Python Programming

Data Containers

Prof. Chang-Chieh Cheng
Information Technology Service Center
National Yang Ming Chiao Tung University

Lists

- A list is an ordered data sequence
- Random access
  - Each element can be accessed by an index
- Duplicate elements are allowed

```
L = [3, 3, 2, 2, 2, 4, 1]
print(L[0])
L[0] = L[1] + L[2]
print(L[0])
print(L)
```

Some useful methods of lists

- `list()`
  - create an empty list.

```
L = list()
print(L)        # []
```

- `list.append(x)`
  - add an item `x` to the end of the `list`

```
L = [5, 6, 7]
print(L)        # [5, 6, 7]
L.append(99)
print(L)        # [5, 6, 7, 99]
```

- `list.insert(i, x)`
  - Insert an item `x` at a given position `i`.
  - Then, `i` will be the index of `x`.

```
L = ['A', 'B', 'C']
L.insert(1, 'X')
print(L)        # ??
```

- `list.reverse()`
  - Reverse the elements of the list in place.

```
L = [5, 6, 7]
L.reverse()
print(L)        # [7, 6, 5]
```

Some useful methods of lists

- `list.clear()`
  - Remove all items.

```
L = [5, 6, 7]
print(L)        # [5, 6, 7]
L.clear()
print(L)        # []
```

- `list.remove(x)`
  - Remove the first item from the list whose value is `x`. It is an error if there is no such item.

```
L = [5, 6, 7, 6]
print(L)        # [5, 6, 7, 6]
L.remove(6)
print(L)        # [5, 7, 6]
L.remove(6)
print(L)        # [5, 7]
L.remove(6)     # An exception of ValueError thrown.
```

del

- Another way to remove an item from a list

```
L= [-1, 1, 66.25, 333, 333, 1234.5]
del(L[0])
print(L)        # [1, 66.25, 333, 333, 1234.5]
del(L[2:4])
print(L)        # [1, 66.25, 1234.5]
del(L[:])
print(L)        # []
```

Some useful methods of lists

- `list.sort(key=None, reverse=False)`
  - Sorting, where *`key`* specifies the comparison method. Just let it be *`None`* in most cases.

```
L = ['cat', 'mouse', 'pig', 'dog', 'bird']
L.sort()
print(L)        # ['bird', 'cat', 'dog', 'mouse', 'pig']
```

```
L = ['cat', 'mouse', 'pig', 'dog', 'bird']
L.sort(key = len)
print(L)        # ['cat', 'pig', 'dog', 'bird', 'mouse']
```

```
L = ['cat', 'mouse', 'pig', 'dog', 'bird']
L.sort(key = len, reverse = True)
print(L)        # ['mouse', 'bird', 'cat', 'pig', 'dog']
```

- Let's try it
  - Sort a list that contains a set of integers by the descending order of the number of digits
  - If any two numbers have the same digit number, their order in the original list must be kept.
  - For example,
    - If L = `[123, 4, 567, 9801, 1234, 0, 2341]`
    - The result is `[9801, 1234, 2341, 123, 567, 4, 0]`

Multidimensional arrays

- 3x4 array

```
L = [[1, 2, 3 ,4], [5, 6, 7, 8], [9, 10, 11, 12]]
print(L)
print(len(L))           # 3
print(len(L[0]))        # 4
print(L[0][0], L[0][1], L[1][0], L[2][3])
Print(L[2])
```

- 2 x 3 x 2 array

```
L = [[[1, 2], [3 ,4], [5, 6]],
     [[7, 8], [9, 10], [11, 12]]]
print(L)
print(len(L))           # 2
print(len(L[0]))        # 3
print(len(L[0][0]))     # 2
print(L[0][0][0], L[0][1][0], L[1][2][1])
print(L[1])
print(L[0][2])
```

Multidimensional arrays

- 1 x 2 x 3 x 2 array

```
L = [[[1, 2], [3 ,4], [5, 6]],
     [[7, 8], [9, 10], [11, 12]]]
print(L)
print(len(L))           # 1
print(len(L[0]))        # 2
print(len(L[0][0]))     # 3
print(len(L[0][0][0])) # 2
print(L[0][0][0][0], L[0][1][2][1])
```

