SWCON104
Web & Python Programming

Data Collection Types

Department of Software Convergence

경희대학교
KYUNG HEE UNIVERSITY

Today

- Storing data using other collection types
  - (Review) Lists
  - Sets
  - Tuples
  - Dictionaries

- You will be able to pick the one that best matches your problem to keep your code as simple and efficient as possible.

[Textbook]
Practical Programming
(An Introduction to Computer Science Using Python),
by Paul Gries, Jennifer Campbell, Jason Montojo,
The Pragmatic Bookshelf, 2017

Practice

- Practice_14_Data_Collection_Types.ipynb

List is a type

- Types
  - Integer
  - Float
  - Boolean
  - String

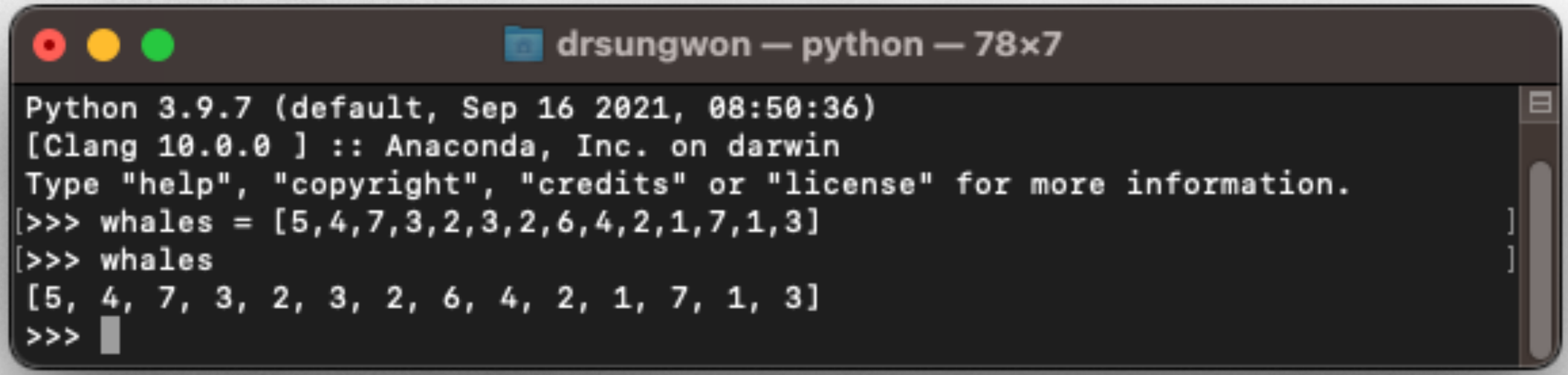
- Lists contain 0 or more objects : Collection of data!

Collection of data

- Number of gray whales counted near the Coal Oil Point Natural Reserve

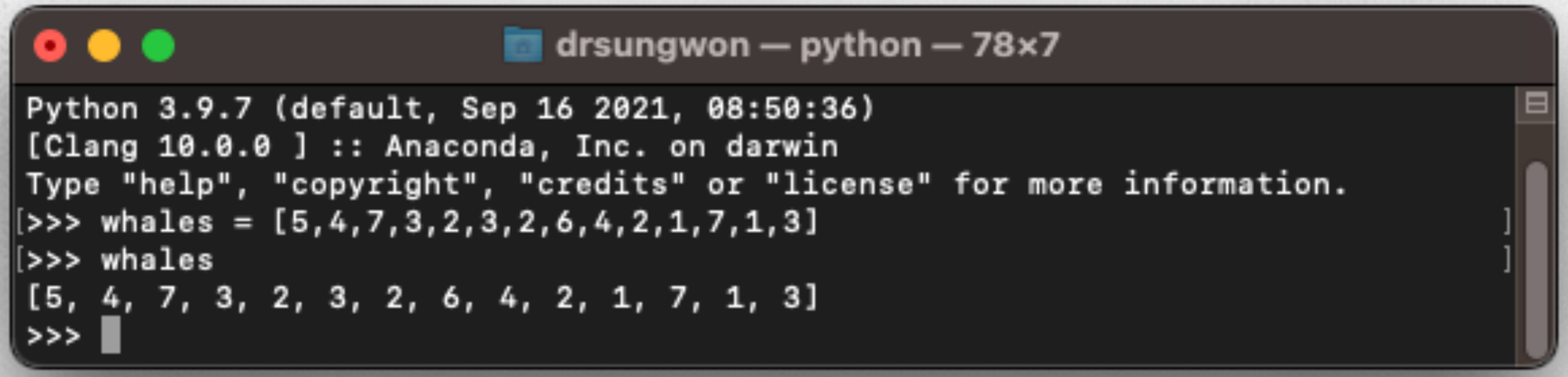
Day	Number of Whales	Day	Number of Whales
1	5	8	6
2	4	9	4
3	7	10	2
4	3	11	1
5	2	12	7
6	3	13	1
7	2	14	3

