Operadores sobre contenedores:

- Operador *in*: *Devuelve True o False según el contenedor contenga, o no, al elemento*

Sintaxis:

elemento *in contenedor:*

Ejemplos

- *Si, mi_lista=[('a',62),('b',56),('c',90),('d',-22),('e',20)]*
  ('c',90) in mi_lista → True
  ('d',21) in mi_lista → False
- *Si, mi_tupla=('a',62,'b',56,'c',90)*
  56 in mi_tupla → True
  'A' in mi_tupla → False
- *Si, mi_conjunto={"gallo",62,"perro","foca",'c',2.89}*
  "foca" in mi_conjunto → True
  2 in mi_conjunto → False

Operadores sobre contenedores:

Operador **+**: *Aplicado a dos contenedores los concatena en uno solo (salvo para conjuntos que no está permitido el operador).*

¡OJO! tiene que ser de mismo tipo (lista con lista, tupla con tupla o string con string)

Ejemplo:

- *Si, mi_tupla=('a',62,'b',56,'c',90)*
  mi_tupla **+** *(20,'h','p',"perro","foca",)* → *('a', 62, 'b', 56, 'c', 90, 20, 'h', 'p',"perro","foca")*

Algoritmia: Filtrar un contenedor

Denominamos normalmente *filtro* al proceso por el que se selecciona determinados elementos de un contenedor. Generalmente lista o conjuntos.

El proceso es sencillo y consiste en:

1. *Definir una función* que reciba en los parámetros formales:
   - El contenedor
   - El/los parámetro/s en los que recibir los valores por los que filtrar
2. *Crear una variable* (*res*), generalmente un contenedor, en el que se almacenarán los elementos que cumplen la condición.
3. *Ejecutar* una sentencia *for* sobre el contenedor, que lo recorrerá elemento a elemento.
4. *Dentro del bloque de sentencias del for ejecutar* una sentencia *if* en la que se evalúe la condición para filtrar los elemento que la cumplen.
5. Dentro del bloque de sentencias del if, *añadir* a la variable (res) del punto 2 el elemento.
6. *Devolver* la variable (res) del punto 2 *return res* (que es la que ha ido almacenando los elementos filtrados).

Algoritmia: Filtrar un contenedor

Ejemplo

Se supone que se dispone de una lista de tuplas en la que cada tupla contiene datos de la población de ciertos países: *Población*=*namedtuple*('población','cód_país, año, num_habitantes')

Se trata de obtener una lista con las tuplas que tienen determinado código de país

(cada línea coincide con el epígrafe de la diapositiva anterior):

```
1) def filtra_país(poblaciones:list[Población],código:str)
                                                ->list[Población]:
2)     res=list()
3)     for p in poblaciones:
4)         if p.cód_país==código:
5)             res.append(p)
6)     return res
```

Algoritmia: Filtrar un contenedor

Ejemplo

Se supone que se dispone de una lista de tuplas en la que cada tupla contiene datos de la población de ciertos países: Población=*namedtuple*('población','cód_país, año, num_habitantes')

Se trata de obtener una lista con las tuplas que tienen determinado código de país *y con cierto número de habitantes o más*.

(la condición exige un operador lógico ya que deben darse dos condiciones):

```
def filtra_país_y_habitantes(poblaciones:list[Población],código:str,
                                        habitantes:int)->list[Población]:
    res=list()
    for p in poblaciones:
        if p.cód_país==código and p.num_habitantes>=habitantes:
            res.append(p)
    return res
```

Algoritmia: Obtener una vista

Denominamos normalmente obtener una vista al proceso por el que se selecciona determinados campos de los elementos de un contenedor. Generalmente lista o conjuntos.

El proceso es sencillo y consiste en:

1. *Definir una función* que reciba en los parámetros formales:
   - El contenedor
2. *Crear una variable* (*res*), generalmente un contenedor, en el que se almacenarán los campos que hay que devolver en la vista. *Si hay más de un campo se agrupan en una tupla*
3. *Ejecutar* una sentencia *for* sobre el contenedor, que lo recorrerá elemento a elemento.
4. En el bloque de sentencias del for, *añadir* a la variable (*res*) del punto 2 el elemento que corresponda. Si son más de uno insertar una tupla que los agrupe. Ojo! Si se agrupan en tupla, habrá dos paréntesis abiertos y dos cerrados.
5. *Devolver* la variable (*res*) del punto 2 *return res* (que es la que ha ido almacenando el/los campo/s que conforman la vista.