Shallow copy & deep copy

- Shallow copy

```
L1 = [[1, 2], [3, 4], 5, 6]
L2 = list(L1)

L2[2] += 1
print(L1, L2)

L2[0][0] += 10
print(L1, L2)
```

Shallow copy & deep copy

- Deep copy

```
import copy
L1 = [[1, 2], [3, 4], 5, 6]
L2 = copy.deepcopy(L1)

L2[2] += 1
print(L1, L2)

L2[0][0] += 10
print(L1, L2)
```

List comprehension

- Using a for-statement to generate a list

```
L1 = [ x for x in range(5) ]
print(L1)       # [0, 1, 2, 3, 4]

L2 = [ x * 2 for x in range(5) ]
print(L2)       # [0, 2, 4, 6, 8]

L3 = [ x * x for x in range(5) if x % 2 == 0]
print(L3)       # [0, 4, 16]
```

List comprehension

- Multiple layers of for-statement and 2D list

```
L4 = [y for x in range(3) for y in range(4)]
print(L4)
# [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]

L5 = [[y for x in range(3)] for y in range(4)]
print(L5)
# [[0, 0, 0], [1, 1, 1], [2, 2, 2], [3, 3, 3]]

L6 = [[y * 3 + x for x in range(3)] for y in range(4)]
print(L6)
# [[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]]
```

Exercise 1

- Using list comprehension to generate a m * n list as follows
  - m = 6, n = 3:

```
[[0, 1, 2],
 [5, 4, 3],
 [6, 7, 8],
 [11, 10, 9],
 [12, 13, 14],
 [17, 16, 15]]
```

  - m = 8, n = 4:

```
[[0, 1, 2, 3],
 [7, 6, 5, 4],
 [8, 9, 10, 11],
 [15, 14, 13, 12],
 [16, 17, 18, 19],
 [23, 22, 21, 20],
 [24, 25, 26, 27],
 [31, 30, 29, 28]]
```

Tuples

- A tuple is also a data container to store a set of data objects
- Like list, using an index to access an item of a tuple

```
t = 12345, 54321, 'hello!'    # without any parenthesis
print(t[0])     # 12345
print(t)        # (12345, 54321, 'hello!')

u = t, (1, 2, 3, 4, 5)
                # A tuple also can be an item of another tuple
print(u)        # ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
                # Notice the parentheses
```

- Each item is immutable (read only).

```
t = 12345, 54321, 'hello!'
t[1] = 0        # Error!
```

Tuples

- Tuple appending

```
t = 1, 2, 3
t += 4          # Error!
t += (4)        # Error! Why?

t += (4,)       # OK!
```

Tuples

- Let's try it
  - The following code can draw a figure from two lists that are X axis's data and Y axis's data.

```
import matplotlib.pyplot as plt
X = [0, 1, 2, 3]
Y = [10, 5, 8, 9]
plt.plot(X, Y)
plt.scatter(X, Y)
plt.show()
```

  - Let's see what kind figure will be drawn if you changed some number in `X` and `Y`.

Tuples

- Let's try it
  - Given a set of tuple of two elements

```
T = (0, 10), (1, 5), (2, 8), (3, 9)
```

  - Write a program to transform `T` as two lists, `X` and `Y`, such that we can use the program in previous page to draw a figure for `T`.

Sets

- **Unordered** collection with **no duplicate** elements.

```
A = {7, 9, 1, 1, 9, 2, 1, 2}
print(A)        # {9, 2, 1, 7}
                # The order is undefined
                # because set is unordered
print(A[0])     # Error!
```

- Item appending and removing

```
A = {4, 6, 1, 2, 2, 1, 3}
A.add(5)
print(len(A))   # 6
print(A)        # {1, 2, 3, 4, 5, 6}
A.remove(2)
print(len(A))   # 5
print(A)        # {1, 3, 4, 5, 6}
A.clear()
print(len(A))   # 0
print(A)        # set()
```

Sets

- Element accessing
  - By for statement

```
A = {7, 9, 1, 1, 9, 2, 1, 2}
for x in A:
    print(x)
```

Sets

- Creating an empty set

```
A = set()
print(len(A))  # 0

A.add(7)
A.add(1)
A.add(1)
A.add(7)
A.add(9)
print(len(A))  # 3
print(A)       # {1, 9, 7}
```

