PROGRAMMING

Lecture 12

Hanbat National University
Dept. of Computer Engineering
Changbeom Choi

```
>>>tup = (“a”,  [1, 2])
>>>tup[1]
[1, 2]
>>>tup[1][1] = 3
>>>tup[1]
[1, 3]
>>>tup[1] = [1, 2]
```

OUTLINE

2010 Winter Olympic

2012 Memento game

2010 WINTER OLYMPIC

PROBLEM 1: USING A SINGLE LIST FOR MEDALS

In the previous lecture, we used four lists to store the country names and their acquired medals. A more effective way is to store the entire medal information including country names and the numbers of their acquired medals in a single list as shown below:

```
medals = [(“United States”, 46, 29, 29),
          (“China”, 38, 27, 23),  ... ,
          (“Australia”, 7, 16, 12)]
```

(Continued)

Unlike in the previous lecture, you should use the medal information for the 2010 Winter Olympic, which is available in a web site,

https://en.wikipedia.org/wiki/2010_Winter_Olympics_medal_table

(2010 Winter Olympics medal table)

Using a single list, develop a program to perform the following tasks:

(Continued)

1. Compute and print the total number of medals for each county as follows:

Countries	total number of medals
United states	37
Germany	30
Canada	26
.....	.....

Main program

medals = create_list()

print_list(medals)

Functions:

def create_list():

- visit the web site for the medal information.
- refer to the last lecture to construct a list.

def print_list(mdals):

- Compute the total number of medals for each country before printing it.
- Refer to the last lecture.

2. Sort the countries in the decreasing order of **their total numbers** of **acquired medals** and print a list that shows the country names together with their total numbers of medals. To sort the list, you should not use any other list or tuple except the one stated in the previous slide.

Main program

medals = create_list() ⟵ Use a function developed in 1-1.

medals = sort_list (medals)

print_list(medals) ⟵ Use a function developed in 1-1.

```
import functools
def sort_list(medals):
    medals.sort(key=functools.cmp_to_key(compare))
    return mdals
def compare(item1, item2):
    medals1 = sum(item1[1:])
    medals2 = sum(item2[1:])
    return cmp(medals1, medals2)

def cmp(a, b):
    if a < b:
        return 1
    elif a > 1:
        return -1
    else:
        return 0
```

The method, sort uses a function name as a parameter.

Computing the total numbers of medals
(nation, g, s, b)

The result is 1, -1 or 0.

3. Prepare and print the following table and plot the table as shown in the next slide:

int	# of medals	# of countries
0	0-2	4
1	3-5	8
2	6-8	3
3	9-11	4
4	12-14	1
	.....	.....
	.....	.....
	.....	.....
11	33-35	0
12	36-38	1

Main program

```
medals = create_list()
table = build_table(medals)
plot_table(table)
```

Building a table: 13 intervals

```
def build_table(medals):
    t = [0] * 13        <-- [0,0,..., 0]
    for item in medals:
        total = sum(item[1:])
        t[total//3] += 1
    return t
```

```
0~2:     ****
3~5:     ********
6~8:     ***
9~11:     ****
12~14:   *
15~17:   **
18~20:
21~23:   *
24~26:   *
27~29:
30~32:   *
33~35:
36~38:   *
```

