


12 Iterables, Mapeo, Filtros y Reducciones.

Objetivo. ...

Funciones de Python: ...

[MACTI-Algebra_Lineal_01](#) by [Luis M. de la Cruz](#) is licensed under [Attribution-ShareAlike 4.0 International](#) 

13 Iterables

- En Python existen objetos que contienen secuencias de otros objetos (listas, tuplas, diccionarios, etc).
- Estos objetos se pueden recorrer usando ciclos **for ... in ...**.
- A estos objetos se les conoce también como **iterables** (objetos iterables, secuencias iterables, contenedores iterables, conjunto iterable, entre otros).

13.1 Ejemplo 1.

Crear una cadena, una lista, una tupla, un diccionario, un conjunto y leer un archivo; posteriormente recorrer cada uno de estos iterables usando un ciclo **for**:

```
mi_cadena = "pythonico"
mi_lista = ['p','y','t','h','o','n','i','c','o']
mi_tupla = ('p','y','t','h','o','n','i','c','o')
mi_dict = {'p':1,'y':2,'t':3,'h':4,'o':5,'n':6,'i':7,'c':8,'o':9}
mi_conj = {'p','y','t','h','o','n','i','c','o'}
mi_archivo = open("mi_archivo.txt")

print('\nCadena:', end=' ')
# Recorremos la cadena e imprimimos cada elemento
for char in mi_cadena:
    print(char, end=' ')

print('\nLista:', end=' ')
# Recorremos la lista e imprimimos cada elemento
for element in mi_lista:
    print(element, end=' ')

print("\nTupla: ", end='')
# Recorremos la tupla e imprimimos cada elemento
for element in mi_tupla:
    print(element, end=' ')
```

```

print("\nDiccionario (claves): ", end='')
# Recorremos el diccionario e imprimimos cada clave
for key in mi_dict.keys():
    print(key, end=' ')

print("\nDiccionario (valores): ", end='')
# Recorremos el diccionario e imprimimos cada valor
for key in mi_dict.values():
    print(key, end=' ')

print("\nConjunto: ", end='')
# Recorremos el conjunt e imprimimos cada elemento
for s in mi_conj:
    print(s, end = ' ')

print("\nArchivo: ")
# Recorremos el archivo e imprimimos cada elemento
for line in mi_archivo:
    print(line, end = '')

```

Cadena: p y t h o n i c o
 Lista: p y t h o n i c o
 Tupla: p y t h o n i c o
 Diccionario (claves): p y t h o n i c
 Diccionario (valores): 1 2 3 4 9 6 7 8
 Conjunto: y h t c i n o p
 Archivo:
 p
 y
 t
 h
 o
 n
 i
 c
 o

Observa el caso del diccionario y del conjunto: * Diccionario: cuando hay claves repetidas, se sustituye el último valor que toma la clave ('0':9). * Conjunto: los elementos se ordenan, y no se admiten elementos repetidos.

14 Mapeo.

En análisis matemático, un *Mapeo* es una regla que asigna a cada elemento de un primer conjunto, un único elemento de un segundo conjunto:

map

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-1} \end{bmatrix} \longrightarrow \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_{n-1} \end{bmatrix}$$

14.1 map

En Python existe la función `map(function, sequence)` cuyo primer parámetro es una función la cual se va a aplicar a una secuencia, la cual es el segundo parámetro. El resultado será una nueva secuencia con los elementos obtenidos de aplicar la función a cada elemento de la secuencia de entrada.

14.2 Ejemplo 2.

Crear el siguiente mapeo con una lista, una tupla, un conjunto

$$f(x) = x^2$$

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 \\ 1 \\ 4 \\ 9 \\ 16 \end{bmatrix}$$

```
# Primero definimos la función
def square(x):
    """
    Calcula el cuadrado de x.
    """
    return x**2

# Luego definimos las secuencias
l = [0,1,2,3,4]
t = (0,1,2,3,4)
s = {0,1,2,3,4}

# Ahora creamos los mapeos
lmap = map(square, l)
tmap = map(square, t)
smap = map(square, s)

# Checamos el tipo de cada mapeo
print(type(lmap), type(tmap), type(smap))
```

```
print('Lista {}'.format(l))
print('Mapeo {}'.format(list(lmap)))

print('Tupla {}'.format(t))
print('Mapeo {}'.format(tuple(tmap)))

print('Conj {}'.format(s))
print('Mapeo {}'.format(set(smap)))
```

```
<class 'map'> <class 'map'> <class 'map'>
```

```
Lista [0, 1, 2, 3, 4]
```

```
Mapeo [0, 1, 4, 9, 16]
```

```
Tupla (0, 1, 2, 3, 4)
```

```
Mapeo (0, 1, 4, 9, 16)
```

```
Conj {0, 1, 2, 3, 4}
```

```
Mapeo {0, 1, 4, 9, 16}
```

Observa que el resultado del mapeo es un objeto de tipo `<class 'map'>` por lo que debemos convertirlo en un tipo que pueda ser desplegado para imprimir.