Sets

- Let's try it
  - Use the following to input a series of positive numbers and store them into a list `L`.

```
S = set()
while True:
    x = int(input('Input a positive number: '))
    if x < 0:
        break
    else:
        S.add(x)
print('The size of S is ', len(S))
print(S)
L = list(S);
print(L)
```

Sets

- Let's try it
  - Given a list `L`.
  - Find the median of `L`
  - Using a set `S` to store all elements in `L`
  - Find the median of `S`
  - For example,
    - if `L = [4, 6, 1, 2, 2, 1, 3 ]`, its median is 2
    - Then, `S` maybe is `{3, 2, 1, 4, 6}`, the median of S is 3

    - if `L = [7, 8, 8, 6, 5, 2, 2, 3, 3 ]`, its median is 5
    - Then, `S` maybe is `{5, 3, 2, 7 ,6 ,8}`, the median of `S` is 6

Sets

- Because a set is unordered, we cannot sort a set
- But sorted function can return a list of sorted data from a set

```
A = {7, 9, 1, 1, 9, 2, 1, 2}
L = sorted(A)
print(A)        # {9, 2, 1, 7}
print(L)        # [1, 2, 7, 9]
```

Sets

- Union two sets

```
A = {4, 3, 1, 2}
B = {3, 6, 5, 4}
C = A.union(B)
print(A)        # {1, 2, 3, 4}
print(B)        # {3, 4, 5, 6}
print(C)        # {1, 2, 3, 4, 5, 6}
```

- Intersection of two sets

```
A = {4, 3, 1, 2}
B = {3, 6, 5, 4}
C = A.intersection(B)
print(A)        # {1, 2, 3, 4}
print(B)        # {3, 4, 5, 6}
print(C)        # {3, 4}
```

Sets

- Check whether two sets are disjointed

```
A = {4, 3, 1, 2}
B = {3, 6, 5, 4}
C = {5, 6}
print(A.isdisjoint(B))          # False
print(A.isdisjoint(C))          # True
```

- Check set B is a subset of set A

```
A = {4, 3, 1, 2}
B = {3, 6, 5, 4}
C = {5, 6}
print(C.issubset(A))            # False
print(C.issubset(B))            # True
print(B.issubset(C))            # False
```

Sets

- Check set B is superset of set A

```
A = {4, 3, 1, 2}
B = {3, 6, 5, 4}
C = {5, 6}
print(C.issuperset(A))          # False
print(C.issuperset(B))          # False
print(B.issuperset(C))          # True
```

- A - B

```
A = {4, 3, 1, 2}
B = {3, 6, 5, 4}
C = A.difference(B)
D = B.difference(A)
print(C)                  # {1, 2}
print(D)                  # {5, 6}
C = A.symmetric_difference(B)
D = B.symmetric_difference(A)
print(C)                  # {1, 2, 5, 6}
print(D)                  # {1, 2, 5, 6}
```

enumerate

- Creating a sequence of tuples for a data container and each tuple contains (index, and data).

```
L = ['ABC', 'DEF', 'GHI']
E = enumerate(L)
for x in E:
    print(x)

S = {'ABC', 'DEF', 'GHI'}
E = enumerate(S)
for x in E:
    print(x)
```

Dictionaries

- A dictionary is similar to a list, but each element is indexed by a key rather than an integer

```
Scores = {'James':82, 'Mary':98, 'Yamamoto':93}
print(Scores['Mary'])  # 98
Scores['Yamamoto'] += 7
print(Scores)          # {'James':82, 'Mary':98, 'Yamamoto':100}
print(len(Scores))     # 3
```

- Like a set, each element is unique in a dictionary.
  - Notice that all elements are unsorted

```
Scores = {'Yamamoto':93, 'James':82, 'Mary':68, 'Mary':98, 'James':80}
print(Scores)  # {'Yamamoto': 93, 'James': 80, 'Mary': 98}
```

Dictionaries

- Element updating and appending

```
Scores = {'James':82, 'Mary':98, 'Yamamoto':93}
print(Scores)  # {'James':80, 'Mary':98, 'Yamamoto':93}
Scores.update({'Yamamoto':84})
print(Scores['Yamamoto'])
Scores.update({'Hideo':77})
print(Scores)  # {'James':80, 'Mary':98, 'Yamamoto':93, 'Hideo':77}
```

