Python for Data Analytics

Functional Programming

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

Imperative vs. Declarative Programming

- Imperative programming
  - Focuses on *how* a program operates
  - Tell the computer what steps to take to solve a problem
  - Procedural languages (Fortran, C, Pascal, etc.)
  - Object-oriented languages (C++, Java, etc.)

- Declarative programming
  - Focuses on *what* the program should accomplish
  - Tell the computer what result you want
  - Functional languages (Haskell, Lisp, etc.)
  - Logic languages (Prolog, etc.)

Higher-order Functions

- Functions either accept a function as an argument or return a function for further processing

```python
def hof_write_n(msg, n, action):
    for i in range(n):
        action(msg)

hof_write_n('Hello', 5, print)
import logging
hof_write_n('Hello', 5, logging.error)
```

Lambda Expressions

- A lambda expression is an anonymous function
- Allow us to define a function much more easily

```python
def hof_product(multiplier):
    return lambda x: x * multiplier

mult6 = hof_product(6)
print(mult6(10))

f = lambda x, y: x + y
print(f(3, 4))
```

Higher-order Functions in Python

- Commonly used higher-order functions from functional programming languages:
  - `zip()` – built-in
  - `filter()` – built-in
  - `map()` – built-in
  - `reduce()` – import `functools`

- Make processing iterable objects such as lists and tuples much easier
- For space/memory efficiency reasons, return an iterator instead of a list
  - Iterator yields a number of objects in a specific order, often creating them on the fly as requested

zip()

- `zip`(**iterables*)
  - Make an iterator that aggregates elements from each of the *iterables*
  - The * operator can be used to unzip a list

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> xy = list(zip(x,y))
>>> xy
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*xy)
>>> x2
(1, 2, 3)
>>> list(y2)
[4, 5, 6]
```

filter()

- `filter`(*function*, *iterable*)
  - Construct an iterator from those elements of *iterable* for which function returns True
  - If function is None, the identity function is assumed, that is, all elements of *iterable* that are False are removed

```
>>> numbers = [1, 1, 2, 3, 5, 8, 13, 21, 34]
>>> result = filter(lambda x: x % 2, numbers)
>>> print(list(result))
[1, 1, 3, 5, 13, 21]
>>> for i in filter(lambda x: x not in 'aeiou', 'ham'):
...     print(i)
h
m
```

map()

- map(*fun*, *iter*)
  - Return an iterator that applies *function* to every item of *iterable* yielding the results
  - `map(f, [a, b, c, ...])` → `[f(a), f(b), f(c), ...]`

```
>>> def inc(x):
...     return x + 1
>>> a = [1, 2, 3]
>>> list(map(inc, a))
[2, 3, 4]
>>> list(map(lambda x: x+x, a))
[2, 4, 6]
>>> l = ['sat', 'bat', 'cat']
>>> list(map(list, l))
[['s', 'a', 't'], ['b', 'a', 't'], ['c', 'a', 't']]
```

reduce()

- *functools.*`reduce`*(function, iterable, ...)*
  - Apply *function* of two arguments cumulatively to the items of *iterable*, from left to right, so as to reduce the *iterable* to a single value
  - `reduce(g, [a, b, c, ...])` → `g(g(g(a,b), c), ...)`

```
>>> import functools as f

>>> l = [1, 3, 5, 6, 2]
>>> f.reduce(lambda a, b: a + b, l)
17
>>> f.reduce(lambda a, b: a if a > b else b, l)
6
```

List Comprehension

List Comprehension

- Provides a concise way to create lists

```
new_list = list()
for i in range(10):
    if i % 2 == 0:
        new_list.append(i*i)
```

```
new_list = [ i*i for i in range(10) if i % 2 == 0 ]
```

List Comprehension: General Form

```
new_list = [ expression(i) for i in old_list if filter(i) ]
```

```
new_list = list()
for i in old_list:
    if filter(i):
        new_list.append(expression(i))
```

Examples (1)