14.3 Ejemplo 3.

Crear un mapeo para convertir grados Fahrenheit a Celsius y viceversa:

```
def toFahrenheit(T):
    """
    Transforma los elementos de T en grados Farenheit.
    """
    return (9/5)*T + 32

def toCelsius(T):
    """
    Transforma los elementos de T en grados Celsius.
    """
    return (5/9)*(T-32)
```

Celsius → Fahrenheit

```
# Lista original con los datos
c = [0, 22.5, 40, 100]

# Construimos el mapeo y lo nombramos en `fmap`.
fmap = map(toFahrenheit, c)
```

```
# Imprimos a lista original y el mapeo
print(c)
print(list(fmap))
```

```
[0, 22.5, 40, 100]
[32.0, 72.5, 104.0, 212.0]
```

NOTA. Solo se puede usar el mapeo una vez, si vuelves a ejecutar la celda anterior el resultado del mapeo estará vacío. Para volverlo a generar debes ejecutar la celda donde se construye el mapeo.

Lo anterior se puede realiza en una sola línea: crear el mapeo, convertir a lista e imprimir

```
print(list(map(toFahrenheit,c)))
```

```
[32.0, 72.5, 104.0, 212.0]
```

Fahrenheit → Celsius

```
# Lista original con los datos
f = [32.0, 72.5, 104.0, 212.0]

# Conversión en una sola línea
print(list(map(toCelsius, f)))
```

```
[0.0, 22.5, 40.0, 100.0]
```

14.4 Ejemplo 4.

Crear un mapeo para sumar los elementos de tres listas que contienen números enteros.

NOTA. La función `map()` se puede aplicar a más de un conjunto iterable, siempre y cuando los iterables tengan la misma longitud y la función que se aplique tenga los parámetros correspondientes.

```
def suma(x,y,z):
    """
    Suma los números x, y, z.
    """
    return x+y+z

# Tres listas con enteros
a = [1,2,3,4]
b = [5,6,7,8]
c = [9,10,11,12]
```

```
# Aplicación del mapeo
print(list(map(suma, a,b,c)))
```

[15, 18, 21, 24]

15 Filtrado.

- Filtrar es un procedimiento para seleccionar cosas de un conjunto o para impedir su paso libremente.
- En matemáticas, un filtro es un subconjunto especial de un conjunto parcialmente ordenado.

filter

$$\begin{array}{ccc} \left[\begin{array}{c} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_{n-1} \end{array} \right] & \begin{array}{c} \text{True} \\ \rightarrow \\ \\ \text{True} \\ \rightarrow \\ \\ \text{True} \\ \rightarrow \end{array} & \left[\begin{array}{c} - \\ f_1 \\ - \\ f_2 \\ f_{m-1} \end{array} \right] \end{array}$$

15.1 filter.

En Python existe la función `filter(function, sequence)` cuyo primer parámetro es una función la cual se va a aplicar a una secuencia, la cual es el segundo parámetro. La función debe regresar un objeto de tipo Booleano: `True` o `False`. El resultado será una nueva secuencia con los elementos obtenidos de aplicar la función a cada elemento de la secuencia de entrada.

15.2 Ejemplo 5.

Usando la función `filter()`, encontrar los números pares en una lista.

```
def esPar(n):
    """
    Función que determina si un número es par o impar.
    """
    if n%2 == 0:
        return True
    else:
        return False
```

```
# Probamos la función
print(esPar(10))
```

```
print(esPar(9))
```

True
False

```
# Creamos una lista de números, del 0 al 19
numeros = list(range(20))
print(numeros)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

```
# Aplicamos el filtro
print(list(filter(esPar, numeros)))
```

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

15.3 Ejemplo 6.

Encontrar los números pares en una lista que contiene elementos de muchos tipos.

Paso 1. Creamos la lista.