- Element removing

```
Scores = {'James':82, 'Mary':98, 'Yamamoto':93}
print(Scores)  # {'James':80, 'Mary':98, 'Yamamoto':93}
Scores.pop('James')
print(Scores)  # {'Mary':98, 'Yamamoto':93}
```

Dictionaries

- Create a new dictionary

```
Scores = dict()

Scores.update({'Yamamoto':84})
Scores.update({'Hideo':77})
print(Scores)  # {'Yamamoto':93, 'Hideo':77}
```

```
Scores = {}

Scores.update({'Yamamoto':84})
Scores.update({'Hideo':77})
print(Scores)  # {'Yamamoto':93, 'Hideo':77}
```

Dictionaries

- DO NOT use a floating point number to be a key

```
D = {}
i = 0.0
while i <= 1.0:
    print(i)
    i += 0.1
    D[i] = i * 100

print(D[0.8]) # Key error!
```

- Why?
  - Storing a floating number may generate an error

```
x = 0.7
y = 0.1
z = x + y
print(x, y, z)
u = 0.9
v = -0.1
w = u + v
print(u, v, w)
if z == w:
    print("!")
```

Dictionaries

- for loop and dictionaries

```
Scores = {'James':82, 'Mary':98, 'Yamamoto':93}
for key in Scores:
    print(key )                          # list all keys

for key in Scores:
    print(key , "=", Scores[key])        # list keys and values
```

Dictionaries

- Sort by key

```
Scores = {'James':82, 'Mary':98, 'Yamamoto':93}

L1 = sorted(Scores)
print(L1)
 # ['James', 'Mary', 'Yamamoto']

L2 = sorted(Scores.items())
print(L2)
 # [('James', 82), ('Mary', 98), ('Yamamoto', 93)]
```

Dictionaries

- Sort by value

```
from operator import itemgetter

Scores = {'James':82, 'Mary':98, 'Yamamoto':93}
L = sorted(Scores.items(), key = itemgetter(1))
print(L)
 # [('James', 82), ('Yamamoto', 93), ('Mary', 98)]
```

- **itemgetter is a function generator**

Dictionaries

- Let's try it
  - Modify all examples of dictionary such that each student can store **three** scores

Dictionaries

- A dictionary with multiple keys
- Example:

```
student1 = {'Name':'James', 'ID':'01008', 'Score':90}
student2 = {'Name':'Mary', 'ID':'01003', 'Score':98}
student3 = {'Name':'Yamamoto', 'ID':'01005', 'Score':93}
print(student1)
print(student2)
print(student3)
```

```
L = list()
L.append({'Name':'James', 'ID':'01008', 'Score':90})
L.append({'Name':'Mary', 'ID':'01003', 'Score':98})
L.append({'Name':'Yamamoto', 'ID':'01005', 'Score':93})
for student in L:
    print(student)
```

Dictionaries

- Data selection from a list of dictionaries
- Example:

```
L = list()
L.append({'Name':'James', 'ID':'01008', 'Score':90})
L.append({'Name':'Ruby', 'ID':'01024', 'Score':89})
L.append({'Name':'Mary', 'ID':'01003', 'Score':98})
L.append({'Name':'Yamamoto', 'ID':'01005', 'Score':93})
L.append({'Name':'Judy', 'ID':'01021', 'Score':73})

L2 = [x for x in L if x['Score'] < 90 ]

for student in L2:
    print(student)

L3 = [x for x in L if x['Name'][-1] == 'y' and x['Score'] >= 80 ]

for student in L3:
    print(student)
```

Dictionaries

- Data selection from a list of dictionaries
- Example:

```
from operator import itemgetter
L = list()
L.append({'Name':'James', 'ID':'01008', 'Score':90})
L.append({'Name':'Ruby', 'ID':'01024', 'Score':89})
L.append({'Name':'Mary', 'ID':'01003', 'Score':98})
L.append({'Name':'Yamamoto', 'ID':'01005', 'Score':93})
L.append({'Name':'Judy', 'ID':'01021', 'Score':73})

L4 = [{'ID':x['ID'], 'Score':x['Score']} for x in L ]
L4.sort(key = itemgetter('ID'))
for student in L4:
    print(student)
```