```
>>> [ i for i in range(10) ]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [ x**2 for x in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [ item*3 for item in [2, 3, 5] ]
[6, 9 ,15]
>>> [ word[0] for word in ['hello', 'world', 'spam' ]]
['h', 'w', 's']
>>> [ x.upper() for x in ['spam', 'ham', 'egg'] ]
['SPAM', 'HAM', 'EGG']
```

Examples (2)

```
>>> [ i if i > 0 else 0 for i in [-2, 5, 4, -7] ]
[0, 5, 4, 0]
>>> fh = open('genesis.txt', 'r')
>>> [ i for i in fh if 'heaven' in i ]
['1:1 In the beginning God created the heaven and the earth. \n', '1:9
And God said, Let the waters under the heaven be gathered together unto
one place, and let the dry land appear: and it was so. \n', ... ]
>>> [ x + y for x in [10, 30, 50] for y in [20, 40, 60] ]
[30, 50, 70, 50, 70, 90, 70, 90, 110]
>>> [ (x, y) for x in [1, 2, 3] for y in [7, 8, 9] ]
[(1, 7), (1, 8), (1, 9), (2, 7), (2, 8), (2, 9), (3, 7), (3, 8), (3, 9)]
```

Examples (3)

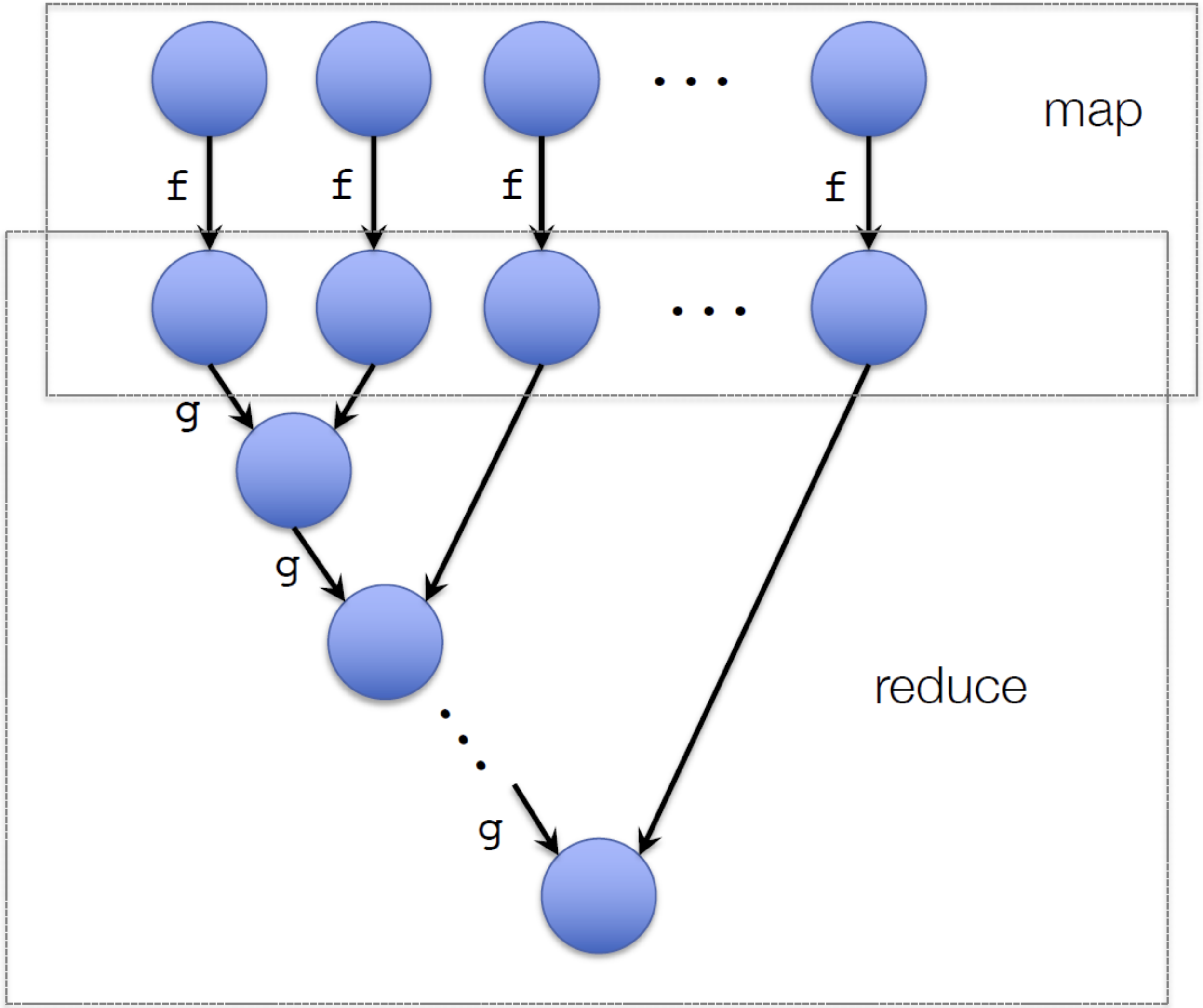
```
>>> [ [0]*4 for _ in range(3) ]
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

