EjerciciosReduce

June 19, 2023

1 Código 5.1

Las tres partes de reduce

[31]:
```
from functools import reduce

xs = [10, 5, 1, 19, 11, 203]

reduce(my_add, xs, 0)
```

[31]:
```
249
```

2 Código 5.2

Una función acumulador para la suma

[26]:
```
def my_add(acc, nxt):
    return acc + nxt
```

3 Código 5.3

Una función *lambda* dentro de la sumatoria *reduce*

[28]:
```
from functools import reduce

xs = [10, 5, 1, 19, 11, 203]

print(reduce(lambda acc, nxt: acc + nxt, xs, 0))
```

```
249
```

4 Código 5.4

Las funciones lambda pueden ser utilizadas para exponer valores contenidos en una clase (diccionario)

[30]:
```python
from functools import reduce

my_products = [
    {"price": 9.99, "sn": '00231'},
    {"price": 59.99, "sn": '11010'},
    {"price": 74.99, "sn": '00013'},
    {"price": 19.99, "sn": '00831'},
]

reduce(lambda acc, nxt: acc + nxt.get("price", 0), my_products, 0)
```

[30]:
```
164.96
```

5 Código 5.5

Sumatoria de una lista en un flotante mediante *reduce*

[3]:
```python
from functools import reduce

xs = [10, 5, 1, 19, 11, 203]

print(reduce(lambda acc, nxt: acc+nxt, xs, 0.0))

# La suma se hace en pares y se acumula el resultado, esto es:
# add(0.0, 10) => 10.0
# add(10.0, 5) => 15.0
# add(15.0, 1) => 16.0
# add(16.0, 19) => 35.0
# add(35.0, 11) => 46.0
# add(46.0, 203) => 249.0
```

```
249.0
```

6 Código 5.6

Filtrar valores impares

[4]:
```python
from functools import reduce

xs = [i for i in range(10)]

def keep_if_even(acc, nxt):
    if nxt%2 == 0:
        return acc + [nxt]
        # No se usa append dado que es indispensable para esta
        # construcción que la función devuelva la lista como
        # resultado. Append devuelve None, por ello no sirve
```

```
        # para este propósito.
    else:
        return acc

reduce(keep_if_even, xs, [])
```

[4]: [0, 2, 4, 6, 8]

6.1 Agregar un elemento a una lista.

No se usa append dado que es indispensable para esta construcción que la función devuelva la lista como resultado. Append devuelve None, por ello no sirve para este propósito.

[5]:
```
lista = [2,3,4]
lista + [5]
```

[5]: [2, 3, 4, 5]

7 Código 5.7

Encontrar las frecuencias utilizando una reducción

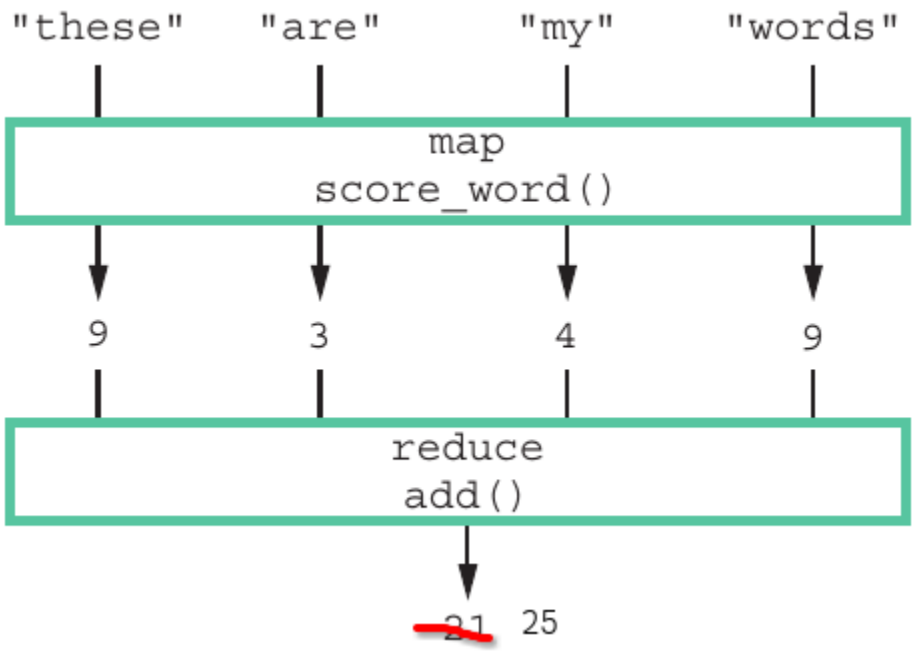
[8]:
```
from functools import reduce

def make_counts(acc, nxt):
    acc[nxt] = acc.get(nxt, 0) + 1
    return acc

def my_frequencies(xs):
    return reduce(make_counts, xs, {})

xs = ["A", "B", "C", "A", "A", "C"]
ys = [1, 3, 6, 1, 2, 5, 4, 2, 4]

print(my_frequencies(xs))
print(my_frequencies(ys))
print(my_frequencies("mississipi"))
```

```
{'A': 3, 'B': 1, 'C': 2}
{1: 2, 3: 1, 6: 1, 2: 2, 5: 1, 4: 2}
{'m': 1, 'i': 4, 's': 4, 'p': 1}
```

