PROGRAMMING

Lecture 13

Hanbat National University
Dept. of Computer Engineering
Changbeom Choi

OUTLINES

Dictionaries

Named parameters

Files

Formatting

String methods

Reading assignment:

- Chapter 13 of the textbook
- Sections 14.1~14.5 in Chapter 14 of the textbook

DICTIONARIES

Given the name of a country(**key**), how can we know the medal information of this country(**value**)?

Use a dictionary!

A **dictionary is a collection** of items(or **elements**). It is an **object** of type **dict.** Every **item**(or **element**) of a dictionary consists of a **key** and a **value.** The **key** is a **value** of any **immutable** type and used as an index of the item.

```
medals = {} (or medals = dict())
medals = {“United States”: (46, 29, 29),
          “China”: (38, 27, 23),
          “Great Britain”:(29, 17, 19), ... ,
          “Australia”: (7, 16, 12)}

>>>medals[“United States”]
(46, 29, 29)
>>>medals[“Rep. of Korea”]
(13, 8, 7)

>>>medals[1]
KeyError: 1
```

Creating a dictionary

```
txt = (“one”, “two”, “three”, “four”, “five”)
num = (1, 2, 3, 4, 5)
dict1 = {}
for i in range(len(txt)):
   dict1[txt[i]] = num[i]
print (dict1)
```

```
{'four': 4, 'three': 3, 'five': 5, 'two': 2,
'one': 1}
```

Search and change

```
>>>dict1 = {“four”:4, “three”:3, “five”: 5, ”two”:2,
           “one”:1}
>>>dict1[“three”]
3
>>>dict1[“five”]
5
>>>dict1[“one”] = “nice”
>>>dict1
{'four': 4, 'three': 3, 'five': 5, 'two': 2, 'one': ‘nice’}
>>>dict1[“one”] = 1
>>>dict1
{'four': 4, 'three': 3, 'five': 5, 'two': 2, 'one': 1}
```

Add and remove

```
>>>dict1[“nine”] = 9
>>>dict1
{'three': 3, 'one': 1, 'four': 4, 'nine': 9,
 'five': 5}
>>>dict1.pop[“nine”]
>>>dict1
{'three': 3, 'one': 1, 'four': 4, 'five': 5}
```

Traversing a dictionary

```
>>>dict1={'four': 4, 'three': 3, 'five': 5, 'two': 2, 'one': 1}
>>>for key in dict1:
        print (key, dict1[key])
```

```
four 4
one 1
five 5
three 3
two 2
```

Converting to a list

```
>>>dict1 = {“four”: 4, “three”: 3, “five”: 5, “two”: 2, “one”: 1}
>>>lst = dict1.items()
>>>lst
[('four', 4), ('one', 1), ('five', 5), ('three', 3), ('two', 2)]
```

Case study

Given a string, count the number of every character that appears in it, using a dictionary.

```
stg = “maintain”
```

m	a	i	n	t
1	2	2	2	1

```
stg = “maintain”
count = {}
for c in stg:
     if c not in count:
          count[c] = 1
     else:
          count[c] += 1
print (count)
```

```
{'a': 2, 'i': 2, 'm': 1, 't': 1, 'n': 2}
```

Now we are to invert the dictionary. In other words, we want to use the **frequencies** as **keys**.:

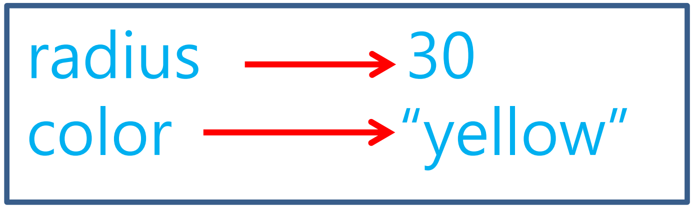
```
    { 1: [“m”, “t”], 2: [a”, “i”, “n”]}
```

```
count = {'a': 2, 'i': 2, 'm': 1, 't': 1, 'n': 2}
inverse = {}
for c in count:
    frequency = count[c]
    if frequency not in inverse:
        inverse[frequency] = [c]
    else:
        inverse[frequency].append(c)
print (inverse)      {1: ['m', 't'], 2: ['a', 'i', 'n']}
```

