Lecture 2—List and Collection

Version 1.0. Prepared by Makzan, 2020 June.

In last lecture, we learn to store values into variables. There were `int`, `float`, `string`, `boolean`. They are all singular value. In this notebook, we learn to store a collection of values by using `list`, `tuple`, `dictionary`, and `Named Tuple`.

Table of Content

List

We can create list by using `[ ]` brackets. Then we separate each item by comma. For example: the following create a list with 4 items.

In [1]:
```
[1,2,3,4]
```

Out[1]:
```
[1, 2, 3, 4]
```

We need a variable to store the list. Otherwise, we cannot reference the list anymore.

In [2]:
```
sample_list = [1,2,3,4]
```

Count and Sum the list

There are basic built-in list functions: `len`, `max`, `min`, `sum`.

In [3]:
```
len(sample_list)
```

Out[3]:
```
4
```

In [4]:
```
max(sample_list)
```

Out[4]:
```
4
```

In [5]:
```
min(sample_list)
```

Out[5]:
```
1
```

In [6]:
```
sum(sample_list)
```

Out[6]:
```
10
```

In [7]:
```
sum(sample_list)/len(sample_list)
```

Out[7]:
```
2.5
```

The following example create a list that stores 5 names.

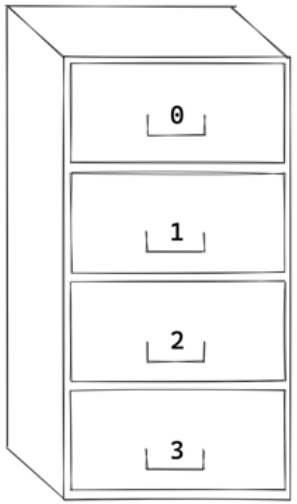
In [8]:
```
names = ['Thomas', 'Steven', 'Jane', 'Tom', 'Susan']
```

By using `len()`, we know that there are 5 items in the list.

In [9]:
```
len(names)
```

Out[9]:
```
5
```

Index of List



We can get individual item from the list by using `[index]`. The index begins with 0.

0	1	2	3	4
Thomas	Steven	Jane	Tom	Susan

-5	-4	-3	-2	-1
Thomas	Steven	Jane	Tom	Susan

In [10]:
```
names[0]
```

Out[10]:
```
'Thomas'
```

In [11]:
```
names[1]
```

Out[11]:
```
'Steven'
```

In [12]:
```
names[-1]
```

Out[12]:
```
'Susan'
```

In [13]:
```
names[-2]
```

Out[13]:
```
'Tom'
```

Appending items to list

We can append new item to the end of the list.

In [14]:
```
names.append("John")
```

In [15]:
```
names
```

Out[15]:
```
['Thomas', 'Steven', 'Jane', 'Tom', 'Susan', 'John']
```

We can also insert item into any position of the list. For example, we can insert item to the beginning by using index `0`.

In [16]:
```
names.insert(0, "Peter")
```

In [17]:
```
names
```

Out[17]:
```
['Peter', 'Thomas', 'Steven', 'Jane', 'Tom', 'Susan', 'John']
```

Sometimes, we will need to append multile items at once. We can do so by using `extend` function.

In [18]:
```
names.extend(['Victoria', 'Terence', 'Eric', 'Nicolas', 'Thomas'])
```

In [19]:
```
names
```

Out[19]:
```
['Peter',
 'Thomas',
 'Steven',
 'Jane',
 'Tom',
 'Susan',
 'John',
 'Victoria',
 'Terence',
 'Eric',
 'Nicolas',
 'Thomas']
```

If we need to remove an item, we can use `remove` and provide the item value.

In [20]:
```
names.remove('Thomas')
```

In [21]:
```
names
```

Out[21]:
```
['Peter',
 'Steven',
 'Jane',
 'Tom',
 'Susan',
 'John',
 'Victoria',
 'Terence',
 'Eric',
 'Nicolas',
 'Thomas']
```

If we know the index of the item, we can use `del` to delete the item by using index. Beware that `del` is not a function. There is no `( )` when using `del`.

In [22]:
```
del names[1]
```

In [23]:
```
names
```

Out[23]:
```
['Peter',
 'Jane',
 'Tom',
 'Susan',
 'John',
 'Victoria',
 'Terence',
 'Eric',
 'Nicolas',
 'Thomas']
```

Exercise: List creation, insert, and remove

Assuming now we want to store a list of cities. They are:

'Macao', 'Beijing', 'Helsinki', 'Kyoto', 'Sydney'

In [24]:
```
# Your code here
cities = None
cities
```

Expected result

['Macao', 'Beijing', 'Helsinki', 'Kyoto', 'Sydney']

Now we want to append city "Hong Kong" to last of the list. Please use `append()` to append the item.

In [25]:
```
# Your code here
None

cities
```

Expected result

['Macao', 'Beijing', 'Helsinki', 'Kyoto', 'Sydney', 'Hong Kong']

Now we want to prepend city "Shanghai" to the first of the list. Please use `insert()` to prepend the item.

In [26]:
```
# Your code here
None

cities
```

Expected result

['Shanghai', 'Macao', 'Beijing', 'Helsinki', 'Kyoto', 'Sydney', 'Hong Kong']

Now we want to append a list of cites to the current `cities` list. We can use `extend()` to do so. Given the following `new_cities` list.

In [27]:
```
new_cities = ['London', 'Stockholm', 'Sao Paulo']
```

In [28]:
```
# Your code here
None

cities
```

Expected result

['Shanghai', 'Macao', 'Beijing', 'Helsinki', 'Kyoto', 'Sydney', 'Hong Kong', 'London', 'Stockholm', 'Sao Paulo']

Slicing

Given a list **L**, we can extract individual item by using `L[index]` where index begins with 0. For example, `L[0]` to get the first item, `L[-1]` to get the last item.