```
# Creamos la lista
lista = ['Hola', 4, 3.1416, 3, 8, ('a', 2), 10, {'x': 1.5, 'y': 12} ]
print(lista)
```

['Hola', 4, 3.1416, 3, 8, ('a', 2), 10, {'x': 1.5, 'y': 12}]

Paso 2. Escribimos una función que verifique si una entrada es de tipo `int`.

```
def esEntero(i):
    """
    Función que determina si un número es entero.
    """
    if isinstance(i, int): # Checamos si la entrada es de tipo int
        return True
    else:
        return False
```

```
print(esEntero("Hola"))
print(esEntero(1))
print(esEntero(1.))
```

False

True

False

Una forma alternativa, más Pythonica, de construir la función `esEntero()` es la siguiente:

```
def esEntero(i):
    return True if isinstance(i, int) else False
```

Paso 3. Usamos la función `esPar()` para encontrar los pares de la lista.

```
print(list(filter(esPar, list(filter(esEntero, lista)))))
```

[4, 8, 10]

Observa que se aplicó dos veces la función `filter()`, la primera para determinar si el elemento de la lista es entero usando la función `esEntero()`, la segunda para determinar si el número es par.

15.4 Ejemplo 7.

Encontrar los números primos en el conjunto $\{2, \dots, 50\}$.

```
# Función que crear una
def noPrimo():
    """
    Determina la lista de números que no son primos en el
    rango [2, 50]
    """
    np_list = []
    for i in range(2, 50):
        for j in range(i*2, 50, i):
            np_list.append(j)
    return np_list

no_primo = noPrimo()

print("Lista de números NO primos en el rango [2, 50] \n{}".format(no_primo))

def esPrimo(number):
    """
    Determina si un número es primo o no.
    """
    np_list = noPrimo()
    if(number not in np_list):
        return True
```



```
# Creación de la lista de números enteros de 2 a 50
numeros = list(range(2,50))

# Calculamos los primos usando filter(), con
# la función esPrimo() y la lista números.
primos = list(filter(esPrimo, numeros))

print("\nNúmeros primos en el rango [2, 50] \n {}".format(primos))
```

Lista de números NO primos en el rango [2, 50]

[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 10, 15, 20, 25, 30, 35, 40, 45, 12, 18, 24, 30, 36, 42, 48, 14, 21, 28, 35, 42, 49]

Números primos en el rango [2, 50]

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

16 Reducción.

- **Reducción** : Disminuir *algo* en tamaño, cantidad, grado, importancia, ..
- La operación de reducción es útil en muchos ámbitos, el objetivo es reducir un conjunto de objetos en un objeto más simple.

Una reducción se hace como sigue:

Dada la función $f()$ y la secuencia $[s_1, s_2, s_3, s_4]$ se tiene que

$$\begin{array}{ccc} \left[\underbrace{s_1, s_2}_{a=f(s_1, s_2)}, s_3, s_4 \right] & \implies & f(\underbrace{f(\underbrace{f(s_1, s_2), s_3}_b), s_4}_c) \\ \underbrace{b=f(a, s_3)} & & \\ \underbrace{c=f(b, s_4)} & & \end{array}$$

16.1 filter.

En Python existe la función `reduce(function, sequence)` cuyo primer parámetro es una función la cual se va a aplicar a una secuencia, la cual es el segundo parámetro. La función debe regresar un objeto que es el resultado de la reducción.

16.2 Ejemplo 8.

Calcular la siguiente serie:

$$1 + 2 + \dots + n = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

Si $n = 4$ entonces $1+2+3+4 = 10$

```
# La función reduce() debe importarse del módulo functools
from functools import reduce
```

```
# Creamos la lista
nums = [1,2,3,4]
print(nums)

# Calculamos la serie usando reduce y una función lambda
suma = reduce(lambda x, y: x + y, nums)
print(suma)
```

```
[1, 2, 3, 4]
10
```

```
# Se pueden usar arreglos de numpy
import numpy as np

# Construimos un arreglo de 1's
a = np.ones(20)
print(a)

suma = reduce(lambda x, y: x + y, a)
print(suma)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
20.0
```

16.3 Ejemplo 9.

Calcular la siguiente serie:

$$\sum_{i=1}^n \frac{1}{i} = \sum_{i=1}^n \frac{1}{i}$$

```
numeros = [3,4,5]
result = reduce(lambda x, y: 1/x + 1/y, numeros)
print(result)
```

```
1.9142857142857144
```

16.4 Ejemplo 10.

Calcular el máximo de una lista de números.

