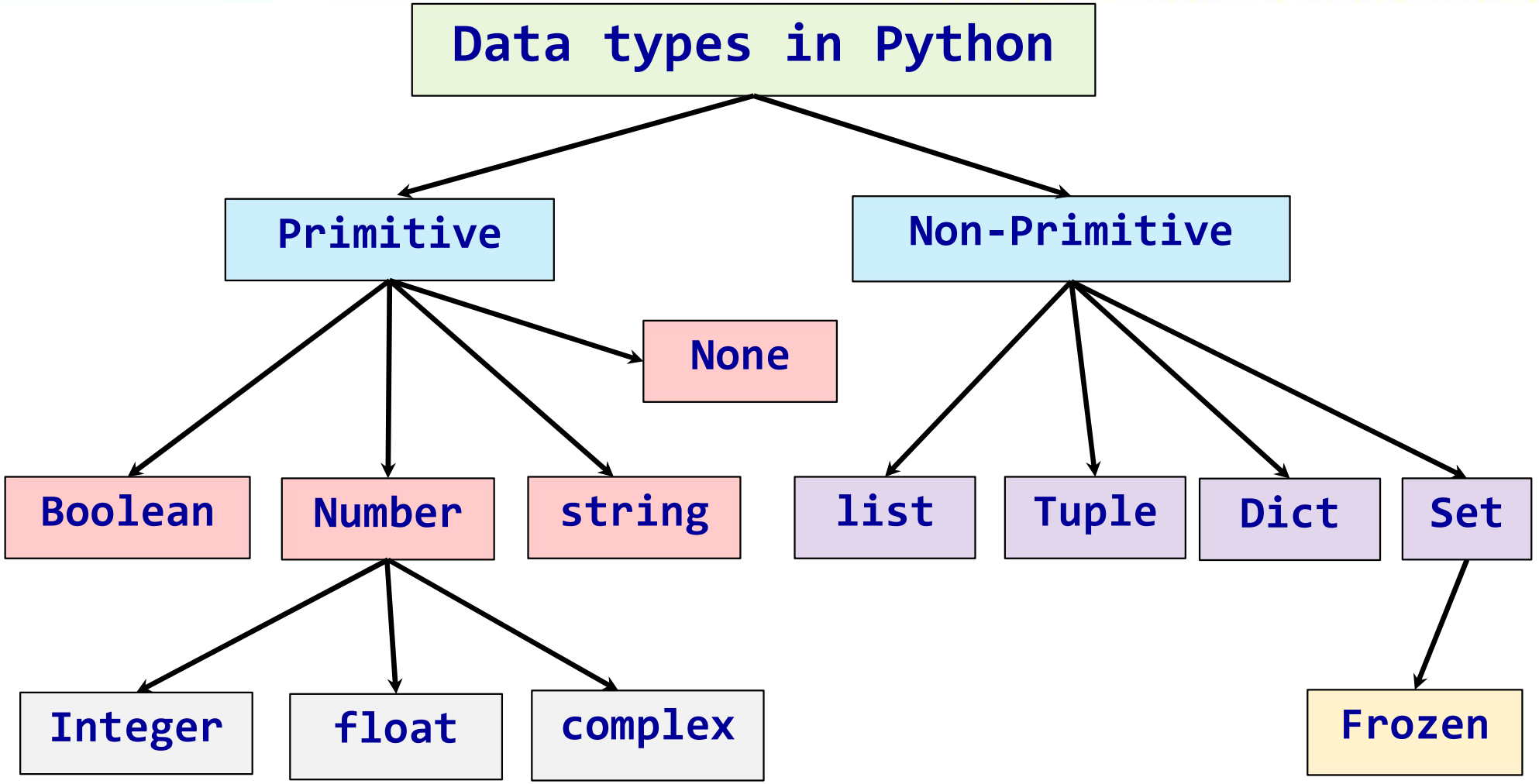
บทที่ 9 Dictionary and Tuple

สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Python Data types



What is a Collection?

A collection is nice because we can put more than one value in it and carry them all around in one convenient package

We have a bunch of values in a single “variable”

We do this by having more than one place “in” the variable

We have ways of finding the different places in the variable

What is Not a "Collection"?

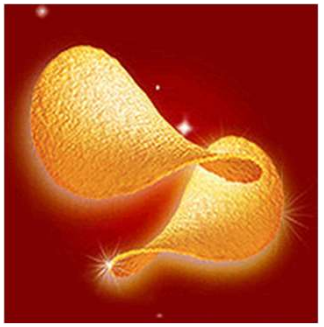
Most of our variables have one value in them - when we put a new value in the variable - the old value is overwritten

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

A Story of Two Collections..

- List

  - A linear collection of values that stay in order



- Dictionary

  - A “bag” of values, each with its own label

Dictionaries

Dictionaries are Python's most powerful data collection

Dictionaries allow us to do fast database-like operations in Python

Dictionaries have different names in different languages

- Associative Arrays - Perl / PHP
- Properties or Map or HashMap - Java
- Property Bag - C# / .Net

Dictionaries

- Lists index their entries based on the position in the list
- Dictionaries are like bags - no order
- So we index the things we put in the dictionary with a "lookup tag"

