LGT3109
Introduction to Coding for Business with Python
(week 8)

Xiaoyu Wang

Dept. of LMS, The HK PolyU

Summary of Week 7

- List is a sequence of any values
- Access element of list using [index]
- List is mutable, unlike string
- Two ways to traverse a list using *in*, len(), range()
- Simplifying a for loop using range()
- List Operations: +, * , *in, not in*
- Slice of a list: [start:end], [start:], [:end]
- Methods for List
  - list(), append(), extend(), pop(), del
- Built-In functions for list
  - max, min, sum, len

- List and string:
  - Similarity: sequence; Difference: mutable and immutable
  - String → List: split(); Double Split Pattern
  - List → String: join()
- Objects, values, references, aliasing
- Passing list as reference to a function

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Dictionary is like list, but more general

- In a list, the index must be integers.

➢votes = ['Joseph', 'Sally', 'Glenn', 'Sally', 'Joseph']

- In a dictionary, the indices can be many types.

➢scores = {'Joseph':2, 'Sally': 1, 'Glenn': 1}

- Both list and dictionary are collections!

What is a Collection?

- In a collection, we can put more than one value and carry them in one convenient package.

➢A single variable has a bunch of values.

➢More than one place in the variable.

What is Not a Collection?

- Most of variables have one value.
- A new value in the variable overwrites the old value.

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

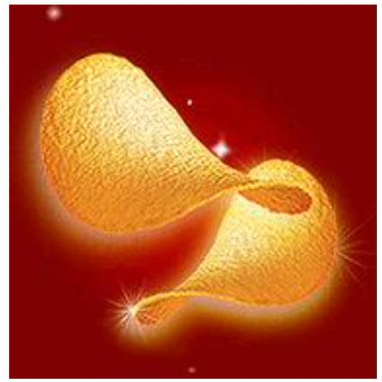
Comparing Two Collections

- List

➢A linear collection of values that stay in order: 1st piece, 2nd piece, …

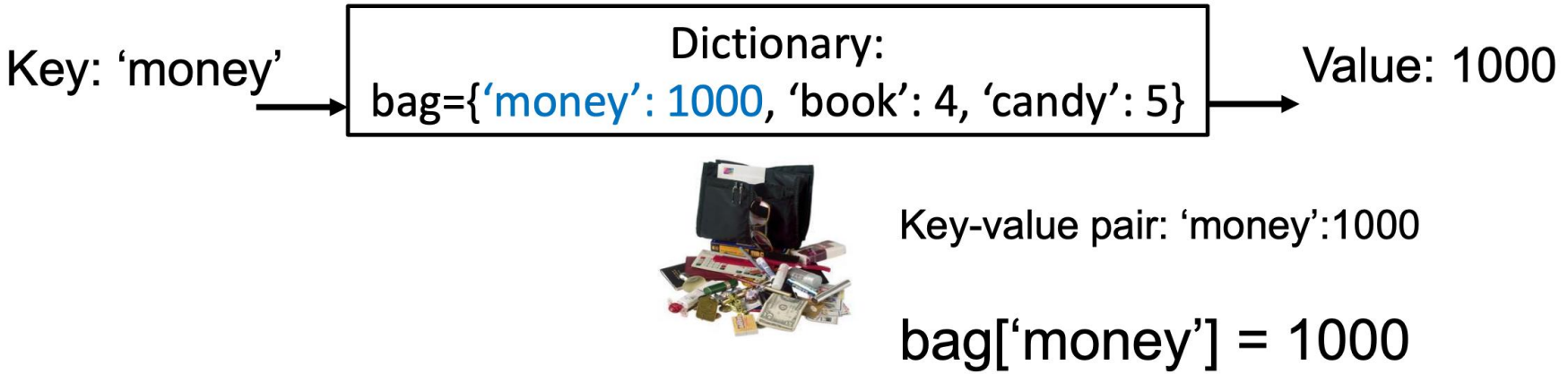
- Dictionary

➢A bag of values, each with its own label: money, calculator, …



Dictionaries

- A dictionary is a mapping between a set of indices (which are called keys) and a set of values.
- Each key maps to a value.
- The association of a key and a value is called a key-value pair (separated by colon:) or sometimes an item.



Dictionaries

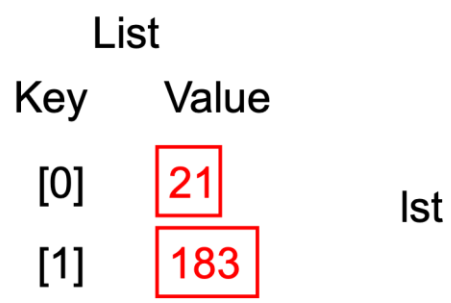
- List position is based on index - like a bookshelf.
- Dictionaries are like bags - no order.
- We index the things in the dictionary with a lookup tag (key).

