Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

Supongamos

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,’c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,’c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t’)}
```

- Método *clear()*:

  Permite borrar todos los elementos de una lista o de un conjunto. Las tuplas son inmutables:

  `tupla.clear()` → *Error*

  `lista.clear()` → *[ ]*

  `conjunto.clear()` → *set()*

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Metodo *sort()*:
  - *Modifica el orden de los elementos de una lista. Si la lista contiene, a su vez, otros contenedores se ordenan por el primero de los elementos de estos contendores.*

    `tupla.sort()` → *Error (las tuplas son inmutables: No se pueden modificar)*

    `lista.sort()` → *lista=*`[(1,'t'),(2,'a'),(4,'c'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]`

    `conjunto.sort()` → *Error (a los conjuntos no se les puede inducir un orden)*

  - *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

    `lista.sort(reverse=True)`→`[(9,'b'),(6, 'w'),(5,'z'),(4,'c'),(4,'c'),(2,'a'),(1,'t')]`

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Si se quiere ordenar *por otro de los elementos* de los contenedores internos se especifica el parámetro *key=lambda e:e[i]* donde *i* es la posición de los elementos por los que ordenar (se recuerda que el primer elemento es el 0).

```
lista.sort(key=lambda e:e[1]) → lista=[(2,'a'),(9,'b'),(4,'c'),(4,'c'),(1,'t'),
                                       (6,'w'),(5,'z')]
```

- *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

```
lista.sort(key=lambda e:e[1], reverse=True) → lista=[(5,'z'),(6, 'w'),(1,'t'),
                                                (4,'c'),(4,'c'),(9,'b'),(2,'a')]
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Función *sorted()*:
  - *Devuelve una lista con los elementos del contenedor ordenados. Si el contenedor contiene, a su vez, otros contenedores se ordenan por el primero de los elementos de estos últimos*

```
sorted(tupla)→[(1,'t'),(2,'a'),(4,'c'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]
sorted(lista)→[(1,'t'),(2,'a'),(4,'c'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]
sorted(conjunto) →[(1,'t'),(2,'a'),(4,'c'),(5,'z'),(6,'w'),(9,'b')]
```

  - *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

```
sorted(tupla, reverse=True)→[(9,'b'),(6,'w'),(5,'z'),(4,'c'),(4,'c'),(2,'a'),
                             (1,'t')]
sorted(lista, reverse=True)→[(9,'b'),(6,'w'),(5,'z'),(4,'c'),(4,'c'),(2,'a'),
                             (1,'t')]
sorted(conjunto, reverse=True)→[(9,'b'),(6,'w'),(5,'z'),(4,'c'),(2,'a'),(1,'t')]
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')}
```

- Si se quiere ordenar *por otro de los elementos* de los contenedores internos se especifica el parámetro *key=lambda e:e[i]* donde *i* es la posición de los elementos por los que ordenar.

```
sorted(tupla,key=lambda e:e[1]) → [(2,'a'),(9,'b'),(4,'c'),(4,'c'),(1,'t'),
                                   (6,'w'),(5,'z')]
sorted(lista,key=lambda e:e[1]) → [(2,'a'),(9,'b'),(4,'c'),(4,'c'),(1,'t'),
                                   (6,'w'),(5,'z')]
sorted(conjunto,key=lambda e:e[1]) → [(2,'a'),(9,'b'),(4,'c'),(1,'t'),(6,'w'),
                                      (5,'z')]
```

- *Si se quiere ordenar en orden inverso se añade el parámetro reverse=True (por defecto es False)*

```
sorted(tupla,key=lambda e:e[1],reverse=True) → [(5,'z'),(6,'w'),(1,'t'),(4,'c'),
                                                (4,'c'),(9,'b'),(2,'a')]
sorted(lista,key=lambda e:e[1],reverse=True) → [(5,'z'),(6,'w'),(1,'t'),(4,'c'),
                                                (4,'c'),(9,'b'),(2,'a')]
sorted(conjunto,key=lambda e:e[1],reverse=True) → [(5,'z'),(6,'w'),(1,'t'),
                                                   (4,'c'),(9,'b'),(2,'a')]
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=(2,4,4,7.2)
lista=[2,4,4,7.2]
conjunto={2,4,4,7.2}
```

- *Función sum():*

  *Devuelve la suma de los elementos de un contenedor. ¡**OJO**! El contenedor debe ser de elementos numéricos:*

  ```
  sum(tupla) →17.2
  sum(lista) → 17.2
  sum(conjunto) → 13.2
  ```

- *Función len():*

  Devuelve el número de elementos de un contenedor.

  ```
  len(tupla) →4
  len(lista) → 4
  len(conjunto) → 3
  ```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

```
tupla=((2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t'))
lista=[(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t')]
conjunto={(2,'a'),(5,'z'),(4,'c'),(9,'b'),(4,'c'),(6,'w'),(1,'t’)}
```

- Función *max() / min()*:
  - *Devuelve el máximo/mínimo de los elementos de un contenedor. Si el contenedor contiene, a su vez otros contenedores, se devuelve el del mayor/menor según el primero de los elementos de estos contendores*

```
max(tupla)→(9,’b’)    / min(tupla)→(1,'t')
max(lista)→(9,’b’)    / min(tupla)→(1,'t’)
max(conjunto) →(9,’b’)/ min(tupla)→(1,'t')
```