>>> [ [i for i in range(4)] for _ in range(3) ]
[[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]

>>> matrix = [ [i for i in range(j, j+4)] for j in range(0, 12, 4)]
>>> matrix
[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

>>> [ e for row in matrix for e in row ]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Examples (4)

```
>>> word1 = 'hi'
>>> word2 = 'sun'
>>> [ i + j for i in word1 for j in word2 ]
['hs', 'hu', 'hn', 'is', 'iu', 'in']

>>> [ [i + j for i in word1] for j in word2 ]
[['hs', 'is'], ['hu', 'iu'], ['hn', 'in']]

>>> [ [w.upper(), w.lower(), len(w)] for w in [word1, word2] ]
[['HI', 'hi', 2], ['SUN', 'sun', 3]]

>>> { i: i*i for i in range(5) }      // dictionary comprehension
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

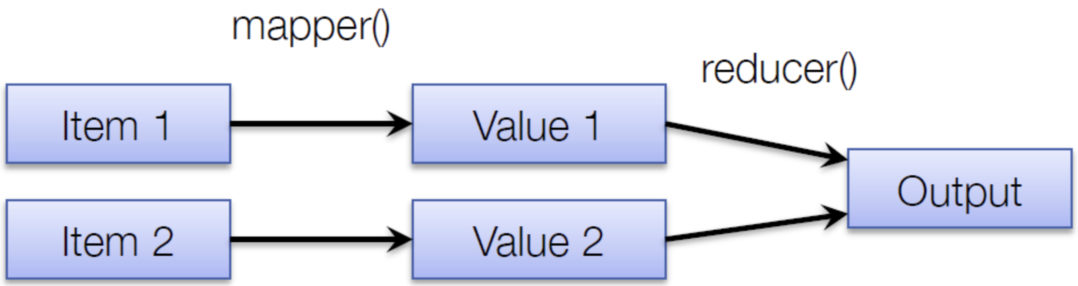