Se crea un diccionario vacío en el *reduce* de la función my_frequencies. En este diccionario se almacenarán las letras (llave) y la frecuencia (valor) de aparición en la lista de entrada.

8 Ejercicio 1.

Dada una lista de entrada con palabras, crear una puntuación para cada letra de cada palabra en la lista y almacenar la puntuación en una lista. Crear un código en Python para realizar este proceso mediante la metodología *map-reduce*.



Considere el siguiente puntaje: * Z: 10 puntos * F, H, V, W: 5 puntos * B, C, M, P: 3 puntos * Cualquier otra letra: 1 punto

9 Código 5.8

Puntuación de palabras con *map* y *reduce*

[12]:
```
from functools import reduce

def score_word(word):
    points = 0
    for char in word:
        if char == "z": points += 10
        elif char in ["f", "h", "v", "w"]: points += 5
        elif char in ["b", "c", "m", "p"]: points += 3
        else: points += 1
    return points

words = ["these", "are", "my", "words", "but", "these", "are", "not", "myself"]

total_score = reduce(lambda acc, nxt: acc + nxt, map(score_word, words), 0)
print(total_score)
```

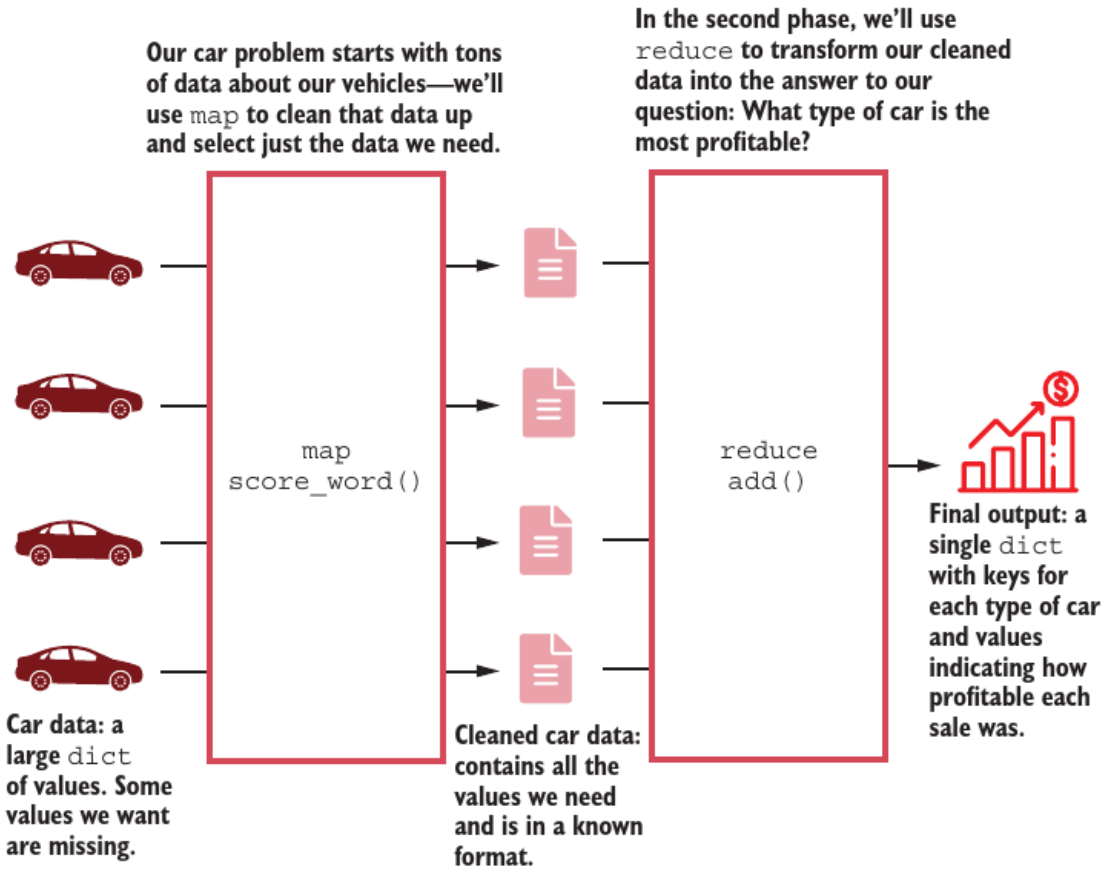
10 Ejercicio 2

Su cliente es una agencia de autos usados. La agencia tiene datos de autos que compraron y vendieron en los últimos seis meses y desean que se determine con que tipo de auto obtienen la mayor ganancia.

Un vendedor cree que los autos que ofrecen mejor desempeño de combustible proporcionan la mayor ganancia, mientras que otro vendedor cree que los autos que tienen un uso entre 60,000 y 100,000 millas ocasionan la mayor ganancia.

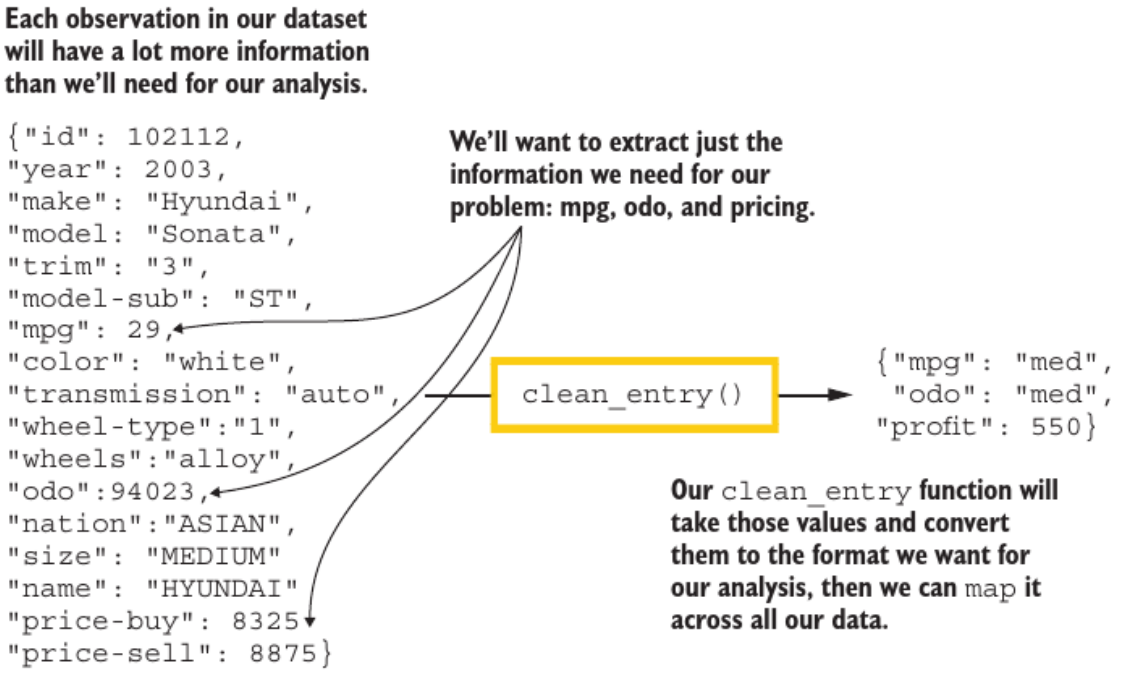
Dado un archivo CSV con una diversidad de atributos acerca de los autos usados, escriba un script que determine la ganancia promedio en autos de baja (<18 mpg), media (18-35 mpg) y alta (>35 mpg) eficiencia de consumo de combustible, así como de bajo (<60,000 millas), medio (60,000-100,000 millas) y alto (>100,000 millas) uso en millas.

