Containers and Randomness

Kenn H. Kim, Ph. D.

School of Business
Clemson University

1

Containers and Randomness

- Container Class dict
- Encoding of String Characters
- Randomness and Random Sampling

2

Dictionaries

Goal: a container of employee records indexed by employee SS#

```
>>> employee[987654321]
['Yu', 'Tsun']
>>> employee[864209753]
['Anna', 'Karenina']
>>> employee[100010010]
['Hans', 'Castorp']
```

Problems:

- the range of SS#s is huge
- SS#s are not really integers

Solution: the dictionary class `dict`

key	value
'864-20-9753'	['Anna', 'Karenina']
'987-65-4321'	['Yu', 'Tsun']
'100-01-0010'	['Hans', 'Castorp']

A dictionary contains (key, value) pairs

```
>>> employee = {
        '864-20-9753': ['Anna',
'Karenina'],
        '987-65-4321': ['Yu', 'Tsun'],
        '100-01-0010': ['Hans', 'Castorp']}
>>> employee['987-65-4321']
['Yu', 'Tsun']
>>> employee['864-20-9753']
['Anna', 'Karenina']
```

A key can be used as an index to access the corresponding value

3

Properties of Dictionaries

CLEMSON College of BUSINESS

The empty dictionary is `{}`

Dictionaries are not ordered

```
>>> employee = {
        '864-20-9753': ['Anna', 'Karenina'],
        '987-65-4321': ['Yu', 'Tsun'],
        '100-01-0010': ['Hans', 'Castorp']}
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-20-
9753': ['Anna', 'Karenina'], '987-65-4321':
['Yu', 'Tsun']}
>>> employee['123-45-6789'] = 'Holden Cafield'
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-20-
9753': ['Anna', 'Karenina'], '987-65-4321':
['Yu', 'Tsun'], '123-45-6789': 'Holden Cafield'}
>>> employee['123-45-6789'] = 'Holden Caulfield'
>>> employee
{'100-01-0010': ['Hans', 'Castorp'], '864-20-
9753': ['Anna', 'Karenina'], '987-65-4321':
['Yu', 'Tsun'], '123-45-6789': 'Holden
Caulfield'}
```

Dictionaries are mutable

- new (key,value) pairs can be added
- the value corresponding to a key can be modified

Dictionary keys must be immutable

```
>>> employee = {[1,2]:1, [2,3]:3}
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    employee = {[1,2]:1, [2,3]:3}
TypeError: unhashable type: 'list'
```

4

Dictionary Operators

Class `dict` supports some of the same operators as class `list`

```
>>> days = {'Mo':1, 'Tu':2, 'W':3}
>>> days['Mo']
1
>>> days['Th'] = 5
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 5, 'W': 3}
>>> days['Th'] = 4
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 4, 'W': 3}
>>> 'Fr' in days
False
>>> len(days)
4
```

Class dict does not support all the operators that class list supports

- + and * for example

5

Dictionary Methods

Operation	Explanation
`d.items()`	Returns a view of the (key, value) pairs in `d`
`d.keys()`	Returns a view of the keys of `d`
`d.pop(key)`	Removes the (key, value) pair with key `key` from `d` and returns the value
`d.update(d2)`	Adds the (key, value) pairs of dictionary `d2` to `d`
`d.values()`	Returns a view of the values of `d`

The containers returned by `d.items()`, `d.keys()`, and `d.values()` (called views) can be iterated over

```
>>> days
{'Mo': 1, 'Tu': 2, 'Th': 4, 'W': 3}
>>> days.pop('Tu')
2
>>> days
{'Mo': 1, 'Th': 4, 'W': 3}
>>> days2 = {'Tu':2, 'Fr':5}
>>> days.update(days2)
>>> days
{'Fr': 5, 'W': 3, 'Th': 4, 'Mo': 1,
'Tu': 2}
>>> days.items()
dict_items([('Fr', 5), ('W', 3), ('Th',
4), ('Mo', 1), ('Tu', 2)])
>>> days.keys()
dict_keys(['Fr', 'W', 'Th', 'Mo',
'Tu'])
>>> vals = days.values()
>>> vals
dict_values([5, 3, 4, 1, 2])
>>>
```

6

Dictionary vs. Multi-way `if` Statement

Uses of a dictionary:

- container with custom indexes
- alternative to the multi-way `if` statement

```
def complete(abbreviation):
    'returns day of the week corresponding to abbreviation'

    if abbreviation == 'Mo':
        return 'Monday'
    elif abbreviation == 'Tu':
        return 'Tuesday'
    elif
        ......
    else: # abbreviation must be Su
        return 'Sunday'
```

```
def complete(abbreviation):
    'returns day of the week corresponding to abbreviation'

    days = {'Mo': 'Monday', 'Tu':'Tuesday', 'We': 'Wednesday',
            'Th': 'Thursday', 'Fr': 'Friday', 'Sa': 'Saturday',
            'Su':'Sunday'}

    return days[abbreviation]
```