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

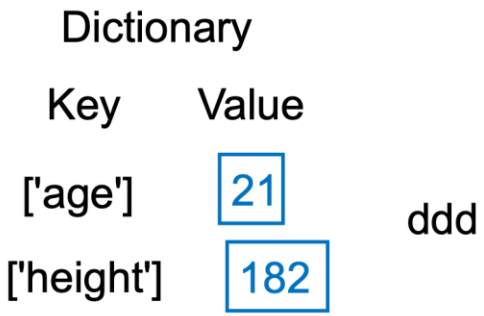
Comparing Lists and Dictionaries

- Dictionaries are like lists except that they use keys instead of numbers to look up values.

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```




```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['height'] = 182
>>> print(ddd)
{'age': 21, 'height': 182}
>>> ddd['age'] = 23
>>> print(ddd)
{'age': 23, 'height': 182}
```



Dictionary Literals

- Dictionary uses curly braces {} and list of keys: value pairs (items).
- Make an empty dictionary using empty curly braces.
- Use "=" to create an item that maps a key to a value.

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = {}
>>> print(ooo)
{}
>>> jjj['test']=1
>>> print(jjj)
{'jan': 100, 'chuck': 1, 'fred': 42, 'test':1}
```

Len() Function

- len() for a dictionary returns the number of its pairs (like list).

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print (len(jjj))
3
```

in Operator

- Like/unlike list, the in operator for a dictionary tells whether something appears as a key in the dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> 'fred' in jjj
True
```

pop and del Operator

- del:
  - ➢Remove specified key.
- pop(k):
  - ➢Remove specified key (k) and return the corresponding value.
- pop(k,d):
  - ➢Remove specified key (k) and return the corresponding value. If key is not found, d is returned.

```
>>> jjj = { 'chuck' : 1 , 'fred' :
42, 'jan': 100}
>>> del jjj['chuck']
>>> print(jjj)
{'fred' : 42, 'jan': 100}
>>> print(jjj.pop('fred'))
42
>>> print(jjj)
{'jan': 100}
>>> print(jjj.pop('chuck',0))
0
>>> print(jjj)
{'jan': 100}
```

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Most Common Name?

marquard

zhen

csev

zhen

cwen

marquard

zhen

csev

cwen

zhen

csev

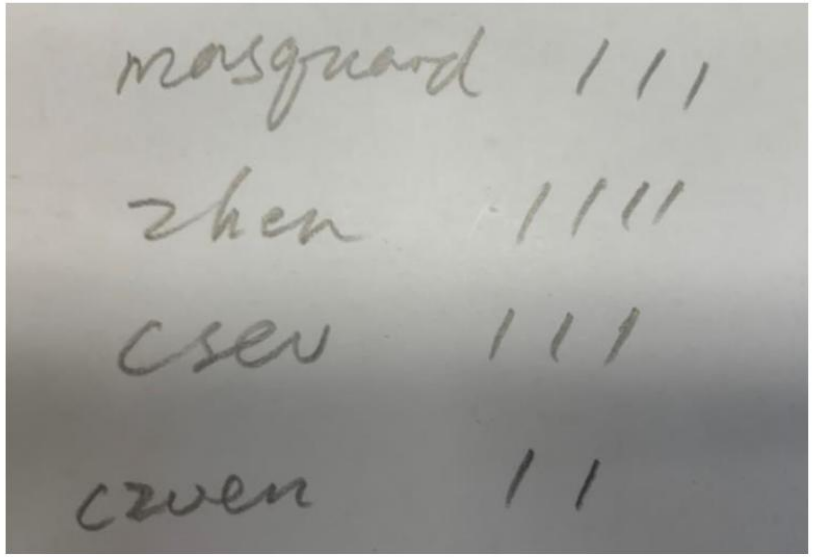
marquard

zhen

Many Counters with a Dictionary

- One common use of dictionaries is counting how often items appears.

```
>>> count = dict()
>>> count['csev'] = 1
>>> count['cwen'] = 1
>>> print(count)
{'csev': 1, 'cwen': 1}
>>> count['cwen'] = count['cwen'] + 1
>>> print(count)
{'csev': 1, 'cwen': 2}
```



Dictionary Tracebacks

- It is an error to refer a key not in the dictionary.
- We can use the in operator to see if a key exists.

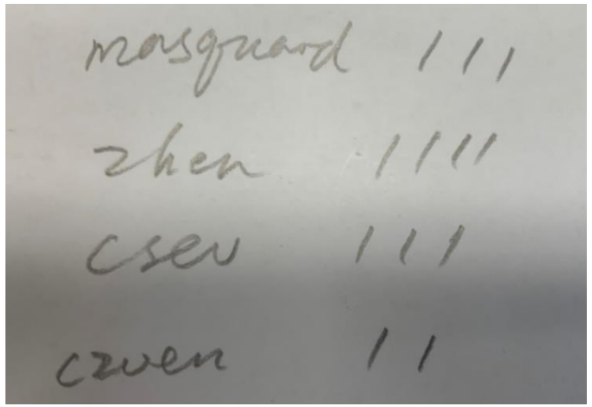
```
>>> count = dict()
>>> print(count['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in count
False
```

Using Loop for Counting

- New name needs to add new entry in the dictionary.
- For names already exists, simply add one to the count in the dictionary under that name.

{'csev': 2, 'zqian': 1, 'cwen': 2}

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print(counts)
```