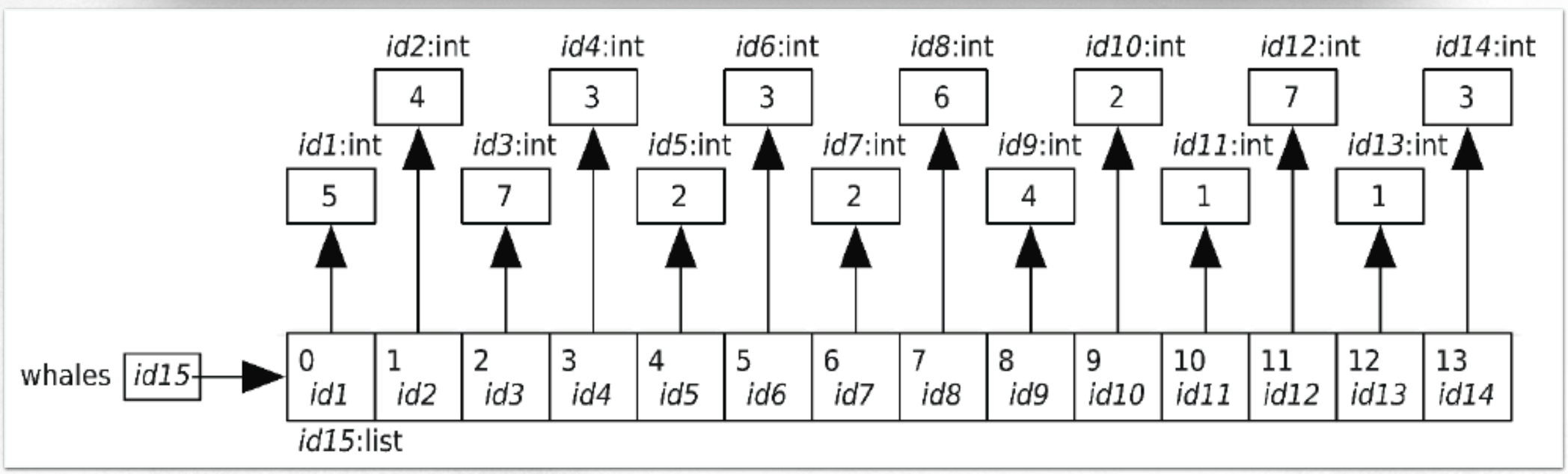
Table 9—Gray Whale Census



List

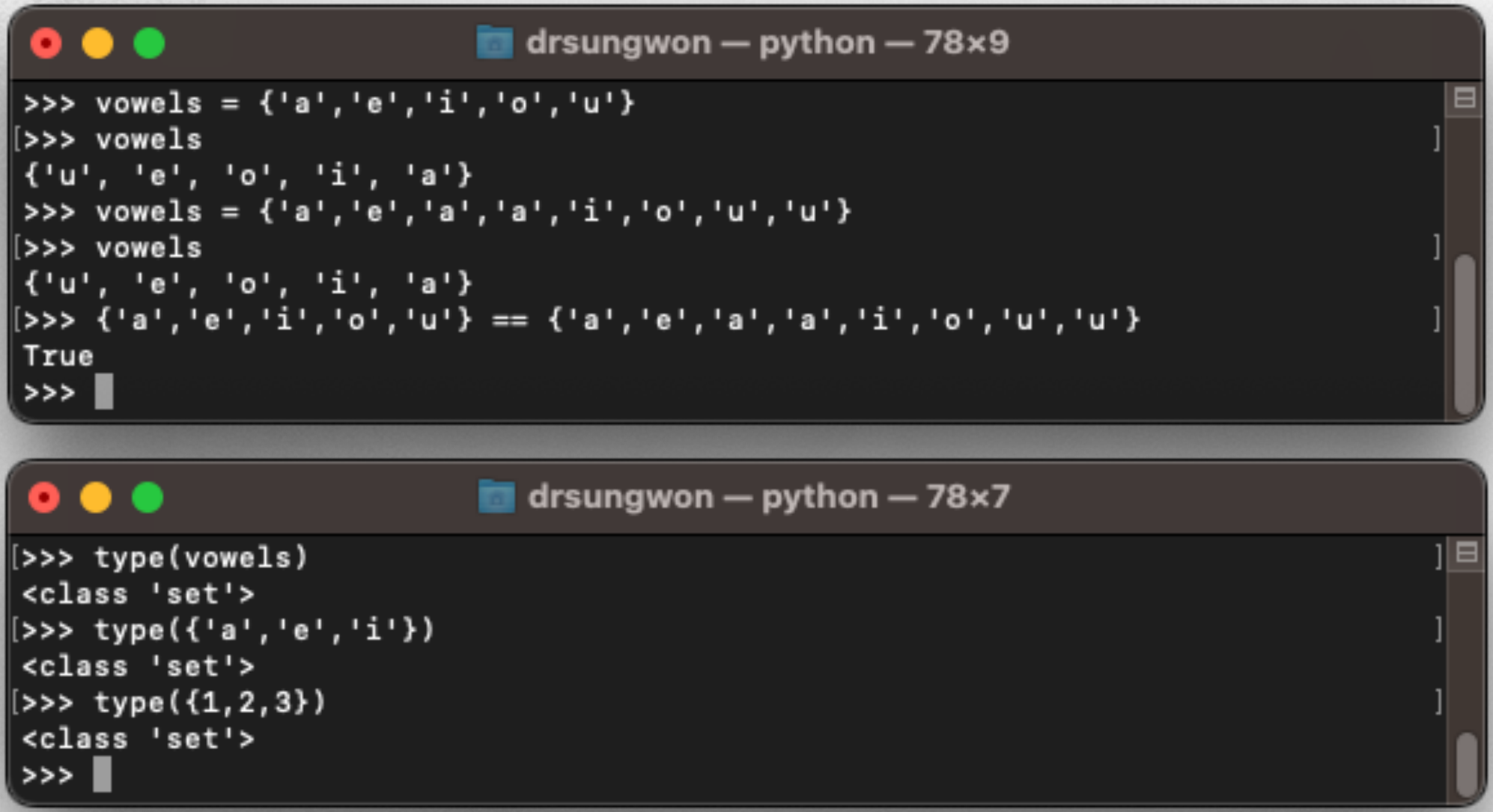
- List is a type
- A list is an object
- An object can be assigned to a variable





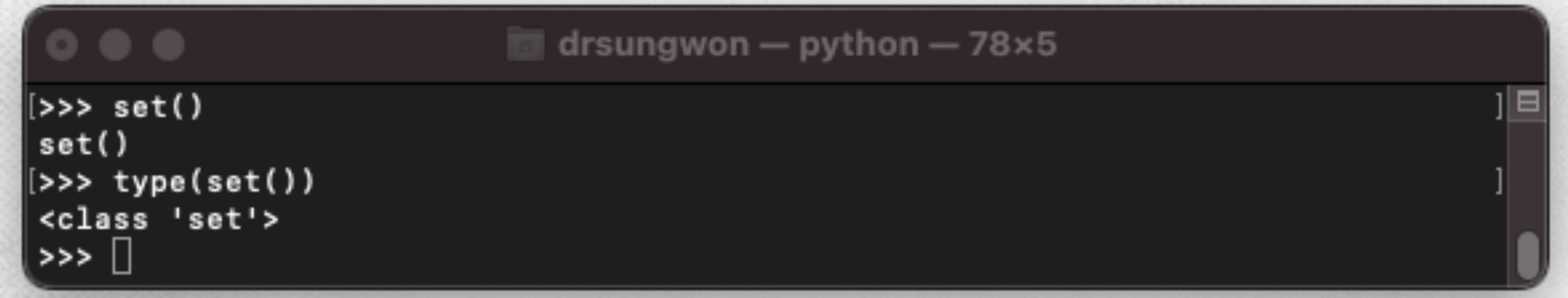
Set is a type

- A set is an unordered collection of distinct items
  - Unordered: items are not stored in any particular order
  - Distinct: Any item appears in a set at most once. No duplicates

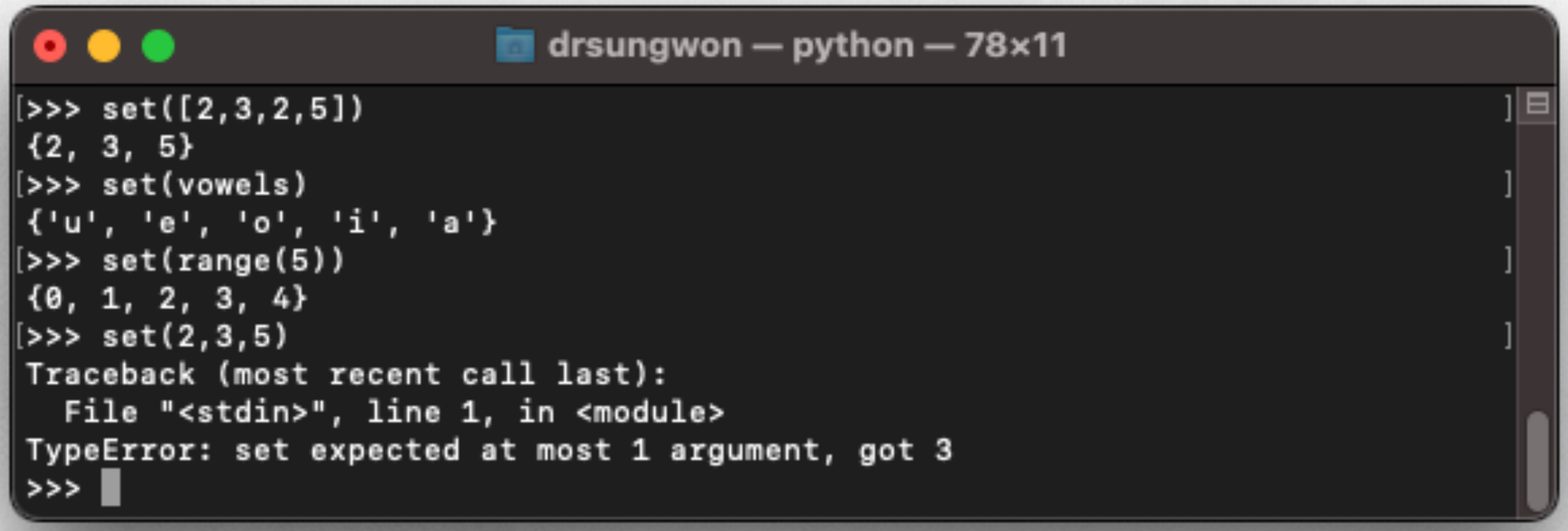


How to define a set

- An empty set is not {}, set()



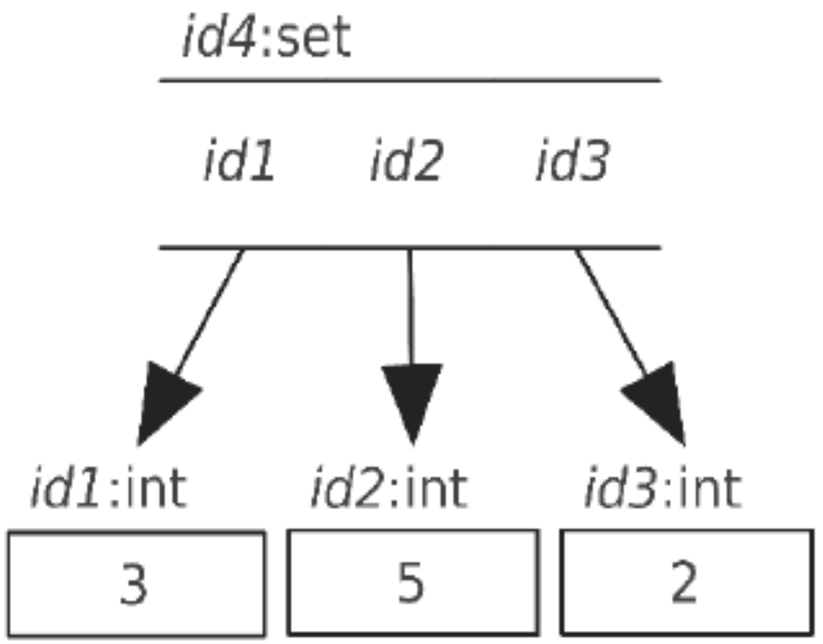
- Function set takes
  - either no arguments or,
  - a single argument (a list, a set, a range, and a tuple)



