PROGRAMMING FOR BUSINESS COMPUTING
商管程式設計

Module 2-3: Data Structures

Hsin-Min Lu
盧信銘
台大資管系

Data Structures in Python

- A data structure is a particular way of organizing data in a computer so that it can be used efficiently.
- Python has many built-in data structures
  - list (discussed before)
  - tuple
  - dictionary
  - set
  - datetime (to handle date and time data)
  - … and more.
- We are going to cover selected data structures that are important for you.

TUPLES

Tuples

- Same as lists, but
  - Immutable
  - Enclosed in parentheses
  - A tuple with a single element ***must*** have a comma inside the parentheses: **a = (11,)**

```
>>> mytuple = (11, 22, 33)
>>> mytuple[0]
11
>>> mytuple[-1]
33
>>> mytuple[0:1]
(11,)
#The comma is required!
```

Why?

- It is clear that **[11]** and **11** are different (list of one element and integer 11)

- But,
- **(11)** is an acceptable expression
  - **(11) without the comma** is the integer 11
  - **(11,) with the comma** is a tuple containing the integer 11
- A small (but critical) piece of info that you need to know.

Tuples are immutable

- `>>> mytuple = (11, 22, 33)`
- `>>> saved = mytuple`
- `>>> mytuple += (44,)`
- `>>> mytuple`
  `(11, 22, 33, 44)`
- `>>> saved`
  `(11, 22, 33)`

Things that do not work

- ```
  mytuple += 55
  Traceback (most recent call last):Z
  …
  TypeError:
   can only concatenate tuple (not "int") to
  tuple
  ```

- Be aware of this ….


  ```

Sorting tuples

```
>>> atuple = (33, 22, 11)
>>> atuple.sort()
Traceback (most recent call last):
…
AttributeError:
'tuple' object has no attribute 'sort'
>>> atuple = sorted(atuple)
>>> atuple
[11, 22, 33]
```

Tuples are immutable!

sorted( ) returns a list!

Tuple and List Share Similar Operations

```
>>> atuple = (11, 22, 33)
>>> len(atuple)
3
>>> 44 in atuple
False
```

Converting Lists into Tuples

```
>>> alist = [11, 22, 33]
>>> atuple = tuple(alist)
>>> atuple
(11, 22, 33)
>>> newtuple = tuple('Hello World!')
>>> newtuple
('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r',
'l', 'd', '!')
```

Example: Taiwan ID Checksum

- Recall the process of computing Taiwan ID checksum.
- Convert the first letter to a two-digit number
- Apply weight to all 11 digits.
- Sum over all digits.

```
def cksum_twid(idstr):
    """Compute Checksum for Taiwn ID"""
    code1 = ord(idstr[0])
    #convert first English character to two-digit
number.
    cmap = [10, 11, 12, 13, 14, 15, 16, 17, 34, 18,
19, 20, 21, 22, 35, 23, 24, 25, 26, 27, 28, 29, 32,
30, 31, 33]
    num1 = cmap[code1 - 65]
    newid = str(num1) + idstr[1:]
    weight = [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
    checksum = 0
    for i in range(0, 11):
        checksum += weight[i] * int(newid[i])
    print("checksum=%d" % checksum)

id1 = "A123456789"
cksum_twid(id1)
```

- Output: checksum=130

Zipping two Variables

- zip: Make an iterator that aggregates elements from each of the iterables.