The get Method for Dictionaries

- get() is for:
  - ➢Checking if a key is already in a dictionary.
  - ➢Assuming a default value if the key is not there.

```
>>>counts = {'csev': 2, 'zqian': 1, 'cwen': 2}
>>>print(counts['cwen'])
2
>>>print(counts['owen'])
KeyError: 'owen'
>>>print(counts.get('owen',0))
0
```

```
if name in counts:
    x = counts[name]
else :
    x = 0
```

```
x = counts.get(name, 0)
```

0 is the default value if key does not exist (to avoid errors in Traceback).

Simplified Counting with get()

- Use get() to provide a default value of zero when the key is not yet in the dictionary and then just add one.

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

Default

{'csev': 2, 'zqian': 1, 'cwen': 2}

Counting Words in Text

```
counts = dict()
line = input('Enter a line of text: ')
words = line.split()
print('Words: ', words) print('Counting...’)
for word in words:
        counts[word] =counts.get(word,0) + 1
print(‘Counts’,counts)
```

- Enter a line of text:

➢ the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car.

➢ Words: ['the', 'clown', 'ran', 'after', 'the', 'car', 'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and', 'the', 'tent', 'fell', 'down', 'on', 'the', 'clown', 'and', 'the', 'car']

➢ Counting...

➢ Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7, 'tent': 2}

Lowest Rates for Lanes

```
#use dictionary variable lowest_rates
file = open('rates-mlanes.txt', 'r')
lowest_rates = dict()

for str_line in file:
    #parse:
    line = str_line.split()
    if len(line)==0:
        break
    lane_id = line[0]
    carrier_id = line[1]
    rate = float(line[2])
    #update
    if (lane_id not in lowest_rates) or rate<lowest_rates[lane_id]:
        lowest_rates[lane_id] = rate
file.close()
for lane_id in lowest_rates:
    print('Lowest rate for lane %s is %g.'% \
          (lane_id, lowest_rates[lane_id]))
```

How to simplify this by using get() method of the dictionary?

lowest_rates[lane_id] = min(lowest_rates.get(lane_id,rate),rate)

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Definite Loops and Dictionaries

- Even though dictionaries are not stored in order, we can write a for loop that goes through all the entries in a dictionary by keys.
- Loop goes through all the keys in the dictionary and looks up the values.

```
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...     print(key, counts[key])
...
jan 100
chuck 1
fred 42
>>>
```

keys(), values(), and items()

- You can get a list of keys, values, or items (both) from a dictionary using methods keys(), values(), and items().

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

Two Iteration Variables!

- We can loop through the key-value pairs in a dictionary using **two** iteration variables
- Each iteration, the first variable is the key and the second variable is the corresponding value for the key

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

```
jan 100
chuck 1
fred 42
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

Find the Most Frequent Word in a File

```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
        words = line.split()
        for word in words:
                counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
        if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

Count words in a file by using two nested loops: a loop inside a loop

Find the most frequent words using two iteration variables

Enter file: words.txt
to 16

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Revisit for Tuples

- We have already seen tuples!

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

Tuples

- Tuple is a comma-separated list of values surrounded by round bracket: ('a', 'b', '3', 'd', 'e')

- To create a tuple with a single element, need to include final comma: t = ('a',)

➢If last comma not included, the value is not a tuple: ('a') is a string.

Tuples Are Like Lists

- Index starts at 0.
- Elements can be accessed by index or slices.
- Functions len, max, min, sum and operators in, +, * are applicable.

```
>>> x = ('Glenn', 'Sally', 'Joseph')
>>> print(x[2])
Joseph
>>> y = ( 1, 9, 2 )
>>> print(y)
(1, 9, 2)
>>> print(y[1:])
(9, 2)
>>> print(max(y))
9
```

```
>>> for iter in y:
...     print(iter)
...
1
9
2
>>>
```

Tuples are Immutable

- Unlike a list, once you create a tuple, you cannot alter its contents.