Strings

- All string operations
  - https://docs.python.org/3/library/stdtypes.html#string-methods
- Substring finding

```
s = 'Hello! My firends!'
if 'My' in s:
    print('OK')

if 'my' in s:
    print('OK')
```

Strings

- `str.index(substring, start=0, end=len(string))`
  - Get the lowest index of *substring* in *str*

```
s = 'Hello! My firends!'
print(s.index('My'))
print(s.index('my')) # Error!
print(s.index('!'))
print(s.index('!', s.index('!') + 1))
```

Strings

- `str.`**`split`**`(`*`sep=None, maxsplit=-1`*`)`
  - Return a list of the words in the string, using `sep` as the delimiter string. If `maxsplit` is given, at most `maxsplit` splits are done.

```
s = 'aaa*bbb*ccc eee fff*ggg '
L1 = s.split(sep = '*')
L2 = s.split(sep = ' ')
L3 = s.split(sep = '*', maxsplit = 2)
print(L1) # ['aaa', 'bbb', 'ccc eee fff', 'ggg ']
print(L2) # ['aaa*bbb*ccc', 'eee', 'fff*ggg', '']
print(L3) # ['aaa', 'bbb', 'ccc eee fff*ggg ']
```

  - Notice zero-length substrings

```
s = 'aaa*bbb**ccc***eee'
L = s.split(sep = '*')
print(L) # ['aaa', 'bbb', '', 'ccc', '', '', 'eee']
```

Strings

- `re.split`
  - Splitting a string with multiple separators

```
import re
s = 'aaa*bbb*ccc eee fff*ggg '
L4 = re.split('ccc|\*| ', s)
print(L4)
```

Exercise

- Input two strings, `s1` and `s2`.
- Using a `set` to store all words in s1.
- Count how many number of words in both `s1` and `s2`.
- For example, `s1` is `'How do you do who are you'` and `s2` is `'What do you think how are you'`
  - The result is 2, which are `'do' * 1`, `'are' * 1`, and `'you' * 2`.

Strings

- `str.`**`replace`**`(`*`old`*`, `*`new`*`[, `*`count`*`])`
  - Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
s = 'AAA ABC ABC ccc abc ABC ABC ddd'
s1 = s.replace('ABC', '*')
s2 = s.replace('ABC', '*', 3)
print(s1)       # AAA * * ccc abc * * ddd
print(s2)       # AAA * * ccc abc * ABC ddd
```

Strings

- `str.`**`count`**`(`*`sub[, start[, end]]`*`)`
  - Return the number of non-overlapping occurrences of substring *`sub`* in the range [*`start`*, *`end`*].

```
s = 'aaa bbb aaa aaa bbb ccc'
print(s.count('aaa'))          # 3
print(s.count('bbb'))          # 2
print(s.count('ccc'))          # 1
print(s.count('ddd'))          # 0
```

collections::Counter

- `collections` is a standard library of Python that includes a lot of many useful data containers
- `collections/Counter`
  - `Counter` is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values.

```
from collections import Counter
s = 'cccbbbaaabbbaaabbb'
cnt = Counter(s)
print('----------')
print(cnt)              # Counter({'b': 9, 'a': 6, 'c': 3})
print('----------')
print(cnt.keys())       # dict_keys(['c', 'b', 'a'])
print('----------')
print(cnt.items())      # dict_items([('c', 3), ('b', 9), ('a', 6)])
```

collections::Counter

- Accessing elements of `Counter`

```
from collections import Counter
s = 'cccbbbaaabbbaaabbb'
cnt = Counter(s)
for item in cnt:
    print(item, '\t', cnt[item])

"""
c         3
b         9
a         6
"""
```

without sorting!

Exercise 2

- Design a program to count the number of occurrences of each word in a text without any iterative statement or expression.
  - Assuming that each word is separated by several space characters.
  - Only count non-zero-length words
    - Just ignore zero-length words
  - Print the results by the number of occurrences in descending order
  - For example, given a text,

    `"  XYZ abc XYZ abc  xyz  abc  ABC  xyz  ",`

    the results will be

    `[('abc', 3), ('XYZ', 2), ('xyz', 2), ('ABC', 1)]`

  - Another example,

    `"  xyz abc  xyz abc  XYZ  abc  ABC  XYZ  ",`

    the results will be

    `[('abc', 3), ('xyz', 2), ('XYZ', 2), ('ABC', 1)]`