Chapter 7. List and Tuples

`map/zip/filter/reduce`

Starting out with Python

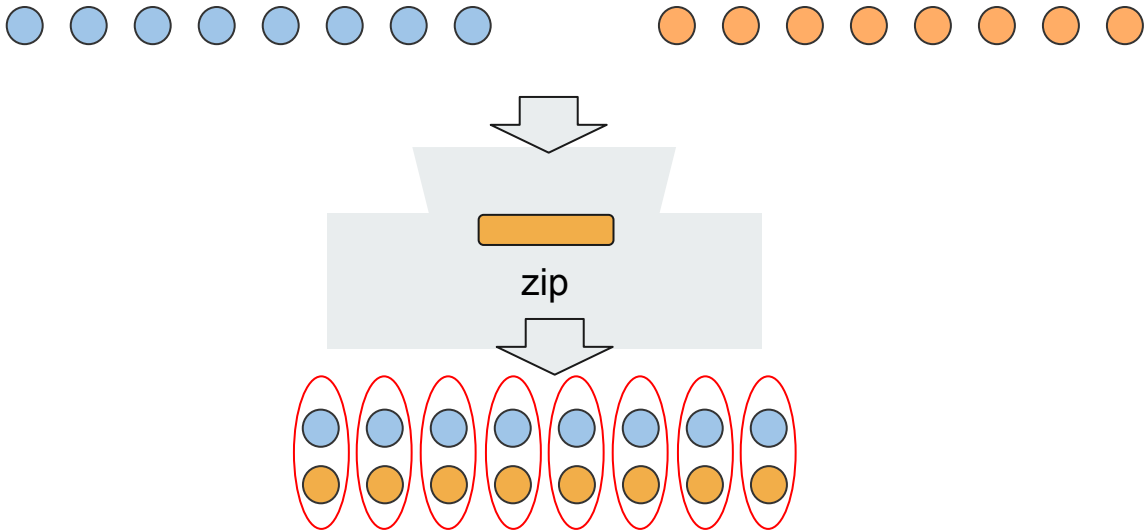
Kyu Lee. Ph. D.

Computer Science

zip

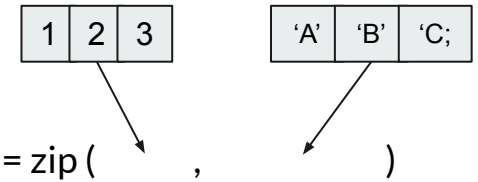
zip

- zip
  - takes Iterable objects or containers
  - return a single iterator, having mapped values from all the containers



zip

- zip
  - takes Iterable objects or containers
  - return a single iterator, having mapped values from all the containers



zipped_objects = zip (,)

1	'A'
2	'B'
3	'C;

```
for zipval in zipped_objects:
    print (zipval)
```

zip

Lab 7

- The list of student's names = ['Bill', 'John', 'Kurt']
- The list of score values = [100, 90, 90]

Make a zip statement and print the paired values with a zip object

zip

Lab 8

- The list of course ID =[1001, 1002, 1003]
- The list of course Name = ['C Programming', 'Java Programming', 'Python Programming']

Make a zip statement and print the paired values with a zip object

https://github.com/LPC-CSDept/CS7L98

Lab 9 : zip to dictionary

- Write a function `makeDict(heading, valueset)` to construct a dictionary from two list values 'heading' and 'valueset'
- **Return** value:
  - a list of **dictionary** made with two list values
- **Requirement**
  - Use `zip( )`

Make a list of dictionaries from two lists

```
heading = ['id', 'name', 'address']

valueset = [ [10, 'Kim', '123 Main'],
             [20, 'Bill', '345 Grand'],
             [30, 'Mary', '123 Blvd']
           ]
```