```
newid = '10123456789'
weight = [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]

for apair in zip(newid, weight):
    print(apair)
```

- Output:

```
('1', 1)
('0', 9)
('1', 8)
('2', 7)
('3', 6)
('4', 5)
('5', 4)
('6', 3)
('7', 2)
('8', 1)
('9', 1)
```

New Version Using “zip()”

```
def cksum_twid_v2(idstr):
    """Compute Checksum for Taiwn ID"""
    code1 = ord(idstr[0])
    #convert first English character to two-digit number.
    cmap = [10, 11, 12, 13, 14, 15, 16, 17, 34, 18, 19, 20,
21, 22, 35, 23, 24, 25, 26, 27, 28, 29, 32, 30, 31, 33]
    num1 = cmap[code1 - 65]
    newid = str(num1) + idstr[1:]
    weight = [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
    checksum = 0
    for apair in zip(newid, weight):
        checksum += apair[1] * int(apair[0])
    print("checksum=%d" % checksum)

#running the function
id1 = "A123456789"
cksum_twid_v2(id1)
```

- Output: checksum=130

The Lambda Operator

- Lambda is a way to define simple functions.

```
>>> def f1 (x):
...     return x**2
...
>>> print (f1(8))
64
>>>
>>> f2 = lambda x: x**2
>>> print (f2(8))
64
```

The Map Operator

- map() provides an easy way to apply an function to a list or tuples.
- Consider the situation when we want to square all numbers in a list.

```
>>> list1=[3,5,1.2, 4, 9]
>>> out1=map(f1, list1)
>>> print(list(out1))
[9, 25, 1.44, 16, 81]
>>>
>>> #using lambda
>>> out2=map(lambda x: x**2, list1)
>>> print(list(out2))
[9, 25, 1.44, 16, 81]
```

Checksum Using Map and Lambda

```
def cksum_twid_v3(idstr):
    """Compute Checksum for Taiwn ID"""
    code1 = ord(idstr[0])
    #convert first English character to two-digit number.
    cmap = [10, 11, 12, 13, 14, 15, 16, 17, 34, 18, 19, 20,
21, 22, 35, 23, 24, 25, 26, 27, 28, 29, 32, 30, 31, 33]
    num1 = cmap[code1 - 65]
    newid = str(num1) + idstr[1:]
    weight = [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
    out1 = map(lambda apair: apair[1] * int(apair[0]),
               zip(newid, weight))
    checksum=sum(out1)
    print("checksum=%d" % checksum)

id1 = "A123456789"
cksum_twid_v3(id1)
```

- Output: checksum=130

DICTIONARY

The *dictionary* data structure

- In Python, a *dictionary* is mapping between a set of indices (**keys**) and a set of **values**
  - The items in a dictionary are key-value pairs
- Keys can be any Python data type
  - Because keys are used for indexing, they should be immutable
- Values can be any Python data type
  - Values can be mutable or immutable

Creating a Dictionary

```
>>> eng2cn = dict()
>>> print(eng2cn)
{}
>>>
>>> eng2cn['one'] = '一'
>>> eng2cn['two'] = '二'
>>> eng2cn['three'] = '三'
>>> eng2cn['four'] = '四'
>>> print(eng2cn)
{'one': '一', 'two': '二', 'three': '三', 'four': '四'}
```

Creating a dictionary

- `>>> eng2cn = {'two': '二', 'three': '三', 'four': '四', 'one': '一'}`
- `>>> print(eng2cn)`
- `{'two': '二', 'three': '三', 'four': '四', 'one': '一'}`

- In general, the order of items in a dictionary is unpredictable
- Dictionaries are indexed by keys (including integers).

Dictionary indexing

```
>>> print(eng2cn['one'])
一
>>> print(eng2cn['two'])
二
>>> print(eng2cn['five'])
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'five'
```

* If the index is not a key in the dictionary, Python raises an exception ☼

Dictionary indexing

```
if 'five' in eng2cn:
    print(eng2cn['five'])
#no output
```

```
>>> print(eng2cn.get('five'))
None
```

The `in` operator

- Note that the `in` operator works differently for dictionaries than for other sequences
  - For strings, lists, and tuples, `x in y` means whether `x` is an item in the sequence
  - For dictionaries, `x in y` checks whether `x` is a key in the dictionary ☼

Keys and values

- The `keys` method returns a list of the keys in a dictionary
- The `values` method returns a list of the values