Algoritmia: Filtrar un contenedor

Ejemplo

Se supone que se dispone de una lista de tuplas en la que cada tupla contiene datos de la población de ciertos países: *Población*=*namedtuple*('población','cód_país, año, num_habitantes')
Se trata de obtener una lista de tuplas con el código y el año.

(cada línea coincide con el epígrafe de la diapositiva anterior):

```
1) def obtiene_código_y_año(poblaciones:list[Población])->list:
2)     res=list()
3)     for p in poblaciones:
4)         res.append((p.cód_país, p.año))
5)     return res
```

¡ojo! Esta lista ya no es de tipo *Población* porque está compuesta por tuplas que sólo tienen dos elementos, par al que no se ha definido un namedtuple

Parámetros de funciones

- *Hemos aprendido que las funciones utilizan los parámetros para darles versatilidad y poderlas usar con distintos valores.*
  - *Parámetros formales: son los que se escriben entre paréntesis en la cabecera de definición de una función.*

    def nombre_función (nombre_parámetro 1:tipo, nombre_parámetro 2_tipo, …)->tipo:

    Por ejemplo:

    def filtra_país(poblaciones:list[Población], código:str)->list[Población]:

  - *Parámetros reales: son los que se escriben entre paréntesis cuando se invoca a la función (pueden ser variables o directamente literales)*

    nombre_función (variable1/literal 1, variable 2/literal 2, …)

    Por ejemplo:

    filtra_país(lista_pob, 'ESP')

Parámetros por defectos:

- *Podemos hacer que los parámetros formales tomen un valor por defecto, de forma que, si se omite su correspondiente parámetro real, tome dicho valor. Estos últimos (los que se pueden omitir) les podemos denominar parámetros opcionales y los demás parámetros obligatorios.*

  Los parámetros opcionales que pueden tomar valor por defecto deben estar escritos en las últimas posiciones. Un parámetro se convierte en opcional añadiendo detrás de su nombre y tipo el signo "=" junto con el valor por defecto.

  def nombre_función (obligatorio1:tipo, obligatorio2:tipo,…opcional1:tipo=valor1,opcional2:tipo =valor2,..)->tipo:

Por ejemplo.

- *en la definición:*

```
def filtra_país_y_habitantes(poblaciones:list[Población],
        código:str, habitantes:int=50000000)->list[Población]:
```

- *en la invocación:*

```
filtra_país_y_habitantes(lista_pob,'ESP',75000000)
filtra_país_y_habitantes(lista_pob,'ESP')
```

(En este último caso el parámetro formal `habitantes` *tomará el valor 5000000)*

Invocación de parámetros por nombre:

Sabemos que cuando se invoca a una función los parámetros reales se escriben en el mismo orden en los esperan los parámetros formales. No obstante, se puede alterar el orden si se antepone a los parámetros reales el nombre del parámetro formal seguido del signo "="

Por ejemplo.

```
filtra_país_y_habitantes(código='ESP',  habitantes= 75000000, poblaciones= lista_pob)
```

Ejercicio: Se trata de modificar el proyecto "*T08_Datos_Personales*" para:

- Añadir una función nueva al archivo/módulo *datos_personales.py* "*calculasumaedades*" que recibiendo como parámetros una lista de tuplas con los datos de personas, devuelva la suma de las edades de las personas de la lista.
- Añadir una función nueva al archivo/módulo *datos_personales.py* "*calculapromedioedades*" que recibiendo como parámetros una lista de tuplas con los datos de personas y una provincia, devuelva el promedio de las edades de las personas de la lista. Importante, si no puede calcular el promedio devuelve *None*.
  Esta función tiene como parámetro formal por defecto "Sevilla"
- Modificar el archivo/módulo *test_datos_personales.py* para probar las nuevas funciones. En el caso de *calculapromedioedades*, haga dos pruebas pasando la lista y una provincia y otra pasando sólo la lista