  - Si se quiere *el máximo/mínimo por otro de los elementos* de los contenedores internos se especifica el parámetro *key=lambda e:e[i]* donde ***i*** es la posición del elemento a buscar.

```
max(tupla,key=lambda e:e[1])→(5,’z’)/min(tupla,key=lambda e:e[1])→ (2,'a')
max(lista,key=lambda e:e[1])→(5,'z’)/min(lista,key=lambda e:e[1])→ (2,'a')
max(conjunto,key=lambda e:e[1])→(5,'z’)/min(conjunto,key=lambda e:e[1])→
                                                              (2,'a')
```

Métodos y funciones sobre contenedores (Tuplas/Listas/Conjuntos):

Si los contenedores internos tuviesen nombre (*namedtuple*) se pueden referenciar en la expresión *key=lambda* el elemento por su nombre.

Por ejemplo: Si las tuplas hubiesen sido creadas con:

Par=*namedtuple*('pareja', '*número*, *letra*')

```
tupla=(Par(2,'a'),Par(5,'z'),Par(4,'c'),Par(9,'b'),Par(4,'c'),Par(6,'w'),Par(1,'t'))
lista=[Par(2,'a'),Par(5,'z'),Par(4,'c'),Par(9,'b'),Par(4,'c'),Par(6,'w'),Par(1,'t')]
conjunto={Par(2,'a'),Par(5,'z'),Par(4,'c'),Par(9,'b'),Par(6,'w'),Par(1,'t')}

  sorted(tupla,key=lambda e:e.letra)→[(2,'a'),(9,'b'),(4,'c'),(4,'c'),
                                      (1,'t'),(6,'w'),(5,'z')]
  max(lista,key=lambda e:e.letra)→ (5,'z')
  min(conjunto,key=lambda e:e.letra)→ (2,'a')
```

Operadores para el manejo de conjuntos

```
conjunto1={2,4,4,7.2}
conjunto2={8.2,4,2,'a','2'}
```

Operador unión (|) *(equivalente a 'or')*: Los elementos de ambos sin repetir

```
conjunto1|conjunto2 → {2, 4, 7.2, 8.2, 'a', '2'}
```

Operador intersección (&) *(equivalente a 'and')*: Los elementos comunes sin repetir

```
conjunto1&conjunto2 → {2,4}
```

Operador diferencia (-) Los elementos del primero que no están en el segundo

```
conjunto1-conjunto2 → {7.2}
conjunto2-conjunto1 → {8.2,'2','a'}
```

Operador diferencia simétrica (^) Los elementos que solo están en uno de los conjuntos=U-∩

```
conjunto1^conjunto2 → {'a',7.2,8.2,'2'}
conjunto2^conjunto1 → {'a',7.2,8.2,'2'}
```

Operadores para el manejo de conjuntos

```
conjunto1={2,4,4,7.2}
conjunto2={8.2,4,2,'a','2',7.2}
conjunto3={8.2,4,2,'a','2',7.2}
```

Operador subconjunto (< o <=)

```
conjunto1<conjunto2 → True
conjunto2<conjunto1 → False
conjunto2<conjunto3 → False
conjunto2<=conjunto3 → True
```

Sentencia break, Esquemas de “todos cumplen” y “existe alguno” (I):

La sentencia *break* interrumpe (termina) un bucle for o while.

Esquema que permite ver *si todos* los elementos de un contenedor cumplen una *condición*:

```
def todos_cumplen_que...(contenedor):
    res=True
    for elemento in contenedor:
        if not condición:
            res=False
            break
    return res
```

Presuponemos que todos cumplen

Sentencia que rompe la ejecución de un bucle

Se supone que todos cumplen la condición (res=*True*). Se van recorriendo todos los elementos y, si alguno no cumple la condición, se cambia res (res=*False*) y no se sigue preguntando (break), lo que hace más eficiente el algoritmo.

Sentencia break, esquemas de “todos cumplen” y “existe alguno” (II):

Esquema que permite ver si *algún* elemento de un contenedor cumple una *condición*:

```
def hay_alguno_que…(contenedor):
    res=False
    for elemento in contenedor:
        if condición:
            res=True
            break
    return res
```

Se presupone que ninguno cumple

Sentencia que rompe la ejecución de un bucle

Se supone que ninguno cumple la condición (res=*False*). Se van recorriendo todos los elementos y si alguno cumple la condición se cambia res (res=*True*) y no se sigue preguntando (break) lo que hace más eficiente el algoritmo

Ejercicio:

Modifique el proyecto *T09_Datos_Personales*

- En el módulo *datos_personales.py*:
  - Añada una función: *todos_entran_entre_años* que, recibiendo una lista de tupla de tipo *Persona4* y un año1 (de tipo *int*) y un año2 (de tipo *int*), devuelva *True* si todas las personas entraron entre los dos años dados (ambos incluidos), en otro caso devuelva *False*.
  - Añada una función: *alguien_ha_madrugado* que, recibiendo una lista de tupla de tipo Persona4 y una hora (de tipo *int*), devuelva *True* si alguna de las personas entró antes de dicha hora, en otro caso devuelva *False*.

- En el módulo *test_datos_personales.py*
  - Añada un test para cada una de las dos funciones