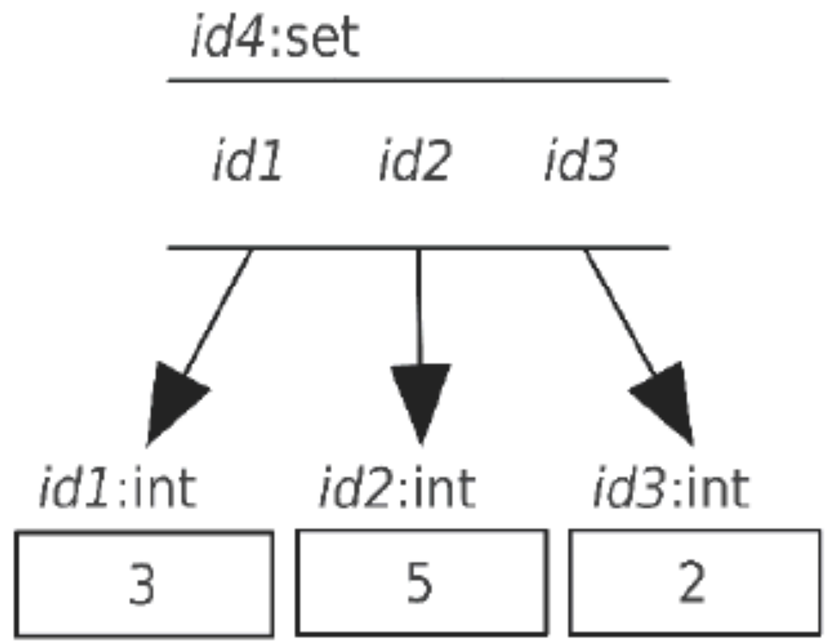
Set: memory model

```
drsungwon — python — 78×11
>>> set([2,3,2,5])
{2, 3, 5}
>>> set(vowels)
{'u', 'e', 'o', 'i', 'a'}
>>> set(range(5))
{0, 1, 2, 3, 4}
>>> set(2,3,5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: set expected at most 1 argument, got 3
>>>
```

`set([3, 5, 2])`



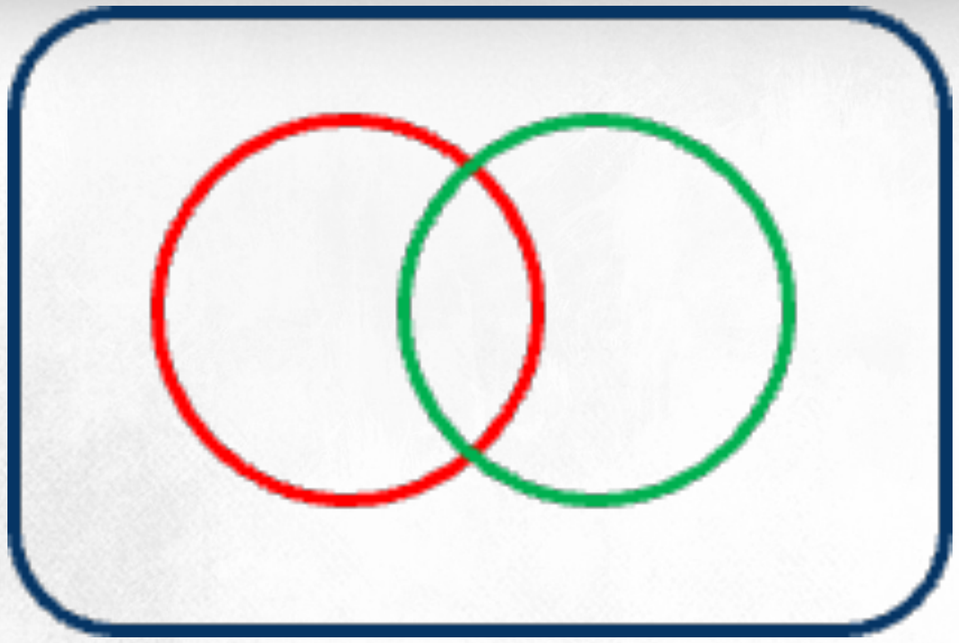
`set([2, 3, 5, 5, 2, 3])`



Set: operations

- In python, set operations are implemented as set methods

```
drsungwon — python — 78×7
>>> vowels = {'a','e','i','o','u'}
>>> vowels
{'u', 'e', 'o', 'i', 'a'}
>>> vowels.add('y')
>>> vowels
{'u', 'e', 'o', 'i', 'a', 'y'}
>>>
```



Set: methods

Method	Description
S.add(v)	Adds item v to a set S—this has no effect if v is already in S
S.clear()	Removes all items from set S
S.difference(other)	Returns a set with items that occur in set S but not in set other
S.intersection(other)	Returns a set with items that occur both in sets S and other
S.issubset(other)	Returns True if and only if all of set S's items are also in set other
S.issuperset(other)	Returns True if and only if set S contains all of set other's items
S.remove(v)	Removes item v from set S
S.symmetric_difference(other)	Returns a set with items that are in exactly one of sets S and other—any items that are in both sets are *not* included in the result
S.union(other)	Returns a set with items that are either in set S or other (or in both)

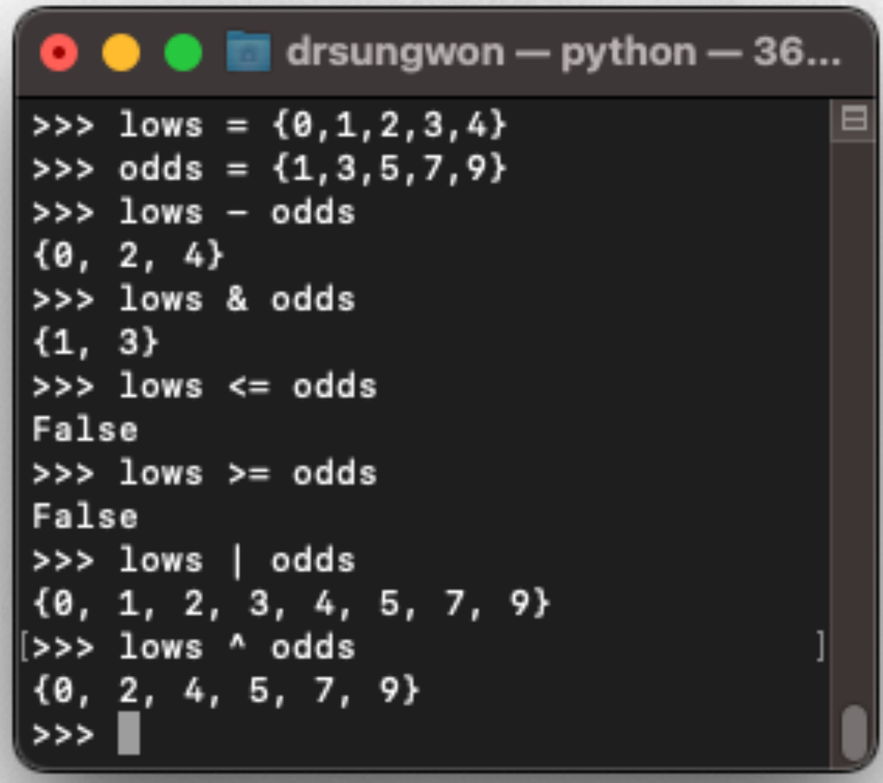
Table 14—Set Operations

```
drsungwon — python — 36...
>>> ten = set(range(10))
>>> lows = {0,1,2,3,4}
>>> odds = {1,3,5,7,9}
>>> lows.add(9)
>>> lows
{0, 1, 2, 3, 4, 9}
>>> lows.difference(odds)
{0, 2, 4}
>>> lows.intersection(odds)
{1, 3, 9}
>>> lows.issubset(ten)
True
>>> lows.issuperset(odds)
False
>>> lows.remove(0)
>>> lows
{1, 2, 3, 4, 9}
>>> lows.symmetric_difference(odds)
{2, 4, 5, 7}
>>> lows.union(odds)
{1, 2, 3, 4, 5, 7, 9}
>>> lows.clear()
>>> lows
set()
>>>
```

Set: operators

Method Call	Operator
set1.difference(set2)	set1 - set2
set1.intersection(set2)	set1 & set2
set1.issubset(set2)	set1 <= set2
set1.issuperset(set2)	set1 >= set2
set1.union(set2)	set1 | set2
set1.symmetric_difference(set2)	set1 ^ set2

Table 15—Set Operators



Set: example

- Arctic birds

```
observations_file = open('observations.txt')
birds_observed = set()
for line in observations_file:
    bird = line.strip()
    birds_observed.add(bird)

print(birds_observed)
```

```
for species in birds_observed:
    print(species)
```