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

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

List

Key	Value
[0]	21
[1]	183

lst

Dictionary

Key	Value
['course']	182
['age']	21

ddd

Accessing unavailable key


```
d = {}
d['apple']=25
d['mango']=70
d['papaya']=37
print(f"d = {d}")
print(f"d['apple'] = {d['apple']}")
print(f"d['mango'] = {d['mango']}")
print(f"d['papaya'] = {d['papaya']}")
print(f"d['orange'] = {d['orange']}")
```

```
d = {'apple': 25, 'mango': 70, 'papaya': 37}
d['apple'] = 25
d['mango'] = 70
d['papaya'] = 37
Traceback (most recent call last):
  File "d:\22s2\22s2 Compro python\Lecture_22s2py\dict\ch09-01.py",
line 9, in <module>
    print(f"d['orange'] = {d['orange']}")
KeyError: 'orange'
```

Deleting item


```
d = {}
d['apple'] = 25
d['mango'] = 70
d['papaya'] = 37
print(f"d = {d}")
del d['apple']
print(f"d = {d}")
del d['mango']
print(f"d = {d}")
del d['papaya']
print(f"d = {d}")
```

```
d = {'apple': 25, 'mango': 70, 'papaya': 37}
d = {'mango': 70, 'papaya': 37}
d = {'papaya': 37}
d = {}
```

Counting fruit


```
fruits = ['apple','papaya','apple','papaya','mango']
d = dict()
for fruit in fruits:
    if fruit in d:
        d[fruit] += 1
    else:
        d[fruit] = 1
print(d)
```

```
{'apple': 2, 'papaya': 2, 'mango': 1}
```

Dictionary Methods

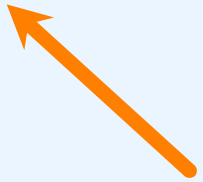

```
>>> d = dict()
>>> print(dir(d) )
['__class__', '__class_getitem__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__ior__', '__iter__', '__le__',
'__len__', '__lt__', '__ne__', '__new__', '__or__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__ror__',
'__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items',
'keys', 'pop', 'popitem', 'setdefault', 'update', 'values’]

>>> dMethods =[ele for ele in dir(d) if ele[0:2] != "__"]

>>> print([ele for ele in dir(d) if ele[0:2] != "__"])
['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop',
'popitem', 'setdefault', 'update', 'values']
```

Counting fruit


```
fruits = ['apple','papaya','apple','papaya','mango']
d = dict()
for fruit in fruits:
    d[fruit] = d.get(fruit, 0)+1
print(d)
```



Default

```
{'apple': 2, 'papaya': 2, 'mango': 1}
```

Counting Pattern


```
counts = dict()
print('Enter a line of text:')
line = input('')

words = line.split()

print('Words:', words)

print('Counting...')
for word in words:
    counts[word] = counts.get(word,0) + 1
print('Counts', counts)
```

The general pattern to count the words in a line of text is to split the line into words, then loop through the words and use a dictionary to track the count of each word independently.

```
python wordcount.py
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}
```

http://www.flickr.com/photos/71502646@N00/2526007974/

```
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)
```

```
python wordcount.py
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}
```

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 - actually it goes through all of 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
>>>
```

Retrieving Lists of Keys and Values


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

What is a “tuple”? - coming soon...

Bonus: Two Iteration Variables!


```
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

Tuples Are Like Lists

Tuples are another kind of sequence that functions much like a list - they have elements which are indexed starting at 0

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

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

but... Tuples are “immutable”

Unlike a list, once you create a tuple, you cannot alter its contents - similar to a string

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

```
>>> y = 'ABC’
>>> print(y[2])
C
>>> y[2] = 'D'
Traceback:'str'
object does
not support item
Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> print(z[2])
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'
>>>
```

Method of list and tuple


```
>>> l = list()
>>> print([ele for ele in dir(l) if ele[0:2] != "__"])
['append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']

>>> t = tuple()
>>> print([ele for ele in dir(t) if ele[0:2] != "__"])
['count', 'index']
```

Tuples are More Efficient

Since Python does not have to build tuple structures to be modifiable, they are simpler and more efficient in terms of memory use and performance than lists

So in our program when we are making “temporary variables” we prefer tuples over lists

Tuples and Assignment

We can also put a tuple on the left-hand side of an assignment statement

We can even omit the parentheses

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
>>> (x,y) = [13,17]
>>> print(x,y)
13 17
```

```
>>> x, y = 4, 'fred'
>>> print(y)
fred
>>> a, b = 99, 98
>>> print(a)
99
>>> x, y = [13,17]
>>> print(x,y)
13 17
```

Tuples and Dictionaries

The items() method in dictionaries returns a list of (key, value) tuples

```
>>> 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)])
```

Tuples are Comparable

The comparison operators work with tuples and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ( 'Jones', 'Sally' ) < ('Jones', 'Sam')
True
>>> ( 'Jones', 'Sally') > ('Adams', 'Sam')
True
```

Sorting Lists of Tuples

We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary

First we sort the dictionary by the key using the items() method and sorted() function

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

Using sorted()

We can do this even more directly using the built-in function sorted that takes a sequence as a parameter and returns a sorted sequence

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

Sort by Values Instead of Key

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')]
```

Even Shorter Version


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

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

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

List comprehension creates a dynamic list. In this case, we make a list of reversed tuples and then sort it.

http://wiki.python.org/moin/HowTo/Sorting

Dictionary summary

- What's dictionary
- Dictionary literal
- creation
- Add item
- Delete item
- Loop through dict
- Sort by key or value
- mutable
- Function dir

Method

- items()
- keys()
- values()
- get()

Tuple summary

- What's tuple
- tuple literal
- Comparability
- Sorting
- Loop through tuple
- Sort by key or value
- Immutable
- Function dir
- Function sorted
-

Method

- count()
- index()