Tip: Try this statement
`dict(zip(heading, valueset[0]))`

heading

ID	Name	Address

valueset

10	Kim	123 Main
20	Bill	345 Grand
30	Mary	123 Blvd

Return value

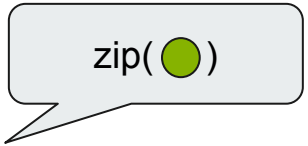
```
[{'id': 10, 'name': 'Kim', 'Address': '123 Main'},
 {'id': 20, 'name': 'Bill', 'Address': '345 Grand'},
 {'id': 30, 'name': 'Mary', 'Address': '123 Blvd'}]
```

* operator in zip ( )

to unzip

Lab 10: zip(*)

```
# Student's information: ID, Name, GradeLevel, Zip
student_list = [ [1001, 'Bill', 'Senior', 94568],
                 [1002, 'Kurt', 'Junior', 94598],
                 [1003, 'Kim', 'Senior', 94598] ]

print (student_list)

for value in zip(*student_list):
        print (value)

IDlist, Namelist, Glist, Zlist = zip(*student_list)

print (IDlist)
print (Zlist)
```



vs

```
for v in zip(student_list):
        print (v)
```

```
([1001, 'Bill', 'Senior', 94568],)
([1002, 'Kurt', 'Junior', 94598],)
([1003, 'Kim', 'Senior', 94598],)
```

zip(*)

- Can we unzip a dictionary?
- Can we unzip a non-zip object?
    - such as list, dictionary, and tuple

Lab 10 : zip(*)

- Unzip Dictionary ?
  - use dictionary.items()

```
dictionary1 = { 10:"Kurt", 20:"Jim", 30:"Bill"}
print (dictionary1)

unzipped = zip(*dictionary1.items())

for values in unzipped:
        print (values)

IDlist, Namelist = zip(*dictionary1.items())
print (IDlist)   # tuple
print (Namelist) # tuple

IDlist?
```

```
dictionary1 = { 10:"Kurt", 20:"Jim", 30:"Bill"}
print (dictionary1)

IDlist, Namelist = zip(*dictionary1.items())
print (IDlist)   # tuple
print (Namelist) # tuple

IDlist?

for v in zip(*Namelist):
        print (v)
('K', 'J', 'B')
('u', 'i', 'i')
('r', 'm', 'l')
```

Lab 11: zip(*)

- Write a function `getColumn(numbers)` to take a list of list numbers as a parameter and returns a new list of lists representing the column values from the original input.
  - For example, [[10, 40, 70, 100], [20, 50, 80, 110], [30, 60, 90, 120]]
- **Requirement**
  - use **zip(*), see page 42 and 43**

```
numbers = [ [10, 20, 30],
            [40, 50, 60],
            [70, 80, 90],
            [100, 110, 120] ]
```

```
returnval = [[10, 40, 70, 100],
             [20, 50, 80, 110],
             [30, 60, 90, 120]]
```

zip(*)

- Unzip list ?

```
namelist = ['Kim', 'Bill', 'Kurt']


for v in zip(*namelist):
        print (v)
('K', 'B', 'K')
('i', 'i', 'u')
('m', 'l', 'r')

# What about integer list?
IDlist = [10, 20, 30]
zip(*IDlist) # Error. why?
```

zip(*)

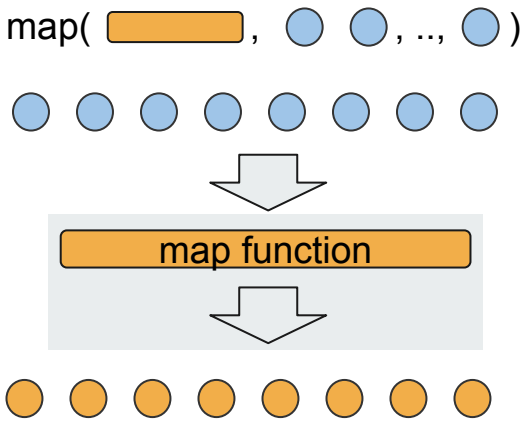
- unzip list of list ?

```
list1 = [ [10, 20, 30], ['Kim', 'Jim', 'Sam']]


for value in zip(*list1):
        print (value)
```