➢Similar to a string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>>[9, 8, 6]
>>>
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback:'str'
object does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback:'tuple'
object does
not support item
Assignment
>>>
```

Things not to do With Tuples

```
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:
AttributeError: 'tuple' object has no attribute 'reverse'
>>>
```

A Tale of Two Sequences

```
>>> l = [1,2,2]
>>> dir(l)
['append', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

```
>>> t = (1,2,2)
>>> dir(t)
['count', 'index’]
>>> print(t.count(2))
2
>>> print(t.index(2))
1
>>> print(t.index(2,2))
2
```

```
>>> help(().count)
Help on built-in function count:

count(value, /) method of builtins.tuple instance
    Return number of occurrences of value.
```

```
>>> help(().index)
Help on built-in function index:

index(value, start=0, stop=2147483647, /) method of builtins.tuple instance
    Return first index of value.

    Raises ValueError if the value is not present.
```

Tuples are More Efficient

- Tuple structures do not need to be modifiable: simpler and more efficient in terms of memory use than list.

```
%timeit list(range(100))
```

```
915 ns ± 57 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
%timeit tuple(range(100))
```

```
878 ns ± 30.5 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Tuples and Assignment

- Can put a tuple on the left-hand side of an assignment statement.

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> a, b = (99, 98)
>>> print(a)
99
```

```
>>> m = [ 'have', 'fun' ]
>>> (x, y) = m
>>> x
'have'
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
>>> print(uname)
monty
```

Tuples and Dictionaries

- items() in dictionaries returns a list of (key, value) tuples.

- Can be used in for loop:

➢for key, val in d.items()

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k,v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

Composite Keys

- To create a composite key to use in a dictionary, we must use a tuple as the key (not list).
- Create a telephone directory, mapping last name and first name to telephone numbers.

➢Key: (last-name, first-name), value: telephone number

```
directory = {('Xiaoyu', 'Wang'): 1234, ('Lei', 'Li'): 2234}
for first, last in directory:
    print(last, first, directory[first,last])
```

```
Wang Xiaoyu 1234
Li Lei 2234
```

directory[last,first] = number

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Sort Dictionary by Keys

- Dictionary can be sorted by keys using function sorted().

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> d.items()
dict_items([('a', 10), ('c', 22), ('b', 1)])
>>> sorted(d)
['a', 'b', 'c']
>>> sorted(d.items())
[('a', 10), ('b', 1), ('c', 22)]
```

- What is the difference between these two sorts?
- How to sort items of a dictionary by values?

Sort Dictionary by Values

- If we could construct a list of tuples of the form (value, key) we could sort by value.
- We do this with a for loop that creates a list of tuples.

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...     tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

Sort Dictionary by Values: A Shorter Version

```
>>> c = {'a':10, 'b':1, 'c':22}

>>> print(sorted(c.items(), key=lambda x:x[1]))

[(1, 'b'), (10, 'a'), (22, 'c')]
```

Use lambda function to customize the comparisons in sorted.

Sorted compares the second element of each tuple in c.items().

Sort Dictionary by Values: Shorterer Version

```
>>> c = {'a':10, 'b':1, 'c':22}

>>> print( sorted( [ (v,k) for k,v in c.items() ] ) )

[(1, 'b'), (10, 'a'), (22, 'c')]
```

Use list comprehension to creates a dynamic list.

In this case, we make a list of tuples and then sort it.

Dictionaries and Tuples

- Dictionaries
  - Concepts
  - Dictionaries for counting
  - Loops
- Tuples
  - Concepts
  - Comparing tuples
  - Sequencing

Strings, Lists, and Tuples are all Sequences

- Strings are more limited than other sequences because the elements must be characters.

➢Immutable.

➢To change characters, must create a new string.

“MMM” is in “MMMM OOCL”

“MMM” is NOT in [“MMMM”, “OOCL”]

Strings, Lists, and Tuples are all Sequences

- Lists are more common than tuples, because it is mutable.
- Tuple is preferred for composite key.

```
directory = {('Xiaoyu', 'Wang'): 1234, ('Lei', 'Li'): 2234}
for first, last in directory:
    print(last, first, directory[first,last])
```

```
Wang Xiaoyu 1234
Li Lei 2234
```

Strings, Lists, and Tuples are all Sequences

- Tuples are immutable. No sort and reverse, which modify existing lists.
- However, built-in functions sorted and reversed work for tuples.

```
t = (1,5,2,4,3)
print(tuple(reversed(t)))
print(tuple(sorted(t)))
```

```
(3, 4, 2, 5, 1)
(1, 2, 3, 4, 5)
```

Attention

- What is wrong?

```
>>> def f(s):
        print(s[0])


>>> f(1)
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    f(1)
  File "<pyshell#71>", line 2, in f
    print(s[0])
TypeError: 'int' object is not subscriptable
```

Acknowledgement

- Acknowledgements / Contributions

- Initial Development: Charles Severance, University of Michigan School of Information
- Further Development: Zhou Xu, Hong Kong Polytechnic University
- Continuous development: Xiaoyu Wang, Hong Kong Polytechnic University