NAMED PAREMETERS

In general, there is a **positional correspondence** between **parameters** and **arguments**: Arguments are mapped to parameters **one by one** and **left to right**.

```
def create_sun(radius, color):
    sun = Circle(radius)
    sun.setFillColor(color)
    sun.setBorderColor(color)
    sun.moveTo(100, 100)
    return sun
yellow_sun = create_sun(30, “yellow”)
```


radius → 30
color → “yellow”

Default parameters

```
def create_sun(radius = 30, color = “yellow”):
    sun = Circle(radius)
    sun.setFillColor(color)
    sun.setBorderColor(color)
    sun.moveTo(100, 100)
    return sun
```

```
sun  = create_sun()
star = create_sun(2)                    OK !
moon = create_sun(28, “red”)
moon =create_sun(“red”)              Wrong !
```

By using the **names** of **parameters** in a **function call**, **the order of arguments does not matter**.

```
moon = create_sun(color = “red”)
moon = create_sun(color = “red”, radius = 28)

moon = create_sun(color = “red”, 28)    Wrong!
```

FILES

Creating a file

```
f = open(“./planets.txt", “w")
for i in range(8):
     planet = input (“Enter a planet”)
     f.write(planet + “\n”)
f.close()
```

```
f.writelines((“Mercury\n”, “Venus\n”, … ,
                       “Neptune\n”))
```

Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune

Reading from a file

```
>>> f = open(“./planets.txt", "r")
>>> s = f.readline()
>>> s, len(s)
('Mercury\n', 8)
```

f is an object of type file!

line separator

How to get rid of white space?

```
>>>s.strip(),  len(s.strip())
(‘Mercury’, 7)
```

Reading the entire file with a single statement

```
>>>f = open(“./planets.txt", "r")
>>>f.readlines()
['Mercury\n', 'Venus\n', 'Earth\n',
'Mars\n', 'Jupiter\n', 'Saturn\n',
‘Uranus\n', 'Neptune\n']
```

We obtain a **list** with **white space** appearing again!

Reading the entire file line by line

```
f = open(“./planets.txt", "r")
for line in f:
     s = line.strip()
     print (s,end=“ ”)
f.close()
```

for-loop with the **file object** f **calls readline() automatically** at each iteration and stops after reading the last line of the file.

```
Mercury Venus Earth Mars Jupiter Saturn  Uranus Neptune
```

You may also use rstrip. Why?

Finding the line containing “Earth”

```
f = open(“./planets.txt”, “r”)
count = 0
in_file = False
for line in f:
    count += 1
    if line.strip().lower() == “earth”
        in_file = True
        break
if in_file:
    print (“Earth is in line”, str(count) + “.”)
```

Apply lower() to line.strip().

Get out of the loop.

Appending a line

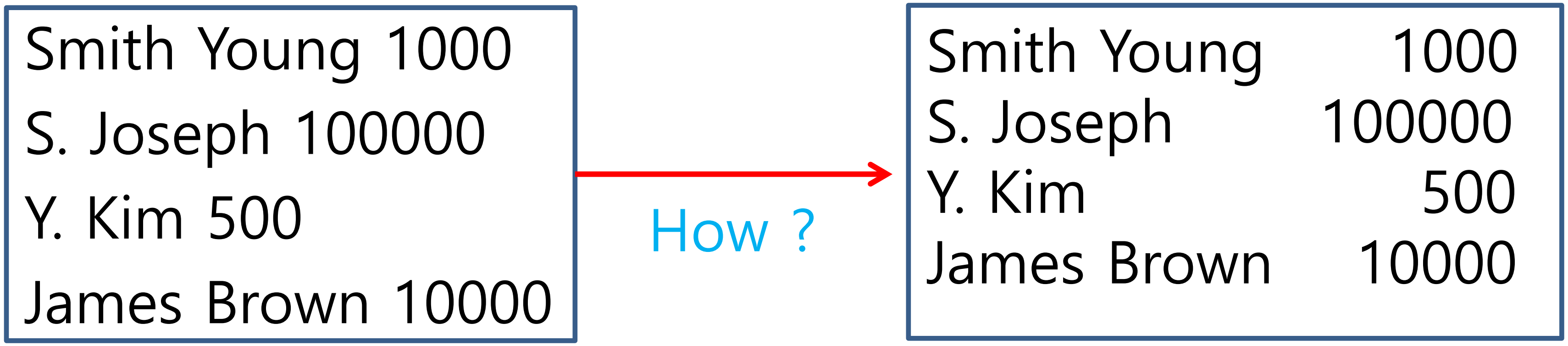
```
>>>f = open(“./planets.txt”, “a”)
>>>f.write(“ Pluto\n”)
```