map

map

- map() is used to
  - apply a **function** on **all elements** of a specified iterable
  - return a **map object**
- map(func, [...])




```
def    square(val):
       return val * val

lst = [1,2,3,4,5]

sqrmap = map(square, lst)

for v in sqrmap:
       print (v)
```

map and lambda

- lambda function can be used in map()

```
sqr = lambda val : val * val

lst = [1,2,3,4,5]

sqrmap = map(sqr, lst)

for v in sqrmap:
       print (v, end=' ')
```

or

```
lst = [1,2,3,4,5]

sqrmap = map(lambda val : val * val, lst)

for v in sqrmap:
       print (v, end=' ')
```

map and list

- list as a map function

```
lst1 = list(‘python’)      # lst1 = [‘p’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’ ]
```

```
list('sat')

mylst = ['C++', 'Python', 'Java']

mapobj = map(list, mylst)
```

Lab 12 : map

- Write a function `halfValue(numbers)` that uses the map() function to divide each element in a list by 2.
  - Truncate the value when there is a fractional value
    - 5 // 2 = 2
  - Save your result as a list and return it

```
numbers = [ 10, 20, 30, 40, 50]


# Expected return value
[5, 10, 15, 20, 25]
```

Lab 13 : map

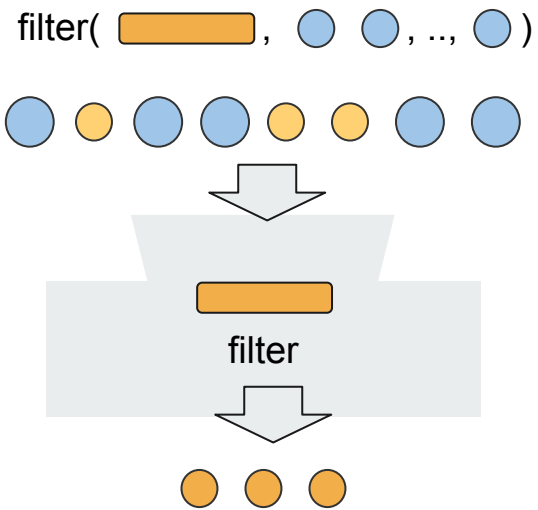
- Write a function `setOddNumber(numbers)` that uses the `map()` function to set to 1 if the element value is an odd number, otherwise set to 0
  - Save your result as a list and return it
- Requirement: use `map`
  - to modify a list, setting all even elements to 0 and all odd elements to 1.

```
mylist = [ 5, 10, 15, 20, 21, 25, 27]


# Expected output
[1, 0, 1, 0, 1, 1, 1]
```

filter

filter

- Filter
  - creates a list of elements for which a function returns true.
  - return an **filter object (iterator)**

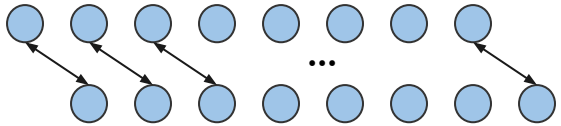



```
def evenfilter(val):
      return True if val % 2 == 0 else False

lst = [1,2,3,4,5]

filterobj = filter(evenfilter, lst)

# lst = list(filterobj)
for value in filterobj:
      print (value)
```

Lab 14 : filter

- Write a function called `gtRight(numbers)` that uses `filter` and `zip` to filter out elements that are greater than the element to their right. (exclude the last element)
  - Return value: the list
  - Requirement: use `filter, zip`

filter **list** **zip**

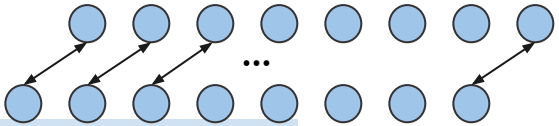



```
mylist = [ 5, 10, 15, 25, 20, 55, 40]

# Expected output
# return this list
[25, 55]
```