```
>>> print(eng2cn.keys())
dict_keys(['two', 'three', 'four', 'one'])
>>> print(eng2cn.values())
dict_values(['二', '三', '四', '一'])
```

Keys and values

- The `items` method returns a list of tuple pairs of the key-value pairs in a dictionary

```
>>> print(eng2cn.items())
dict_items([('two', '二'), ('three', '
三'), ('four', '四'), ('one', '一')])
```

Example from Section 11.2, Think Python Ver 2.2.17

Example:

```
def histogram(seq):
    d = dict()
    for element in seq:
        if element not in d:
            d[element] = 1
        else:
            d[element] += 1
    return d

h = histogram('brontosaurus')
print(h)
```

- Output:

```
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1}
```

Example from Chapter 11, Think Python Ver 2.2.17

Example:

Another way to output results:

```
def print_hist(hist):
    for key in hist:
        print(key, hist[key])

h = histogram('brontosaurus')
print_hist(h)
```

Output:

```
a 1
b 1
o 2
n 1
s 2
r 2
u 2
t 1
```

Example from Chapter 11, Think Python Ver 2.2.17

Example:

Change the print_hist function:

```
def print_hist2(hist):
    for key, value in hist.items():
        print (key, value)

h = histogram('brontosaurus')
print_hist2(h)
```

Output:

```
a 1
b 1
o 2
n 1
s 2
r 2
u 2
t 1
```

Example from Chapter 11, Think Python Ver 2.2.17

Sorting the keys

Change the print_hist function:

```
def print_hist3(hist):
    keys = hist.keys()
    for key in sorted(keys):
        print (key, hist[key])

h = histogram('brontosaurus')
print_hist3(h)
```

Output:

```
a 1
b 1
n 1
o 2
r 2
s 2
t 1
u 2
```

Using lists as values

- Inverting the mapping: What are the letters with a given count?

```
def invert_dict(d):
    inv = dict()
    for key in d:
        val = d[key]
        if val not in inv:
            inv[val] = [key]
        else:
            inv[val].append(key)
    return inv
```

Example from Chapter 11,
Think Python Ver 2.2.17

Inverting the Mapping: Example

```
def invert_dict(d):
    inv = dict()
    for key in d:
        val = d[key]
        if val not in inv:
            inv[val] = [key]
        else:
            inv[val].append(key)
    return inv

hist = histogram('parrot')
print (hist)
inverted = invert_dict(hist)
print (inverted)
```

Output:

```
{'a': 1, 'p': 1, 'r': 2, 't': 1, 'o': 1}
{1: ['a', 'p', 't', 'o'], 2: ['r']}
```

Example from Chapter 11, Think Python Ver 2.2.17

SETS

Sets

- Identified by ***curly braces***
  - **{'Marry', 'Bob', 'John'}**
  - **{'Dean'}** is a ***singleton***
- Sets can only contain ***unique elements***
  - ***Duplicates are eliminated***
- ***Immutable*** like tuples and strings ☼

```
>>> cset = {11, 11, 22}
>>> cset
{11, 22}
```

☼

Sets are Immutable

```
>>> aset = {11, 22, 33}
>>> bset = aset
>>> #union of two sets
>>> aset = aset | {55}
>>>
>>> aset
{33, 11, 22, 55}
>>> bset
{33, 11, 22}
```

Sets have no Order

- `>>> {1, 2, 3, 4, 5, 6, 7}`
- `{1, 2, 3, 4, 5, 6, 7}`
- `>>> {11, 22, 33}`
- `{33, 11, 22}`

Sets do not Support Indexing

```
>>> myset = {'大象', '長頸鹿', '蝸牛'}
>>> myset
{'蝸牛', '長頸鹿', '大象'}
>>> myset[0]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Examples

```
>>> alist = ['大象', '長頸鹿', '蝸牛', '大象', '猴子']
>>> aset = set(alist)
>>> aset
{'猴子', '蝸牛', '長頸鹿', '大象'}
>>> #set does not support + operation
>>> aset = aset + {'蟒蛇'}
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'set'
and 'set'
```

