- Replicación:
  - Lista * n : repite la lista n veces.
- Conversión de tipos:
  - str(lista) : convierte la lista en una cadena.
- Transformación:
  - Lista.reverse() : retorna la lista con el orden de los valores invertido.
  - Lista.remove(valor) : elimina el valor indicado de la lista.
  - Lista.sort() : ordena los valores de la lista de menor a mayor.

```
[47]: Lista1 = [1,2,3,4,5]
      Lista2 = [6,7,8,9,10]
      print(Lista1+Lista2)
      print(Lista1)
      Lista1.extend(Lista2)
      print(Lista1)
      Lista1.append(0)
      print(Lista1)
      Lista1.reverse()
      print(Lista1)
      Lista1.remove(5)
      print(Lista1)

      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
      [1, 2, 3, 4, 5]
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0]
      [0, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
      [0, 10, 9, 8, 7, 6, 4, 3, 2, 1]
```

2.2. Listas

Anidación: *combinación de elementos iterables formando estructuras multidimensionales.*

- Asignación:
  - *lista =[obj1, obj2, obj3, …,objN]*
  - *matriz = [lista1, lista2, lista3, …,listaN]*
- Indexación:
  - *lista[índice]*
  - *matriz[índice][índice]*

```
[24]: lista_1 = [1,2,3]
      lista_2 = [4,5,6]
      lista_3 = [7,8,9]
      matriz = [lista_1, lista_2, lista_3]
      matriz

[24]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

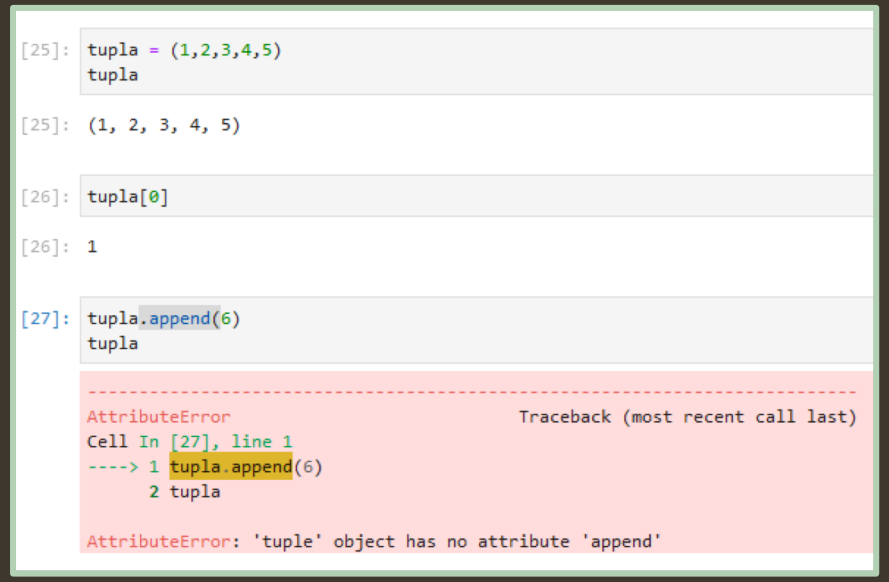
```
[27]: matriz = [[1,2,3], [4,5,6], [7,8,9]]
      matriz[1][2]

[27]: 6
```

2.3. Tuplas

Tupla: *secuencia* ***inmutable*** *indexada de objetos.*

- Asignación:
  - *Tupla = (obj1, obj2, obj3, …,objN)*
- Puede estar compuesta por objetos de diferente clase.
- Es un objeto **inmutable** por lo que no se puede introducir, eliminar o modificar elementos de la lista.

https://docs.python.org/es/3.13/tutorial/datastructures.html#tuples-and-sequences



2.3. Tuplas

- Indexación: comienza en 0 y puede ser negativa para extraer valores de la cola.
- Extracción:
  - Índice : tupla[índice].
  - Rangos : tupla[inicio : final (no incluido) : paso].
- Operadores:
  - len(tupla) : retorna el número de elementos de la tupla.
  - in : comprueba si existe el valor (retorna bool).
  - is : retorna True/False si la tupla es la misma.
  - == : compara el contenido de las tuplas (retorna bool).

```
[54]: tupla1 = ('Hola', [1,2,3], 100)
      tupla2 = ([1,2,3], 'Hola', 100)
      print(tupla[1])
      print(len(tupla))
      print('Hola' in tupla)
      print(tupla1 == tupla2)

      [1, 2, 3]
      3
      True
      False
```

2.3. Tuplas

- Concatenación:
  - Tupla1 + Tupla2 : concatena las dos tuplas.
- Replicación:
  - Tupla * n : repite la tupla n veces.
- Conversión de tipos:
  - str(tupla) : convierte la tupla en una cadena.