We can also extract a range of items by using slicing. The syntax is `[ start : end : step ]`.

The range includes the `start` index and exclude the `end` index.

In [29]:
```
sample_list = [1,2,3,4,5,6,7,8,9,10]
```

In [30]:
```
sample_list[0:5]
```

Out[30]:
```
[1, 2, 3, 4, 5]
```

Item at index 1—3 (index begins with 0)

In [31]:
```
sample_list[1:4]
```

Out[31]:
```
[2, 3, 4]
```

Item at index 1, 3

In [32]:
```
sample_list[1:4:2]
```

Out[32]:
```
[2, 4]
```

We can omit 0 for the starting index. For example, this gets the first 5 items.

In [33]:
```
sample_list[:5]
```

Out[33]:
```
[1, 2, 3, 4, 5]
```

Items from beginning to last 5th item.

In [34]:
```
sample_list[:-5]
```

Out[34]:
```
[1, 2, 3, 4, 5]
```

We need to omit the end value to indicate until-the-end. For example, this gets the last 3 items.

In [35]:
```
sample_list[-3:]
```

Out[35]:
```
[8, 9, 10]
```

Every second item from beginning to end

In [36]:
```
sample_list[::2]
```

Out[36]:
```
[1, 3, 5, 7, 9]
```

A copy of the whole list

In [37]:
```
sample_list[:]
```

Out[37]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

A copy of the whole list, in reversed order

In [38]:
```
sample_list[::-1]
```

Out[38]:
```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Tuple

Tuple is the read-only version of list. We can express a tuple by using `( )`.

In [39]:
```
sample_tuple = (1,2,3,4,5,6,7,8,9,10)
```

We can read the Tuple as same as reading List. But we cannot modify the Tuple.

In [40]:
```
sample_tuple[0]
```

Out[40]:
```
1
```

Dictionary

Dictionary is a collection of key-value pairs. We can express a dictionary by using `{ }` with keys and values inside.

It is useful to collect information for the same subject together in one place. For example, we can store a student profile inside a dictionary. The benefit is that we can

In [41]:
```
sample_dict = {
    'name': 'Thomas',
    'score': 65
}
```

In [42]:
```
sample_dict
```

Out[42]:
```
{'name': 'Thomas', 'score': 65}
```

We can store dictionaries inside a list. Indeed, an "item" in collection can be another collection. So we can put dictionary inside list. Or we can put list inside dictionary.

In [43]:
```
students = [
    {'name': 'Thomas', 'score': 65},
    {'name': 'Alan', 'score': 95},
    {'name': 'Jane', 'score': 85},
    {'name': 'Susan', 'score': 75},
    {'name': 'Chris', 'score': 45}
]
```

Here is another example that put a list as value inside dictionary.

In [44]:
```
student_a = {
    'name': 'Steven',
    'email': 'steven@example.com',
    'classes': [
        'Introducing Python',
        'Web scraping with BeautifulSoup',
        'Plotting graph with MatPlotLib'
    ]
}
```

Exercise: Using Dictionary

Assuming now we want to store data of book. There are 3 attributes for each book. They are `title`, `category`, `price`.

Please create a `book` varibale to store the following book.

title	category	price
Python Tricks	Programming	240

In [45]:
```
book = None
book
```

Expected result

{'title': 'Python Tricks', 'category': 'Programming', 'price': 240}

Now we want to store a list of books. Here is the books data:

title	category	price
Python Tricks	Programming	240
Python Crash Course	Programming	200
Getting Real	Startup	200

In [46]:
```
book_a = None
book_b = None
book_c = None

books = [
    book_a,
    book_b,
    book_c
]

books
```

Out[46]:
```
[None, None, None]
```

Expected result

[{'title': 'Python Tricks', 'category': 'Programming', 'price': 240}, {'title': 'Python Crash Course', 'category': 'Programming', 'price': 200}, {'title': 'Getting Real', 'category': 'Startup', 'price': 200}] ||

Exercise: Slicing

Assuming now we have a list of numbers

In [47]:
```
sample_list = [1,2,3,4,5,6,7,8,9,10]
```

How to get the last item in the list?

In [48]:
```
sample_list
```

Out[48]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Expected result

10

How to get last 3 items in the list?

In [49]:
```
sample_list
```

Out[49]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Expected result
[8, 9, 10]

How to get the first 3 items in the list?

In [50]:
```
sample_list
```

Out[50]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Expected result
[1, 2, 3]

🤖 Boss Level

Now assume we have a list of books.

In [3]:
```
books = [
    {'title': 'Python Tricks', 'category': 'Programming', 'price': 240},
    {'title': 'Python Crash Course', 'category': 'Programming', 'price': 20
    {'title': 'Getting Real', 'category': 'Startup', 'price': 200}
]
```

How to get the first book title?

In [18]:
```
result_1 = None

result_1
```

Expected result
Python Tricks

How to get the price of last book?

In [19]:
```
result_2 = None

result_2
```

Expected result
200

In [20]:
```
def hello_boss(result_1, result_2):
    if result_1 == "Python Tricks" and result_2 == 200:
        print("Congratulations! You passed Lesson 2 Collection.")
    elif result_1 == "Python Tricks":
        print("Sorry! Your result 2 is incorrect.")
    elif result_2 == 200:
        print("Sorry! Your result 1 is incorrect.")
    else:
        print("Sorry! Your results are incorrect.")

hello_boss(result_1, result_2)
```

```
Sorry! Your results are incorrect.
```

Expected result
Congratulations! You passed Lesson 2 Collection.

Summary

In this notebook, we learned the essential techniques to store a series of data into list. Next, we will take a look at logic-flow with if-condition and for-loop.