Boolean Operations on sets

```
>>> aset = {11, 22, 33}
>>> bset = {12, 23, 33}
```

Union of two sets

```
>>> aset | bset
{33, 22, 23, 11, 12}
```

Intersection of two sets:

```
>>> aset & bset
{33}
```

Boolean Operations on sets

```
>>> aset = {11, 22, 33}
>>> bset = {12, 23, 33}
```

Difference:

```
>>> aset - bset
{11, 22}
```

Symmetric difference:

```
>>> aset ^ bset
{11, 12, 22, 23}
```

☼

Contains all elements that are **either**
in set **A** but **not in** set **B** **or**
in set **B** but **not in** set **A**

DATETIME

Handling Date and Time in Python

- Python has a build-in library “datetime” that can process date and time data.
  - Need to do “import datetime” first

```
>>> import datetime
>>> #create datetime by year, month, day
>>> d1=datetime.datetime(2005,5,3)
>>> d1
datetime.datetime(2005, 5, 3, 0, 0)
>>> print(d1)
2005-05-03 00:00:00
```

The datetime Object

```
>>> #create datetime by
>>> #   year, month, day, hour, minute, second
>>> d2=datetime.datetime(2017, 2, 5, 8, 5, 20)
>>> d2
datetime.datetime(2017, 2, 5, 8, 5, 20)
>>> print(d2)
2017-02-05 08:05:20
>>> #extract the date components
>>> d2.date()
datetime.date(2017, 2, 5)
>>> #extract the time component
>>> d2.time()
datetime.time(8, 5, 20)
```

Getting Date-of-Week, and Today's Date

```
>>> #get day-of-week
>>> ##Monday is 0 and Sunday is 6
>>> d2.date().weekday()
6

>>> #return today's date
>>> datetime.date.today()
datetime.date(2017, 8, 22)
```

Getting the Value of Each Slot

```
>>> #get value  of each slot
>>> d2.year
2017
>>> d2.month
2
>>> d2.day
5
>>> d2.hour
8
>>> d2.minute
5
>>> d2.second
20
```

Difference of Datetime

```
>>> d3=datetime.datetime(1998, 2, 5, 8, 5, 20)
>>> d4=datetime.datetime(1999, 2, 1, 22, 4, 15)
>>> diff = d4 - d3
>>> #difference in days + seconds
>>> diff
datetime.timedelta(361, 50335)
>>> print(diff)
361 days, 13:58:55
>>> #get individual slots
>>> diff.days
361
>>> diff.seconds
50335
```

Time Shifting by timedelta

```
>>> diff2 = datetime.timedelta(days=3,seconds=4)
>>> d5 = datetime.datetime(2000,1,1,0,0,0)
>>> d6 = d5 + diff2
>>> print(d6)
2000-01-04 00:00:04
```

Datetime ←→String

```
>>> #datetime to string
>>> d7 = datetime.datetime(2002,5,2,13,15,45)
>>> print(str(d7))
2002-05-02 13:15:45
>>> print(d7.strftime('%Y-%m-%d'))
2002-05-02
>>> print(d7.strftime('%B %d, %Y'))
May 02, 2002
>>> print(d7.strftime('%Y-%m-%d %H:%M:%S'))
2002-05-02 13:15:45
>>> print(d7.strftime('%Y-%m-%d %I:%M:%S %p, %A'))
2002-05-02 01:15:45 PM, Thursday
>>>
```

Datetime ←→String

```
>>> #string to datetime.
>>> dstr = "2007-03-04 21:08:12"
>>> d9 = datetime.datetime.strptime(dstr, "%Y-%m-%d %H:%M:%S")
>>> d9
datetime.datetime(2007, 3, 4, 21, 8, 12)
```

- Full document here: https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior

THANK YOU!

Questions?