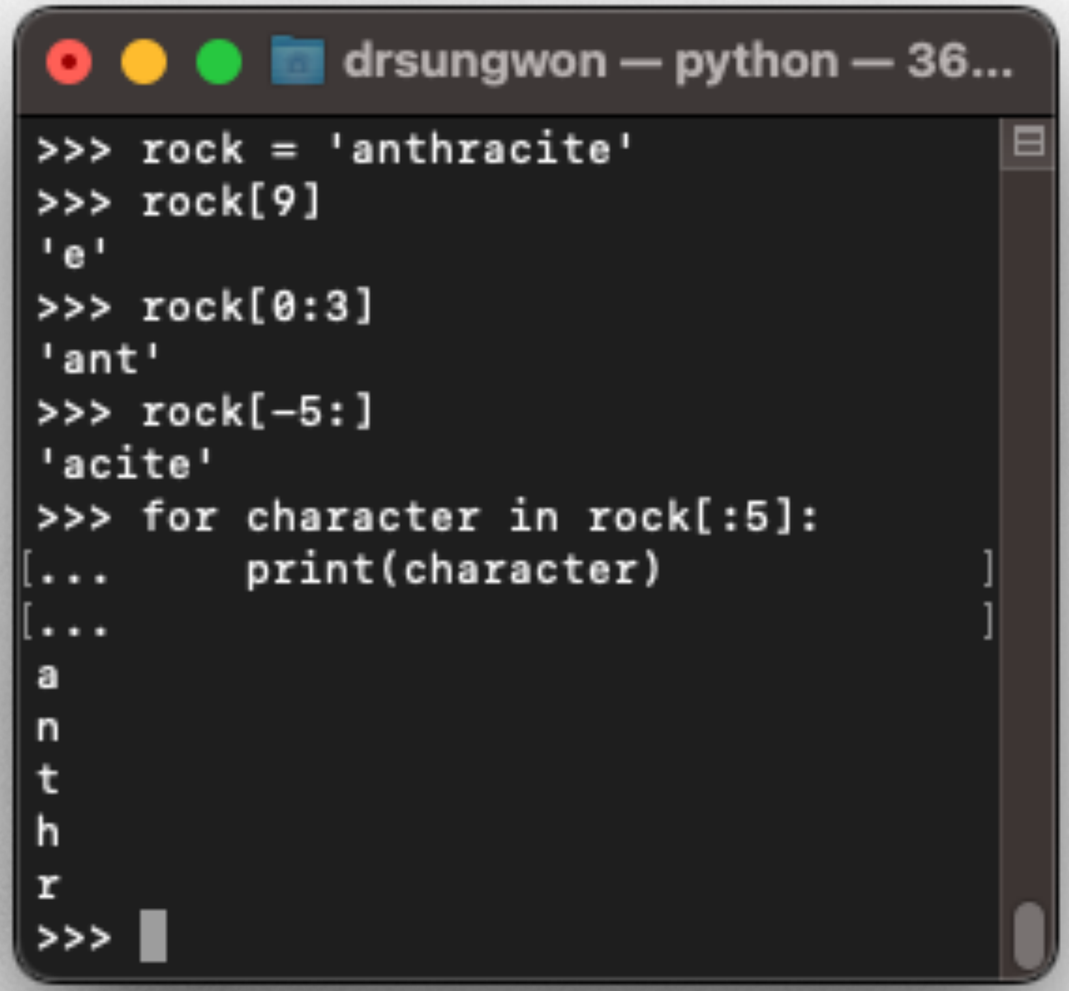
observations.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
canada goose
canada goose
long-tailed jaeger
canada goose
snow goose
canada goose
long-tailed jaeger
canada goose
northern fulmar
```

Recall string

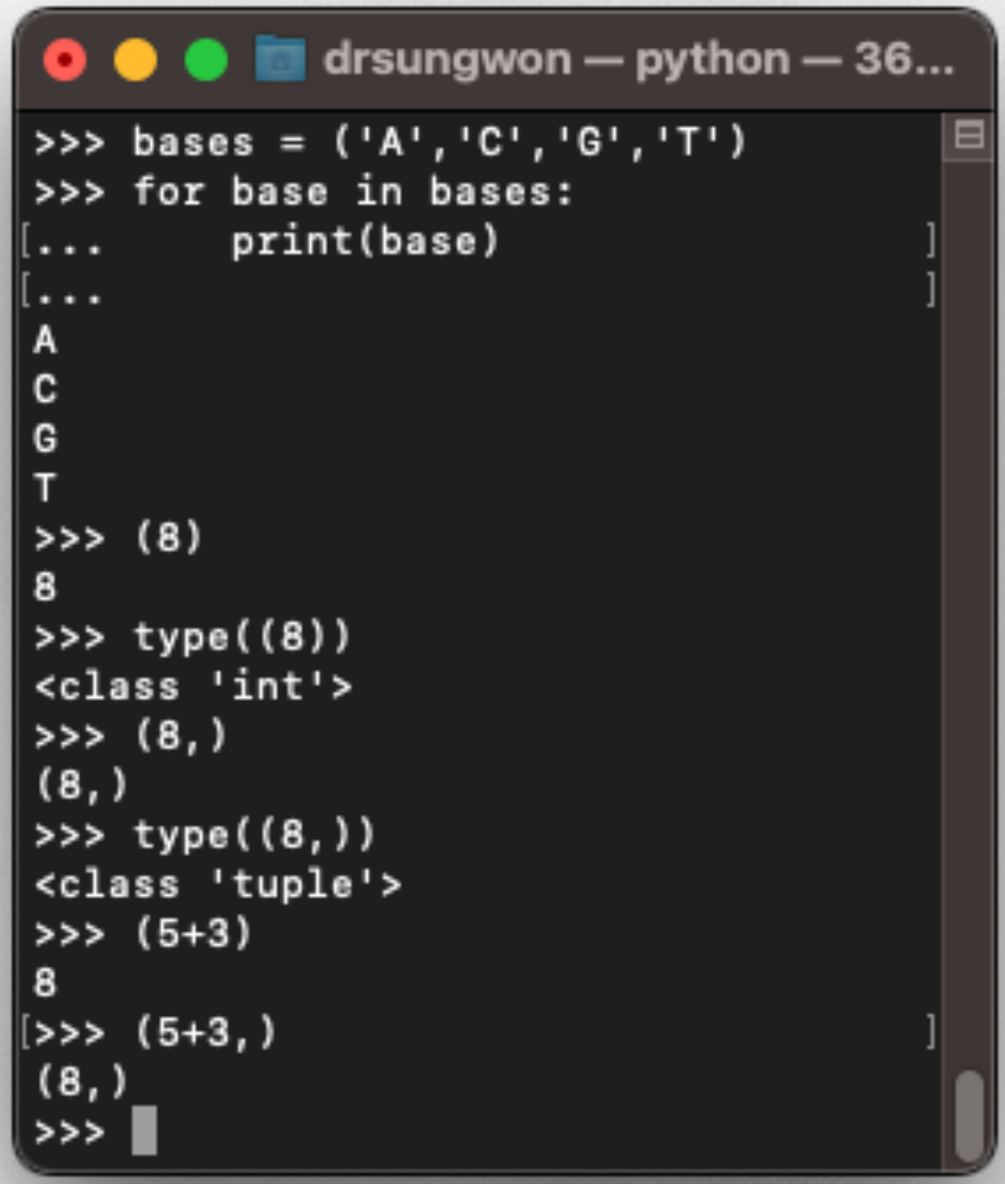
- String is an immutable sequence of characters.
- The characters in a string are ordered.
- A string can be indexed and sliced like a list to create new strings.



Tuple is a type

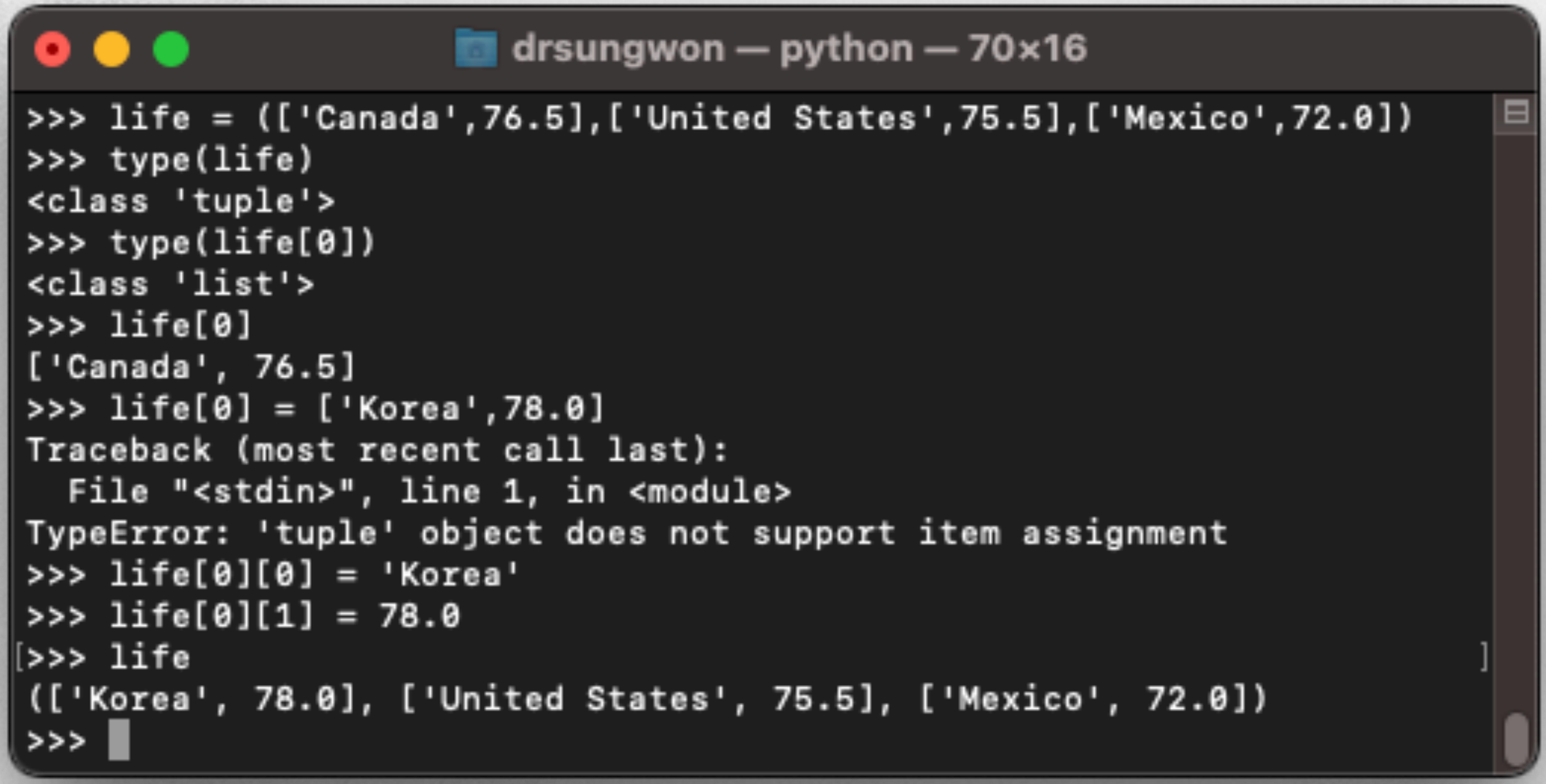