Lab 15 : filter

- Make a program using zip, filter and map
  - Write a function called `gtLeft(numbers)` that uses `filter` and `zip` to find elements in a list that are greater than the element to their left. Ignore the first element
  - Return value: the list
  - Requirement: use `filter, zip`

filter **list** **zip**

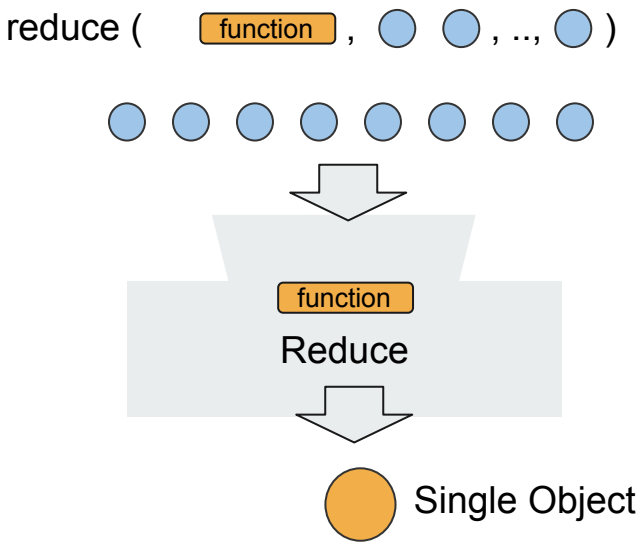



```
mylist = [ 5, 10, 15, 25, 20, 55, 40]

# Expected output
[10, 15, 25, 55]
```

reduce

Reduce

- Reduce
  - function for performing some computation on a iterable and
  - returns a single value

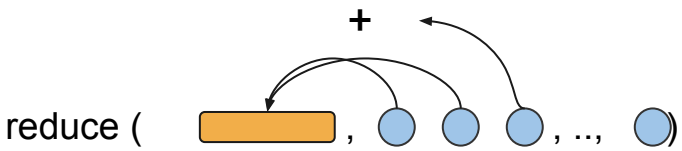



```
from functools import reduce

lst = [1,2,3,4,5,6,7,8,9,10]
# lst = [1,2,3,4, 5]

lstsum = lambda x, y: x + y

print (reduce(lstsum, lst))
```

Reduce

- Reduce
  - function for performing some computation on a iterable and
  - returns a single value

```
from functools import reduce

lst = [1,2,3,4,5,6,7,8,9,10]


lstsum = lambda x, y: x + y

print (reduce(lstsum, lst))
```



- **At first step, first two elements are calculated.**
- **Next, the result of the first step and the 3rd element are calculated**
- **And so on.**

Lab 16 : Reduce

- Write a function called `getMaxSum(numbers)` that takes a list of lists, `numbers`, and computes the sum of the maximum elements from each sublist in `numbers`."
- Requirement
    - Use `reduce( )` function and create your lambda function to compute the max element summation

```
mylist = [[1, 2, 3, 4, 5],
          [10, 20, 30, 40, 50],
          [100, 200, 1000]
          ]

# Expected output
1055
```

```
####
def

numb
maxs
prin
# Re
```

Lab 17 : Reduce

- Write a function called `getAvg(numbers)` that takes a list of lists, `numbers`, and computes the average of all elements in `numbers`."
- Requirement
  - Use `reduce`( ) function and create your lambda function to compute the summation

- Make a program using reduce
  - get the average of the list

```
mylist = [ 5, 10, 15, 25, 20, 55, 40]

# Expected output
24
```

Reduce

- Reduce
  - Using **operator**

```
from functools import reduce
import operator

mylist = [1,2,3,4,5]

mysum = reduce(operator.add, mylist)
print (mysum)

productoflist = reduce(operator.mul, mylist)
print (productoflist)
```

Summary of
zip, map, filter and reduce

Summary of zip, map, filter and reduce