```
[67]: tupla1 = ('a',65,'hola',[1,2,3])
      tupla2 = (4,5)
      print(tupla1 + tupla2)
      print(tupla2*3)
      print(type(str(tupla1)))
      print(str(tupla1))

      ('a', 65, 'hola', [1, 2, 3], 4, 5)
      (4, 5, 4, 5, 4, 5)
      <class 'str'>
      ('a', 65, 'hola', [1, 2, 3])
```

2.4. Diccionarios

Diccionario: *conjunto* ***mutable*** *de pares clave-valor.*

- Asignación:
  - *Diccionario = {clave1 : valor1, clave2 : valor2, …}*
- Tanto la clave como el valor puede estar compuesta por objetos de diferente clase y estos pueden ser modificados.
- Es un objeto **mutable** por lo que se puede introducir, eliminar o modificar las claves y los valores del diccionario.

https://docs.python.org/es/3.13/tutorial/datastructures.html#dictionaries

```
[71]: diccionario = {'Pedro':8, 'María':7, 'Luis':5}
      diccionario

[71]: {'Pedro': 8, 'María': 7, 'Luis': 5}

[72]: diccionario['Pedro']

[72]: 8

[73]: diccionario['Pedro'] = 4
      diccionario

[73]: {'Pedro': 4, 'María': 7, 'Luis': 5}
```

2.4. Diccionarios

- Indexación: se indexa en función de las claves en vez de valores de índices numéricos.
- Extracción:
  - Clave : diccionario[clave].
  - Claves actuales : diccionario.keys().
  - Valores actuales: diccionario.values().
- Inserción y modificación:
  - diccionario[clave_nueva] = valor_nuevo
  - diccionario[clave_existente] = valor_nuevo
- Operadores:
  - len(diccionario) : retorna el número de pares.
  - in : comprueba si existe la clave (retorna bool).
  - is : retorna True/False si el diccionario es el mismo.
  - == : compara el contenido de los diccionarios (retorna bool).

```
[91]: diccionario = {'Pedro':8, 'María':7, 'Luis':5}
      diccionario['Luis']

[91]: 5

[92]: diccionario.keys()

[92]: dict_keys(['Pedro', 'María', 'Luis'])

[93]: diccionario.values()

[93]: dict_values([8, 7, 5])

[94]: len(diccionario)

[94]: 3
```

2.5. Conjuntos

Conjunto: *conjunto de valores* ***desordenados y no indexados.***

- Asignación:
  - *conjunto = {valor1, valor2, varlo3, …}*
  - *conjunto_vacio = set()*
- Los valores son almacenados de forma desordenada y no indexada.
- Su característica fundamental es que no contienen elementos repetidos.

https://docs.python.org/es/3.13/tutorial/datastructures.html#sets

```
[1]: conjunto1 = {1,3,5}
     conjunto2 = {2,4,6}
     conjunto1[0]

---------------------------------------------------------------------------
TypeError                                 Traceba
Cell In[1], line 3
      1 conjunto1 = {1,3,5}
      2 conjunto2 = {2,4,6}
----> 3 conjunto1[0]

TypeError: 'set' object is not subscriptable
```

2.5. Conjuntos

- Unión:
  - Conjunto1 | Conjunto2 : todos los elementos no repetidos.
- Intersección:
  - Conjunto1 & Conjunto2: elementos en ambos conjuntos.
- Diferencia:
  - Conjunto1 – Conjunto2: elementos del primer conjunto que no están en el segundo.

```
[3]: conjunto1 = {1,3,5}
     conjunto2 = {2,4,6}
     conjunto1 | conjunto2

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

```
[4]: conjunto1 = {1,3,5}
     conjunto2 = {2,4,6}
     conjunto1 & conjunto2

[4]: set()
```

```
[6]: conjunto1 = {1,3,5}
     conjunto2 = {3,5,6}
     conjunto1 - conjunto2

[6]: {1}
```

2.5. Conjuntos

- Operadores:
  - len(diccionario) : retorna el número de pares.
  - in : comprueba si existe la clave (retorna bool).
  - is : retorna True/False si el diccionario es el mismo.
  - == : compara el contenido de los diccionarios (retorna bool).
- Inserción y modificación:
  - conjunto.add(valor)
  - conjunto.discard(valor)

```
[10]: conjunto1 = {1,3,5}
      conjunto2 = {1,3,5}
      len(conjunto1)

[10]: 3

[11]: 3 in conjunto1

[11]: True

[12]: conjunto1 is conjunto2

[12]: False

[13]: conjunto1 == conjunto2

[13]: True
```