- Tuple is another immutable sequence type.
- Tuples are defined using parentheses.
- Tuples can be subscripted, sliced, and looped over.

- The empty tuple is ()
- A Tuple with one element is (x,)

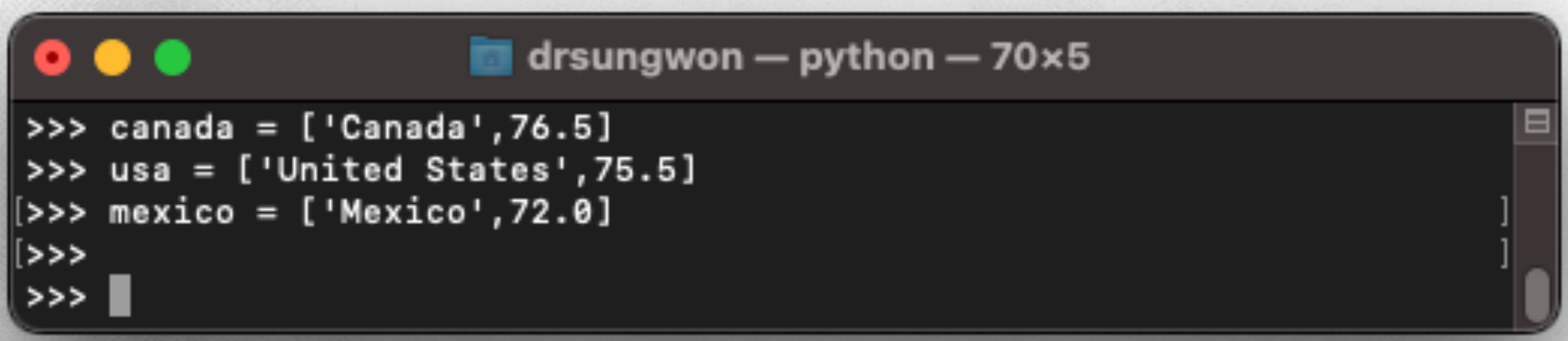


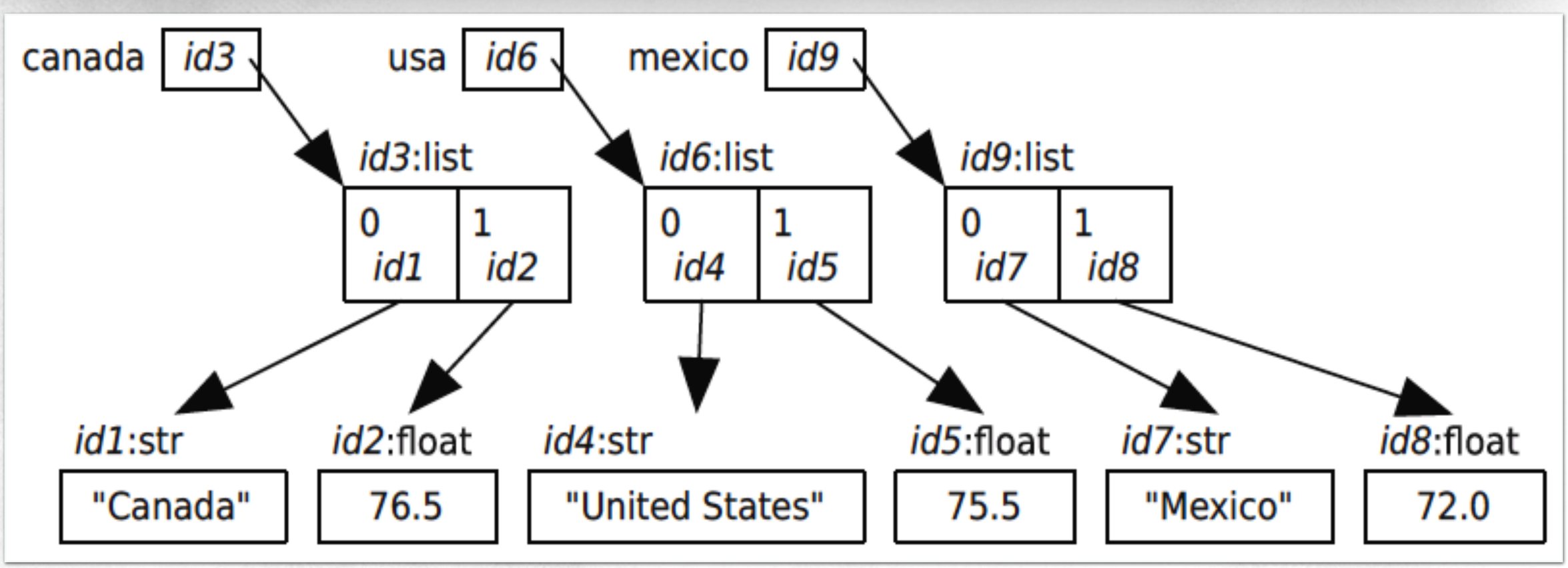
Tuple is immutable

- The references contained in a tuple cannot be changed after the tuple has been created, though the objects referred to may be mutated.

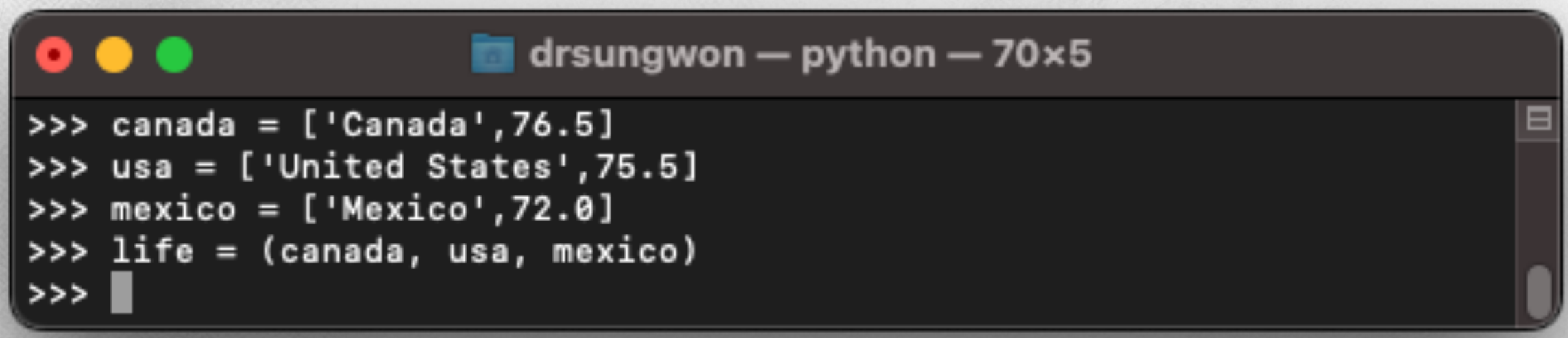