What if we use “w” instead of “a” ?

FORMATTING

Format string

```
lst = [(“Smith Young”, 1000), (“S. Joseph”, 100000),
        ( “Y. Kim”, 500), (“James Brown, 10000”)]
for name, money in lst:
     print (name, money)
```



```
Smith Young 1000
S. Joseph 100000
Y. Kim 500
James Brown 10000
```

How ?

```
Smith Young        1000
S. Joseph        100000
Y. Kim              500
James Brown       10000
```

```
Smith Young    1000
S. Joseph    100000
Y. Kim          500
James Brown   10000
         11 2     6
```

```
.........................
for name, money in lst:
    print (“%-11s  %6d” %  (name, money))
```

2 spaces

left-aligned

```
x1 = raw_input(“x1 = “)      “60”
x2 = raw_input(“x2 = “)     “150”
val = int(x1) + int(x2)     210
print (str(val) + “ is “ + x1 + “ + “  + x2)
                    210 is 60 + 150
                    val    x1     x2
print (“%d is %s + %s” % (val, x1, x2))
```

place holders

Format operators

```
    format string % (arg0, arg1, …..)
```

Every element in the **tuple** has its corresponding **place holder** in the **format string**.

Place holders

%d	integers in decimal
%s	strings
%g	floats
%f	floats

%.5g # of significant digits is 5
%.5f # of digit after the decimal point

Field width:

```
>>>”%8.3f  %8.3g” %  ( 123.456789, 123.456789)
‘ 123.457       123’
>>>name    = “Joseph S. Shin”
>>>amount = 100000
>>>”%20s  spent  % 10d  for shopping.”  %  (name, amount)
‘      Joseph S. Shin  spent      100000  for shopping.’
>>>”%-20s  spent  % -10d  for shopping.” %  (name, amount)
‘Joseph S. Shin        spent  100000    for shopping.’
>>>”My name is %-15s .”  %  name
‘My name is Joseph S. Shin .’
```

STRING METHODS

A string is an **immutable sequence** of characters.

in operator

```
>>>”abc” in “01234abcdefg”
True
>>>”fgh” in “01234abceefg”
False
```

A bit different from the usage for tuples & lists

String objects have many useful methods:

- upper(), lower(), and capitalize()
- isalpha() and isdigit()
- startswith(prefix) and endswith(suffix)
- find(str), find(str, start), and find(str, start, end)
  (return -1 if str is not in the string)
- replace(str1, str2)
- rstrip(), lstrip() and strip() to remove
  white space on the right, left, or both ends.

```
>>> "joseph is My name".capitalize()
‘Joseph is my name’
>>>”12345”.isdigit()
True
>>>”This book is mine”.startswith(”this”)
False
>>>”This book is mine”.find(“book”)
5
>>>”This book is mine. That is also mine”.replace(“mine”, “yours”)
'This book is yours. That is also yours'
```

String methods for converting between list and string

split() to split with **white space** as separators

split(sep) to split with a **given separator** sep

join(lt) to create a **string** by concatenating its elements

```
>>> "I like\rthis\ncourse\tvery much.".split()
['I', 'like', 'this', 'course', 'very', 'much.']
>>>lt = ['I', 'like', 'this', 'course', 'very',
         'much.']
>>>” “.join(lt)
‘I like this course very much.’
```