7

Dictionary as a Container of Counters

CLEMSON College of BUSINESS

Uses of a dictionary:

- container with custom indexes
- alternative to the multi-way `if` statement
- container of counters

Problem: computing the number of occurrences of items in a list

```
>>> grades = [95, 96, 100, 85, 95, 90, 95, 100, 100]
>>> frequency(grades)
{96: 1, 90: 1, 100: 3, 85: 1, 95: 3}
>>>
```

Solution:

- a dictionary mapping a grade (the key) to its counter (the value)
- iterate through the list and, for each grade, increment the counter corresponding to the grade.

8

Dictionary as a Container of Counters

Problem: computing the number of occurrences of items in a list

```
>>> grades = [95, 96, 100, 85, 95, 90, 95, 100, 100]
              ^   ^   ^    ^   ^   ^   ^
```


```
def frequency(itemList):
    'returns frequency of items in itemList'

    counters = {}
    for item in itemList:
        if item in counters: # increment item counter
            counters[item] += 1
        else: # create item counter
            counters[item] = 1
    return counters
```

9

Exercise (Notebook)

Implement function `wordcount()` that takes as input a text—as a string— and prints the frequency of each word in the text; assume there is no punctuation in the text.

```
>>> text = 'all animals are equal but some animals are more equal than other'
>>> wordCount(text)
all      appears 1 time.
animals  appears 2 times.
some     appears 1 time.
equal    appears 2 times.
but      appears 1 time.
other    appears 1 time.
are      appears 2 times.
than     appears 1 time.
more     appears 1 time.
>>>
```

10

Exercise (Notebook)

CLEMSON College of BUSINESS

Implement function `lookup()` that implements a phone book lookup application. Your function takes, as input, a dictionary representing a phone book, mapping tuples (containing the first and last name) to strings (containing phone numbers)

```
>>> phonebook = {
        ('Anna','Karenina'):'(123)456-78-90',
        ('Yu', 'Tsun'):'(901)234-56-78',
        ('Hans', 'Castorp'):'(321)908-76-54'}
>>> lookup(phonebook)
Enter the first name: Anna
Enter the last name: Karenina
(123)456-78-90
Enter the first name:
```

12

Randomness

CLEMSON College of BUSINESS

Some apps need numbers generated “at random” (i.e., from some probability distribution):

- scientific computing
- financial simulations
- cryptography
- computer games

Truly random numbers are hard to generate

Most often, a pseudorandom number generator is used

- numbers only appear to be random
- they are really generated using a deterministic process

The Python standard library module `random` provides a pseudo random number generator as well useful sampling functions

19

Standard Library module `random`

Function `randrange()` returns a "random" integer number from a given range

Example usage: simulate the throws of a die

Function `uniform()` returns a "random" `float` number from a given range

range is from 1 up to (but not including) 7

```
>>> import random
>>> random.randrange(1, 7)
2
>>> random.randrange(1, 7)
1
>>> random.randrange(1, 7)
4
>>> random.randrange(1, 7)
2
>>> random.uniform(0, 1)
0.19831634437485302
>>> random.uniform(0, 1)
0.027077323233875905
>>> random.uniform(0, 1)
0.8208477833085261
>>>
```

20

Standard Library module `random`

Defined in module `random` are functions `shuffle()`, `choice()`, `sample()`, …

```
>>> names = ['Ann', 'Bob', 'Cal', 'Dee', 'Eve', 'Flo', 'Hal', 'Ike']
>>> import random
>>> random.shuffle(names)
>>> names
['Hal', 'Dee', 'Bob', 'Ike', 'Cal', 'Eve', 'Flo', 'Ann']
>>> random.choice(names)
'Bob'
>>> random.choice(names)
'Ann'
>>> random.choice(names)
'Cal'
>>> random.choice(names)
'Cal'
>>> random.sample(names, 3)
['Ike', 'Hal', 'Bob']
>>> random.sample(names, 3)
['Flo', 'Bob', 'Ike']
>>> random.sample(names, 3)
['Ike', 'Ann', 'Hal']
>>>
```

21

Exercise (Notebook)

Develop function `game()` that:

- takes integers `r` and `c` as input,
- generates a field of `r` rows and `c` columns with a bomb at a randomly chosen row and column,
- and then asks users to find the bomb

```
>>> game(2, 3)
Enter next position (format: x y): 0 2
No bomb at position 0 2
Enter next position (format: x y): 1 1
No bomb at position 1 1
Enter next position (format: x y): 0 1
You found the bomb!
```

22

Questions?

24