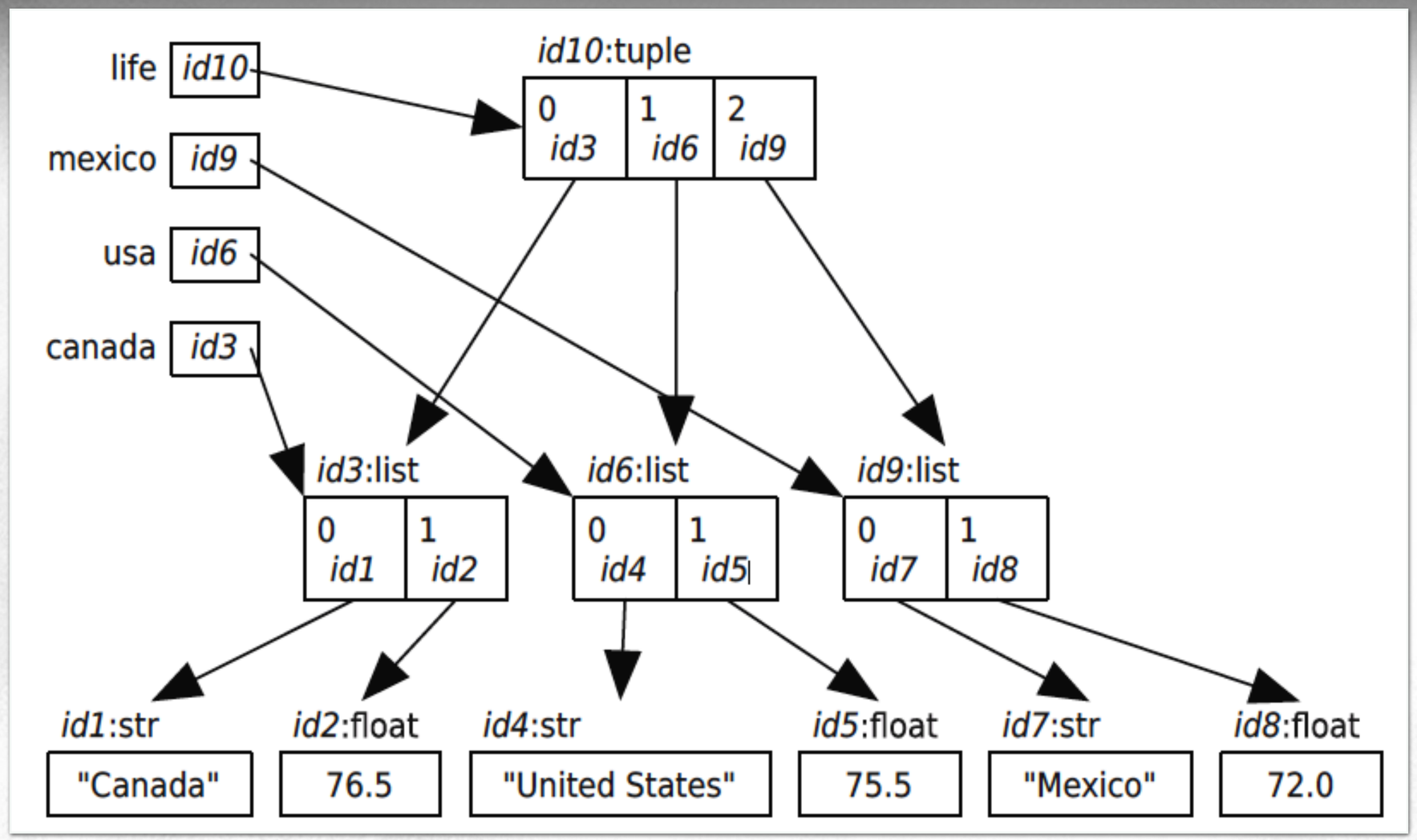
Tuple: memory model





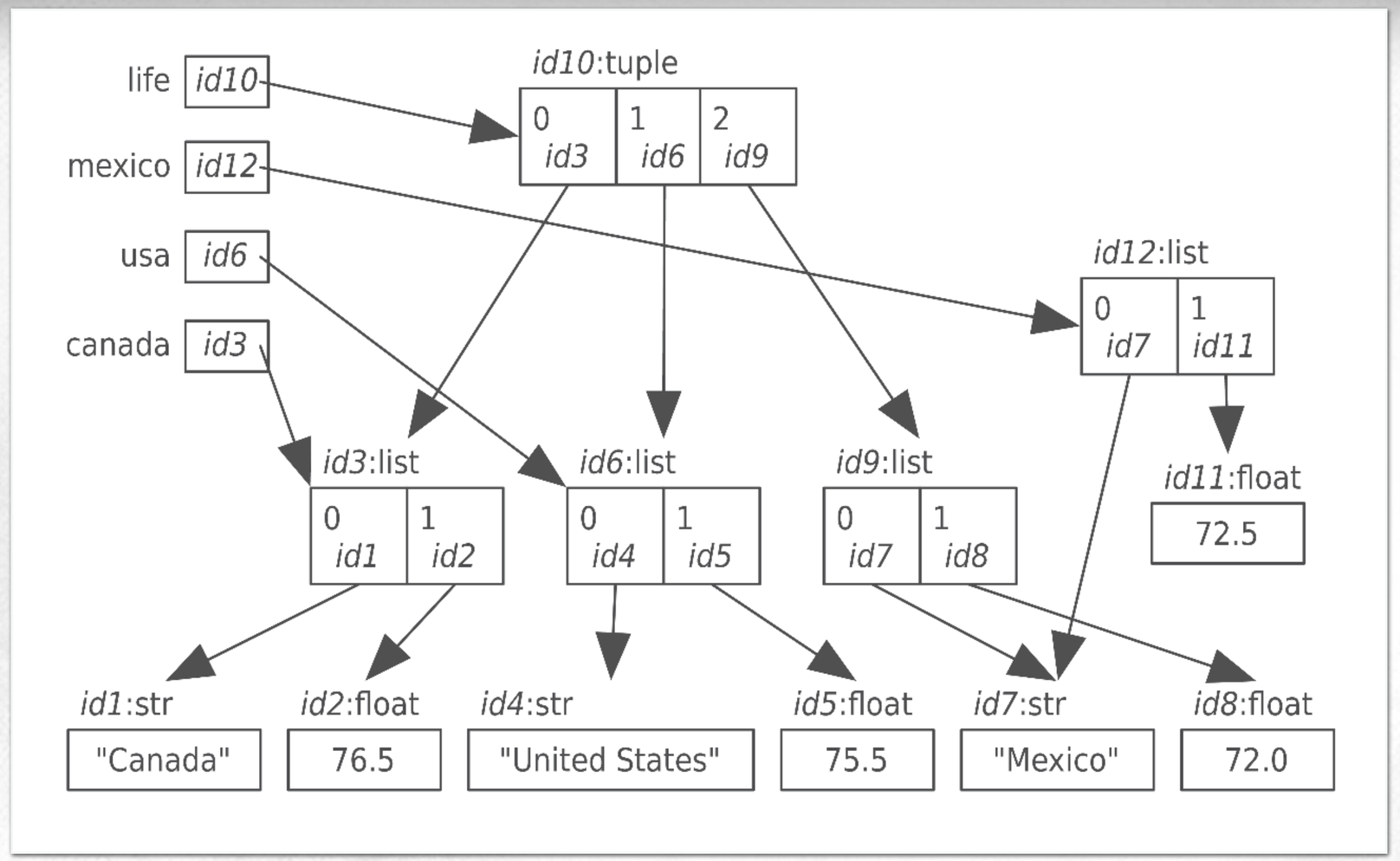
Tuple: memory model





Tuple: memory model

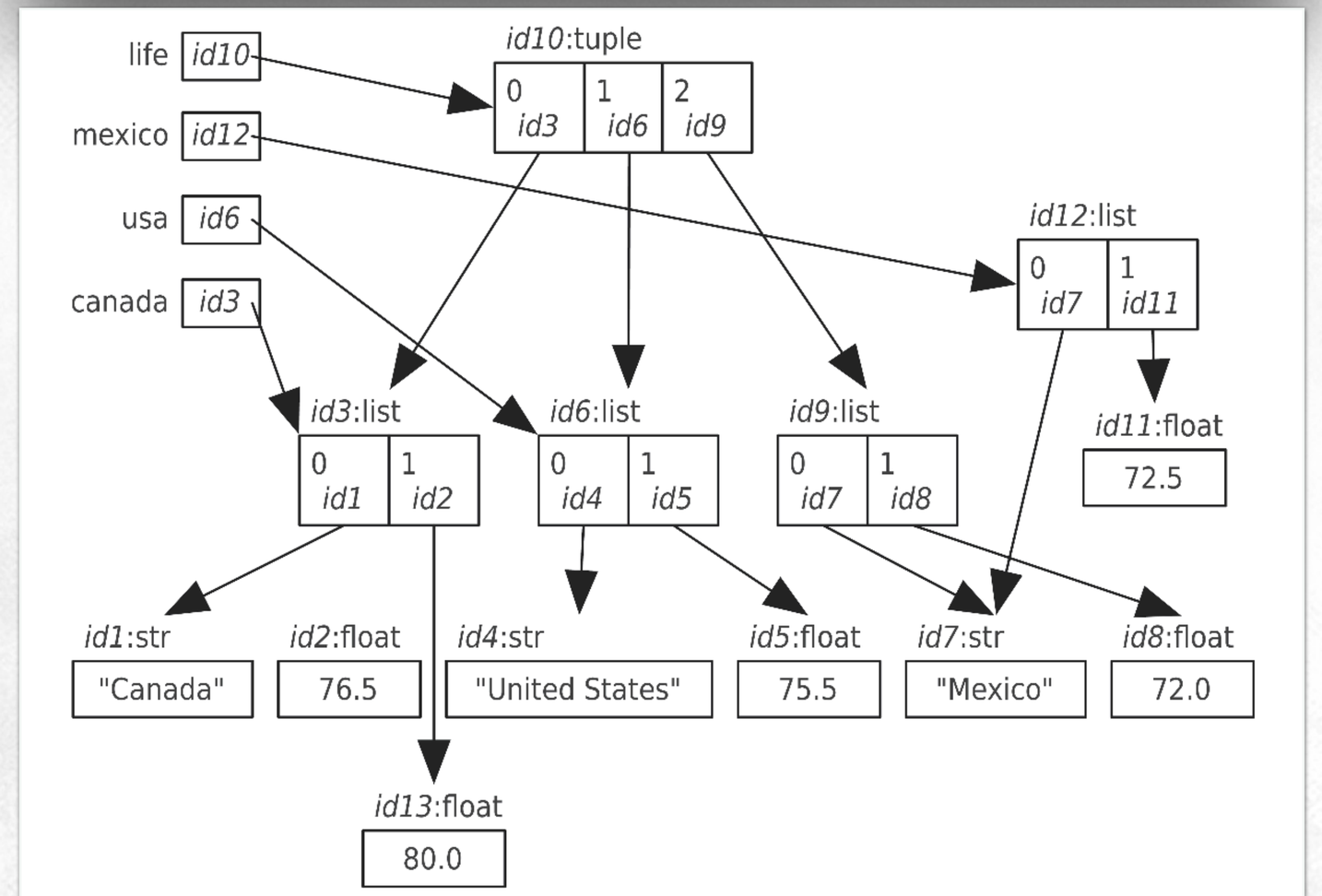
```
drsungwon — python — 70×8
>>> canada = ['Canada',76.5]
>>> usa = ['United States',75.5]
>>> mexico = ['Mexico',72.0]
>>> life = (canada, usa, mexico)
>>> mexico = ['Mexico',72.5]
>>> life
(['Canada', 76.5], ['United States', 75.5], ['Mexico', 72.0])
>>>
```



Tuple: memory model

drsungwon — python — 70×13