10.1 Análisis para el ejercicio 2



}

10.2 Muestra de los datos de entrada



De los datos de entrada se necesita: 1. Calcular la ganancia obtenida por el vehículo (valor de compra y venta) 2. Ordenar los vehículos respecto a su consumo de combustible (mpg) en los rangos: bajo, medio y alto 3. Ordenar los vehículos respecto a su uso (millas totales) en los rangos: bajo, medio y alto

Para ello se desarrollaran funciones independientes que realicen esa tarea.

11 Código 5.9

Función que calcula la ganancia mediante los valores obtenidos en el diccionario de entrada

[1]:
```
def get_profit(d):
    return d.get("price-sell", 0) - d.get("price-buy", 0)
```

12 Código 5.10

Una función genérica bajo-medio-alto

[2]:
```
def low_med_hi(d, k, low, high):
    if d[k] < low:
        return "low"
    elif d[k] < high:
        return "med"
    return "high"
```

Ya con esta función escrita, aún se requiere hacer lo siguiente: 1. Tomar un diccionario 2. Limpiar el diccionario con la función *select_keys* 3. Devolver un diccionario que tenga tres llaves

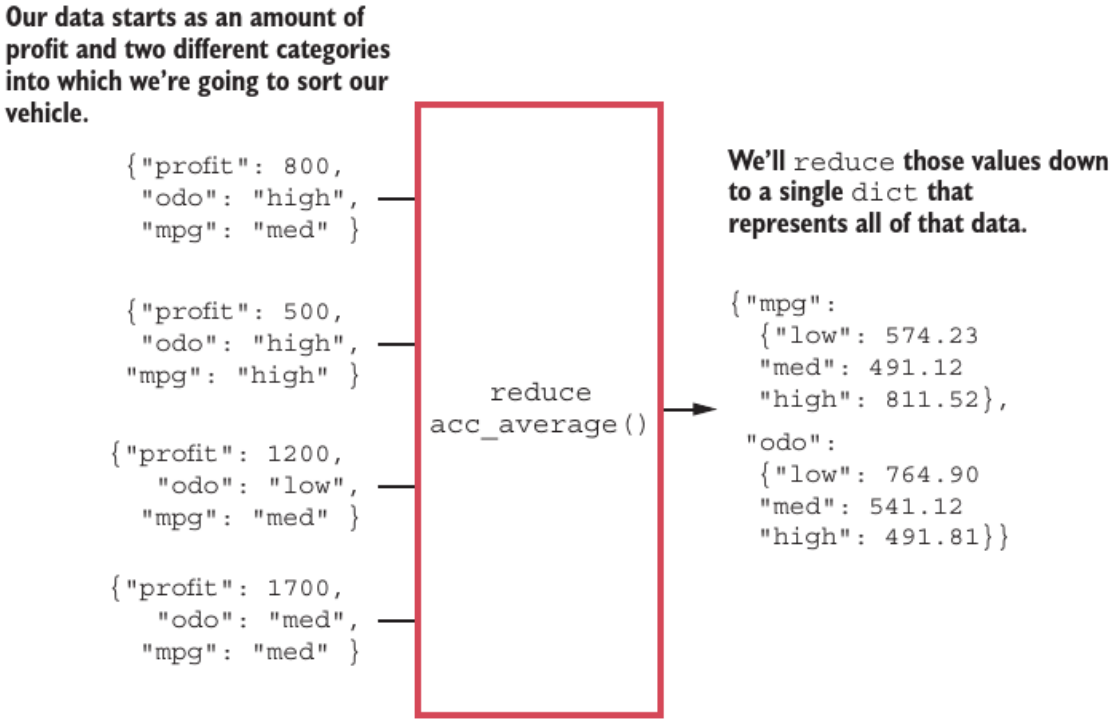
a. Llave *profit* que indique la ganancia obtenida por el vehículo
b. Llave *mpg* que indique la categoría del consumo de combustible del vehículo
c. Llave *odo* que indique el uso del vehículo

13 Código 5.11

Uniendo las funciones *helper* en una sola

[10]:
```
def clean_entry(d):
    r = {}
    r['profit'] = get_profit(d)
    r['mpg'] = low_med_hi(d, 'mpg', (18, 35)) # Se utilizará una tupla para low␣
 ↪y high
    r['odo'] = low_med_hi(d, 'odo', (60000, 100000))
    return r
```

13.1 Reducción muchos a un diccionario



Esta función debe devolver un sólo diccionario con seis llaves: bajo, medio y alto para *mpg* y *odo*.