Word Count with Map+Reduce

Map and Reduce

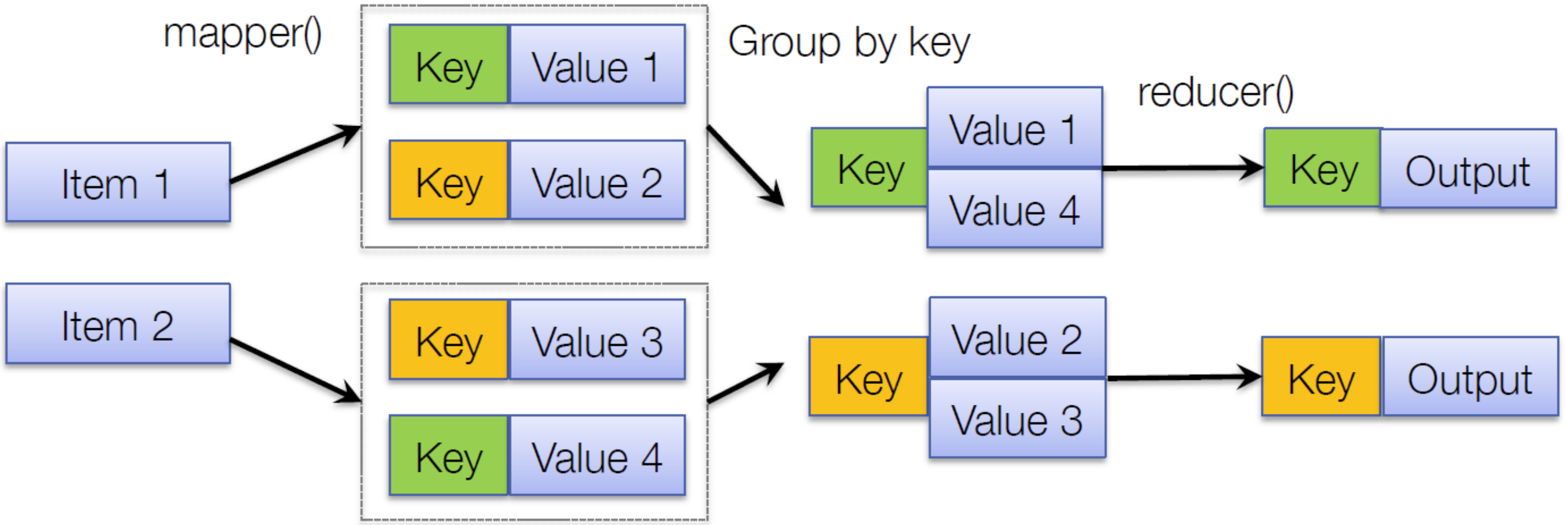


MapReduce

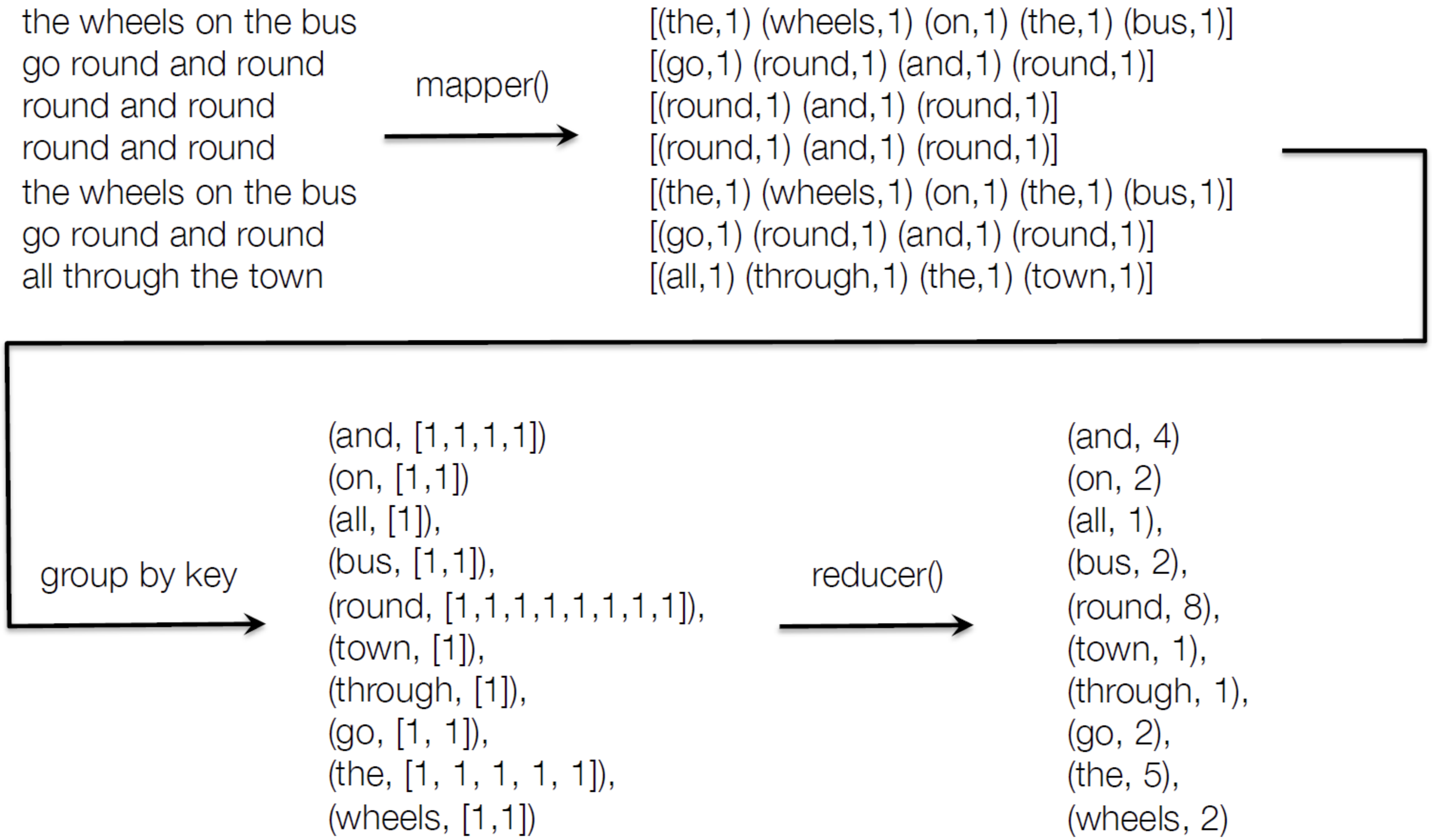
map + reduce



MapReduce



MapReduce: Word Count



the wheels on the bus
go round and round
round and round
round and round
the wheels on the bus
go round and round
all through the town

mapper()

[(the,1) (wheels,1) (on,1) (the,1) (bus,1)]
[(go,1) (round,1) (and,1) (round,1)]
[(round,1) (and,1) (round,1)]
[(round,1) (and,1) (round,1)]
[(the,1) (wheels,1) (on,1) (the,1) (bus,1)]
[(go,1) (round,1) (and,1) (round,1)]
[(all,1) (through,1) (the,1) (town,1)]

group by key

(and, [1,1,1,1])
(on, [1,1])
(all, [1]),
(bus, [1,1]),
(round, [1,1,1,1,1,1,1,1]),
(town, [1]),
(through, [1]),
(go, [1, 1]),
(the, [1, 1, 1, 1, 1]),
(wheels, [1,1])

reducer()

(and, 4)
(on, 2)
(all, 1),
(bus, 2),
(round, 8),
(town, 1),
(through, 1),
(go, 2),
(the, 5),
(wheels, 2)

Mapper and Reducer

```python
def mapper_wc(line):
    return [(word, 1) for word in line.rstrip().split()]

def reducer_sum(k, v):
    return (sum(v), k)

fh = open('genesis.txt')
words = do_mapreduce(fh.readlines(), mapper_wc, reducer_sum)
sorted_words = sorted(words, reverse=True)
for v, k in sorted_words[:10]:
   print(v, k)
```

MapReduce Execution Engine

```python
def do_mapreduce(data, mapper, reducer):
    values = map(mapper, data)

    groups = dict()
    for items in values:
        for k, v in items:
            if k not in groups:
                groups[k] = [v]
            else:
                groups[k].append(v)
    output = [reducer(k, v) for k, v in groups.items()]
    return output
```