```
>>> canada = ['Canada',76.5]
>>> usa = ['United States',75.5]
>>> mexico = ['Mexico',72.0]
>>> life = (canada, usa, mexico)
>>> mexico = ['Mexico',72.5]
>>> life
(['Canada', 76.5], ['United States', 75.5], ['Mexico', 72.0])
>>> life[0][1] = 80.0
>>> canada
['Canada', 80.0]
>>> life
(['Canada', 80.0], ['United States', 75.5], ['Mexico', 72.0])
>>>
```



Tuple: multiple assignment

```
>>> 10,20
(10, 20)
>>> a = 10,20
>>> a
(10, 20)
>>> type(a)
<class 'tuple'>
```

```
>>> type(10,20)
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    type(10,20)
TypeError: type() takes 1 or 3 arguments
>>> type((10,20))
<class 'tuple'>
```

```
>>> x,y = 10,20
>>> x
10
>>> y
20
>>> type(x)
<class 'int'>
>>> type(y)
<class 'int'>
```

```
>>> [[w,x],[[y],z]] = [{10,20},[(30,),40]]
>>> w
10
>>> x
20
>>> y
30
>>> z
40
>>> type(w)
<class 'int'>
>>> type(x)
<class 'int'>
>>> type(y)
<class 'int'>
>>> type(z)
<class 'int'>
```

Tuple: multiple assignment (swap)

- Traditional way to swap

```
>>> s1 = 'first'
>>> s2 = 'second'
>>> temp = s2
>>> s2 = s1
>>> s1 = temp
>>> s1
'second'
>>> s2
'first'
```

- Swap in Python

```
>>> s1 = 'first'
>>> s2 = 'second'
>>> s1,s2 = s2,s1
>>> s1
'second'
>>> s2
'first'
```

Dictionary: intro

```
observations_file = open('observations.txt')
bird_counts = []

for line in observations_file:
    bird = line.strip()
    found = False

    # Find bird in the list of bird counts.
    for entry in bird_counts:
        if entry[0] == bird:
            entry[1] = entry[1] + 1
            found = True

    if not found:
        bird_counts.append([bird, 1])

observations_file.close()
for entry in bird_counts:
    print(entry[0], entry[1])
```

observations.txt - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
canada goose
canada goose
long-tailed jaeger
canada goose
snow goose
canada goose
long-tailed jaeger
canada goose
northern fulmar
```

NOTE: If the list is very long, scanning the list of names each time we want to add a new observation would take a long, long time

Dictionary (a.k.a. map)

- A dictionary is an unordered mutable collection of key/value pairs
- English dictionaries map words to definitions
- Python dictionaries map a key to a value

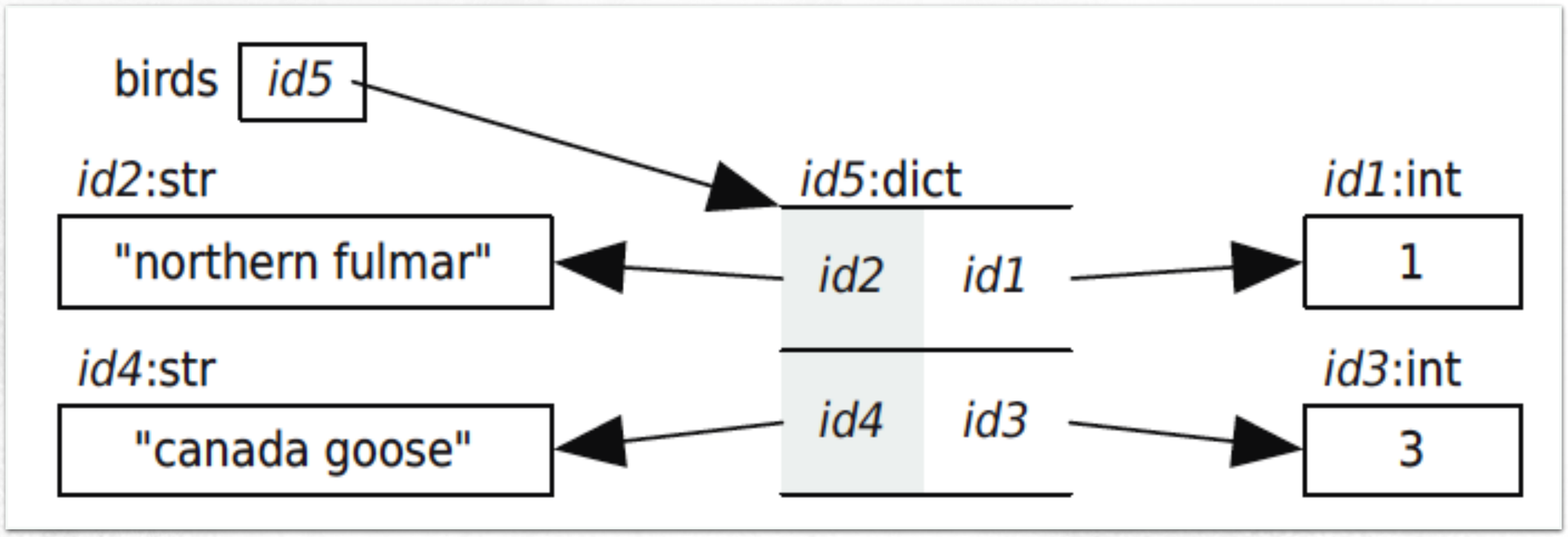
- The keys form a set.
  - Any key can appear once at most in a dictionary
  - Keys must be immutable
  - Values are mutable