14 Código 5.12

Acumulador promedio de ganancias y funciones *helper*

La función toma el total acumulado, un contador y el promedio de la categoría del auto y los mezcla para el nuevo auto. También incrementa el contador y calcula de nuevo el promedio.

[7]:
```
# Cálculo de la ganacia promedio
def acc_average(acc, profit):
    acc['total'] = acc.get('total', 0) + profit
    acc['count'] = acc.get('count', 0) + 1
    acc['average'] = acc.get['total']/acc['count']
    return acc

# Se busca el elemento dado por _nxt_ en el diccionario y si no se encuentra
# se crea uno vacío.
def sort_and_add(acc, nxt):
    profit = nxt['profit']
    nxt_mpg = acc['mpg'].get(nxt['mpg'], {})
    nxt_odo = acc['odo'].get(nxt['odo'], {})
    acc['mpg'][nxt['mpg']] = acc_average(nxt_mpg, profit)
    acc['odo'][nxt['odo']] = acc_average(nxt_odo, profit)
    return acc
```

Se utilizará *map* para aplicar la función *clean_entry* para cada elemento en los datos, resultando así una secuencia limpia de datos que están listos para el *reduce*.

Ahora se llamará a la función *reduce* con sus tres parámetros: 1. La función acumulador 2. Los datos 3. Un inicializador opcional

Para la función acumulador se utiliza *sort_and_add*. Para los datos se utilizan los resultados de la operación *map*. Para el inicializador se utiliza un diccionario vacío.

15 Código 5.13

Map y *reduce* para encontrar la ganancia promedio de autos usados

[15]:
```
from functools import reduce

def low_med_hi(d, k, breaks):
    if float(d[k]) < breaks[0]:
        return "low"
    elif float(d[k]) < breaks[1]:
        return "medium"
    else:
        return "high"

def clean_entry(d):
    r = {'profit': None, 'mpg': None, 'odo': None}
    r['profit'] = float(d.get("price-sell", 0)) - float(d.get("price-buy", 0))
    r['mpg'] = low_med_hi(d, 'mpg', (18, 35))
    r['odo'] = low_med_hi(d, 'odo', (60000, 100000))
    return r
```

```
def acc_average(acc, profit):
    acc['total'] = acc.get('total', 0) + profit
    acc['count'] = acc.get('count', 0) + 1
    acc['average'] = acc['total']/acc['count']
    return acc

def sort_and_add(acc, nxt):
    p = nxt['profit']
    acc['mpg'][nxt['mpg']] = acc_average(acc['mpg'].get(nxt['mpg'], {}), p)
    acc['odo'][nxt['odo']] = acc_average(acc['odo'].get(nxt['odo'], {}), p)
    return acc

if __name__ == "__main__":
    import json
    with open("cars.json") as f:
        xs = json.load(f)
    results = reduce(sort_and_add, map(clean_entry, xs), {"mpg": {}, "odo": {}})
    print(json.dumps(results, indent=4))
```


```
{
    "mpg": {
        "medium": {
            "total": 14587.98,
            "count": 174,
            "average": 83.83896551724138
        },
        "high": {
            "total": -5875.0,
            "count": 142,
            "average": -41.37323943661972
        },
        "low": {
            "total": 7007.0,
            "count": 34,
            "average": 206.08823529411765
        }
    },
    "odo": {
        "medium": {
            "total": 33576.0,
            "count": 255,
            "average": 131.6705882352941
        },
        "low": {
            "total": -18531.02,
            "count": 85,
            "average": -218.012
```

```
        },
        "high": {
            "total": 675.0,
            "count": 10,
            "average": 67.5
        }
    }
}
```