```
numeros = [23,5,23,56,87,98,23]
maximo = reduce(lambda a,b: a if (a > b) else b, numeros)
print(maximo)
```

98

16.5 Ejemplo 11.

Calcular el factorial de un número.

17 Más ejemplos Pythonicos.

17.1 Ejemplo 12.

Convertir grados Fahrenheit a Celsius y viceversa combinando `map()` con `lambda`.

```
c = [0, 22.5, 40,100]

# Conversión a Fahrenheit
f = list(map(lambda T: (9/5) * T + 32, c))
print(f)

# Conversión a Celsius
print(list(map(lambda T: (5/9)*(T - 32), f)))
```

[32.0, 72.5, 104.0, 212.0]

[0.0, 22.5, 40.0, 100.0]

17.2 Ejemplo 13.

Sumar tres arreglos combinando `map()` con `lambda`.

```
a = [1,2,3,4]
b = [5,6,7,8]
c = [9,10,11,12]

print(list(map(lambda x,y,z : x+y+z, a,b,c)))
```

[15, 18, 21, 24]

17.3 Ejemplo 14.

Encontrar todos los números pares de una lista combinando `filter()` con `lambda`.

```
# Lista de números
nums = [0, 2, 5, 8, 10, 23, 31, 35, 36, 47, 50, 77, 93]

# Aplicación de filter y lambda
result = filter(lambda x : x % 2 == 0, nums)

print(list(result))
```

[0, 2, 8, 10, 36, 50]

17.4 Ejemplo 15.

Encontrar todos los números primos en el conjunto $\{2, \dots, 50\}$ combinando combinando `filter()` con `lambda`.

```
# Lista de números de 2 a 50
nums = list(range(2, 51))

# Cálculo de los números primos usando
# filter y lambda
for i in range(2, 8):
    nums = list(filter(lambda x: x == i or x % i, nums))

print(nums)
```

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

17.5 Ejemplo 16.

Contar el número de caracteres de un texto, combinando `reduce()`, `map()` y `lambda`.

```
# Contar los caracteres de una cadena
texto = 'Hola Mundo'

palabras = texto.split()
print(palabras)

# Conteo de caracteres
print(reduce(lambda x,y: x+y, list(map(lambda palabras: len(palabras), palabras))
```

['Hola', 'Mundo']

9

```
# Contar los caracteres de un texto en un archivo
archivo = open('QueLesQuedaALosJovenes.txt', 'r')

suma = 0
for linea in archivo:
    palabras = linea.split()
    suma += reduce(lambda x,y: x+y, list(map(lambda palabras: len(palabras), palabras)))
print(suma)
archivo.close()
```

824

Lo anterior se puede realizar si construir una función que cuenta los caracteres de una lista de cadenas:

```
def cuentaCaracteres(palabras):
    return reduce(lambda x,y: x+y, list(map(lambda palabras: len(palabras), palabras)))
```

```
texto = 'Hello Motto'.split()
cuentaCaracteres(texto)
```

10

```
# Contar los caracteres de un texto en un archivo
archivo = open('QueLesQuedaALosJovenes.txt', 'r')

suma = 0
```

```
for linea in archivo:
    palabras = linea.split()
    suma += cuentaCaracteres(palabras)
print(suma)
archivo.close()
```

824

17.6 Ejemplo 17.

La siguiente función es impura porque modifica la **lista**:

```
def cuadradosImpuros(lista):
    for i, v in enumerate(lista):
        lista[i] = v ** 2
    return lista

numeros_originales = list(range(5))
print(numeros_originales)
print(cuadradosImpuros(numeros_originales))
print(numeros_originales)
```

La salida del código anterior es el siguiente:

```
[0, 1, 2, 3, 4]
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
```

Escribe una versión pura de la función **cuadradosImpuros(lista)** usando **map()** y **lambda**.

Una manera alternativa es la siguiente:

```
numeros_originales = list(range(5))
print(numeros_originales)
print(list(map(lambda x: x ** 2, numeros_originales)))
print(numeros_originales)
```

```
[0, 1, 2, 3, 4]
[0, 1, 4, 9, 16]
[0, 1, 2, 3, 4]
```