Dictionary: example

- An empty dictionary is {}

```
>>> bird_to_observations = {'canada goose': 3, 'northern fulmar': 1}
>>> bird_to_observations
{'canada goose': 3, 'northern fulmar': 1}
>>> type(bird_to_observations)
<class 'dict'>
```

```
>>> bird_to_observations['northern fulmar']
1
>>> bird_to_observations['canada goose']
3
>>> bird_to_observations['long-tailed jaeger']
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    bird_to_observations['long-tailed jaeger']
KeyError: 'long-tailed jaeger'
```



Dictionary: looping

- Updating and checking membership/ Looping over dictionaries

```
>>> bird_to_observations = {}
>>> bird_to_observations['snow goose'] = 33
>>> bird_to_observations['eagle'] = 999
>>> bird_to_observations
{'snow goose': 33, 'eagle': 999}
```

```
>>> bird_to_observations['eagle'] = 9
>>> bird_to_observations
{'snow goose': 33, 'eagle': 9}
```

```
>>> del bird_to_observations['snow goose']
>>> bird_to_observations
{'eagle': 9}
>>> del bird_to_observations['gannet']
Traceback (most recent call last):
  File "<pyshell#94>", line 1, in <module>
    del bird_to_observations['gannet']
KeyError: 'gannet'
```

```
>>> 'eagle' in bird_to_observations
True
>>> 'gannet' in bird_to_observations
False
```

```
>>> bird_to_observations = {'canada goose': 183,
'long-tailed jaeger': 71, 'snow goose': 63, 'nort
hern fulmar': 1}
>>> for bird in bird_to_observations:
        print(bird, bird_to_observations[bird])


snow goose 63
long-tailed jaeger 71
northern fulmar 1
canada goose 183
```

Dictionary: methods

Method	Description
D.clear()	Removes all key/value pairs from dictionary D.
D.get(k)	Returns the value associated with key k, or None if the key isn't present. (Usually you'll want to use D[k] instead.)
D.get(k, v)	Returns the value associated with key k, or a default value v if the key isn't present.
D.keys()	Returns dictionary D's keys as a set-like object—entries are guaranteed to be unique.
D.items()	Returns dictionary D's (key, value) pairs as set-like objects.
D.pop(k)	Removes key k from dictionary D and returns the value that was associated with k—if k isn't in D, an error is raised.
D.pop(k, v)	Removes key k from dictionary D and returns the value that was associated with k; if k isn't in D , returns v.
D.setdefault(k)	Returns the value associated with key k in D.
D.setdefault(k, v)	Returns the value associated with key k in D; if k isn't a key in D, adds the key k with the value v to D and returns v.
D.values()	Returns dictionary D's values as a list-like object—entries may or may not be unique.
D.update(other)	Updates dictionary D with the contents of dictionary other; for each key in other, if it is also a key in D, replaces that key in D's value with the value from other; for each key in other, if that key isn't in D, adds that key/value pair to D.

Table 16—Dictionary Methods

```
>>> scientist_to_birthdate = {'Newton' : 1642, 'Darwin' : 1809,
...                           'Turing' : 1912}
>>> scientist_to_birthdate.keys()
dict_keys(['Darwin', 'Newton', 'Turing'])
>>> scientist_to_birthdate.values()
dict_values([1809, 1642, 1912])
>>> scientist_to_birthdate.items()
dict_items([('Darwin', 1809), ('Newton', 1642), ('Turing', 1912)])
>>> scientist_to_birthdate.get('Newton')
1642
>>> scientist_to_birthdate.get('Curie', 1867)
1867
>>> scientist_to_birthdate
{'Darwin': 1809, 'Newton': 1642, 'Turing': 1912}
>>> researcher_to_birthdate = {'Curie' : 1867, 'Hopper' : 1906,
...                            'Franklin' : 1920}
>>> scientist_to_birthdate.update(researcher_to_birthdate)
>>> scientist_to_birthdate
{'Hopper': 1906, 'Darwin': 1809, 'Turing': 1912, 'Newton': 1642,
 'Franklin': 1920, 'Curie': 1867}
>>> researcher_to_birthdate
{'Franklin': 1920, 'Hopper': 1906, 'Curie': 1867}
>>> researcher_to_birthdate.clear()
>>> researcher_to_birthdate
{}
```

Dictionary: examples

```
observations_file = open('observations.txt')
bird_to_observations = {}

for line in observations_file:
    bird = line.strip()
    if bird in bird_to_observations:
        bird_to_observations[bird] = bird_to_observations[bird] + 1
    else:
        bird_to_observations[bird] = 1

observations_file.close()

for bird, observations in bird_to_observations.items():
    print(bird, observations)
```

```
observations_file = open('observations.txt')
bird_to_observations = {}

for line in observations_file:
    bird = line.strip()
    bird_to_observations[bird] = bird_to_observations.get(bird, 0) + 1

observations_file.close()

for bird, observations in bird_to_observations.items():
    print(bird, observations)
```

Dictionary: examples

```
>>> sorted_birds = sorted(bird_to_observations.keys())
>>> for bird in sorted_birds:
        print(bird, bird_to_observations[bird])


canada goose 5
long-tailed jaeger 2
northern fulmar 1
snow goose 1
```

```
>>> type(sorted_birds)
<class 'list'>
>>> type(bird_to_observations.keys())
<class 'dict_keys'>
```

Dictionary: inverting

- Keys -> Values
- Values -> Keys

- No guarantee that values are unique
  - Ex) inverting {'a':1, 'b':1, 'c':1} ?
  - {1: ['a', 'b', 'c']} is possible

Dictionary: inverting

```
>>> bird_to_observations
{'snow goose': 1, 'canada goose': 5, 'northern fulmar': 1,
'long-tailed jaeger': 2}
```

```
observations_to_birds_list = {}
for bird, observations in bird_to_observations.items():
    if observations in observations_to_birds_list:
        observations_to_birds_list[observations].append(bird)
    else:
        observations_to_birds_list[observations] = [bird]
```

```
>>> observations_to_birds_list
{1: ['snow goose', 'northern fulmar'], 2: ['long-tailed jaeger'],
5: ['canada goose']}
```

```
>>> observations_sorted = sorted(observations_to_birds_list.keys())
>>> for observations in observations_sorted:
        print(observations,":", end="")
        for bird in observations_to_birds_list[observations]:
                print(" ", bird, end="")
        print()


1 :  snow goose  northern fulmar
2 :  long-tailed jaeger
5 :  canada goose
```

in operator on tuples, sets, and dictionaries

- Checks whether an item is a member of a tuple or a set
- Checks whether a value is a key in the dictionary
- The values in the dictionary are ignored

```
>>> odds = set([1,3,5,7,9])
>>> evens = (0,2,4,6,8)
>>> type(odds)
<class 'set'>
>>> type(evens)
<class 'tuple'>
```

```
>>> 9 in odds
True
>>> 8 in odds
False
>>> '9' in odds
False
```

```
>>> 4 in evens
True
>>> '4' in evens
False
>>> 11 in evens
False
```

```
>>> bird_to_observations = {'canada goose': 183, 'long-tailed
jaeger': 71, 'snow goose': 63, 'northern fulmar': 1}
>>> 'snow goose' in bird_to_observations
True
>>> 183 in bird_to_observations
False
```

Comparing collections

Collection	Mutable?	Ordered?	Use When…
str	No	Yes	You want to keep track of text.
list	Yes	Yes	You want to keep track of an ordered sequence that you want to update.
tuple	No	Yes	You want to build an ordered sequence that you know won't change or that you want to use as a key in a dictionary or as a value in a set.
set	Yes	No	You want to keep track of values, but order doesn't matter, and you don't want to keep duplicates. The values must be immutable.
dictionary	Yes	No	You want to keep a mapping of keys to values. The keys must be immutable.

Table 17—Features of Python Collections

Summary

- Sets are used in Python to store unordered collections of unique values. They support the same operations as sets in mathematics.
- Tuples are another kind of Python sequence. Tuples are ordered sequences like lists, except they are immutable.
- Dictionaries are used to store unordered collections of key/value pairs. The keys must be immutable, but the values need not be.
- Looking things up in sets and dictionaries is much faster than searching through lists. If you have a program that is doing the latter, consider changing your choice of data structures.

Thank you

경희대학교
KYUNG HEE UNIVERSITY