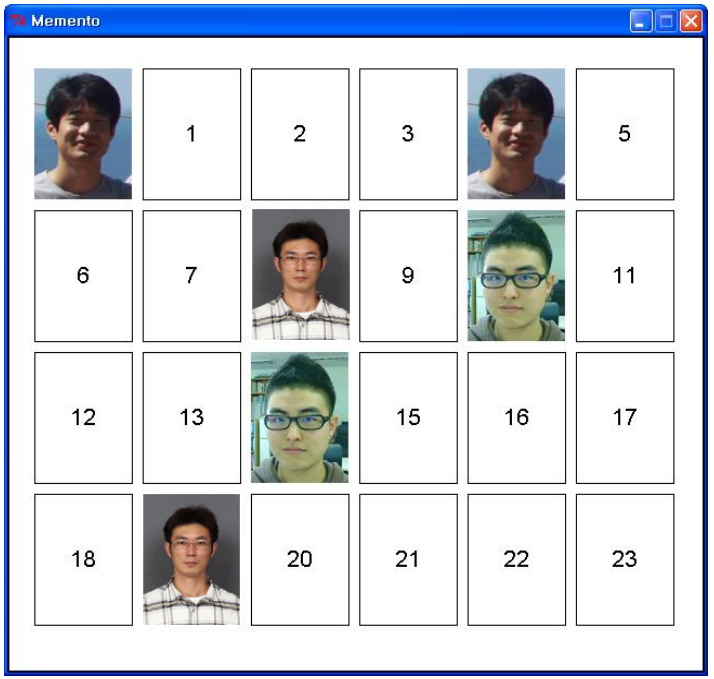
Plotting the table

```
def plot_table(t):
    for i in range(13):
       print (str(3*i) + “~” + str(3*i +2)
          + “:\t”  + “*” *  t[i])
```

MEMENTO GAME

PROBLEM 2: MEMENTO GAME

Suppose that you work for a game company. Your boss asked you to develop a **memento game.** The game rule is simple. You have 24 cards on a **6 x 4 board** as shown on the left. Every card has a picture on its face. The picture of a card is shared by exactly one other card. Therefore, there are 12 pairs with each pair sharing the same picture. All cards have their own identification numbers ranging from 0 to 23.



A number shown on the back of a hidden card indicates the position of the card on the board. At the beginning, all cards are hidden. At each round, a player is allowed to open exactly two hidden cards. If they have the same photo, the both cards will remain open. Otherwise, the player will flip down them on the board to hide the photos on their faces. The game will be over when all cards are open. The objective of the game is to minimize the number of rounds until the end of the game. In your hand-out, a program for playing the Memento game is given. **Your tasks are two-fold: (1) to type in the code in the hand-out to obtain you own working program, and (2) to put comments so that the code is readable.**

Note: For test photo files, please contact TAs.

Pseudo code

1. Initialize 24 cards and a 4 x 6 board
2. While there are hidden cards on the board:

2.1 Open a pair of hidden cards and count the number of rounds

2.2 If they have the same photo on the face, **then** count the number of correct pairs and leave them open on the board.
Otherwise, flip down the both cards on the board to hide the photos on their faces.

3. Show the number of rounds.

Data structures

- Maintaining **the 12 photos** that are shown on the cards.

  photos = [photo file name, ……..]

- Maintaining the **entire 24 cards**. Each card has a **photo identification.**

  cards = [photo_id , …….]

- Maintaining the **status** of **every card** on the 6 X 4 board.

  board = [(photo_id, status), ……..]

  hidden or open

  index of photo

#Main Program

main()

```
def main():
    from cs1graphics import *
    import random, time

    canvas = Canvas(540, 480)
    canvas.setBackgroundColor("light blue")
    canvas.setTitle("Memento Games")

    photos = []
    cards = []
    board = []
```

```
correct_pairs = 0
round = 0

layer = Layer()
layer.setDepth(40)
canvas.add(layer)

initialize_cards_and_board()
show_initial_screen()
```

```
while(correct_pairs < 12):
    card1 = get_a_card(1)
    card2 = get_a_card(2)
    if card1 == card2:
        print (card1, card2, ": You choose the same card! Retry.")
    elif is_valid(card1, card2):
        if check_cards(card1, card2, layer):
            print ("Great! You got it.")
            correct_pairs += 1
        else:
            print ("Sorry! Try again.")
    else:
        print (card1, "or", card2, " is invalid.")
    round += 1
    print ("the " + str(round) + round_suffix(round) + " round"+ ": ",
       correct_pairs, "correct pairs")
print ("You are done! Your number of trials is ", round, ".")
```

step 2

step 2.1

step 2.2

step 3

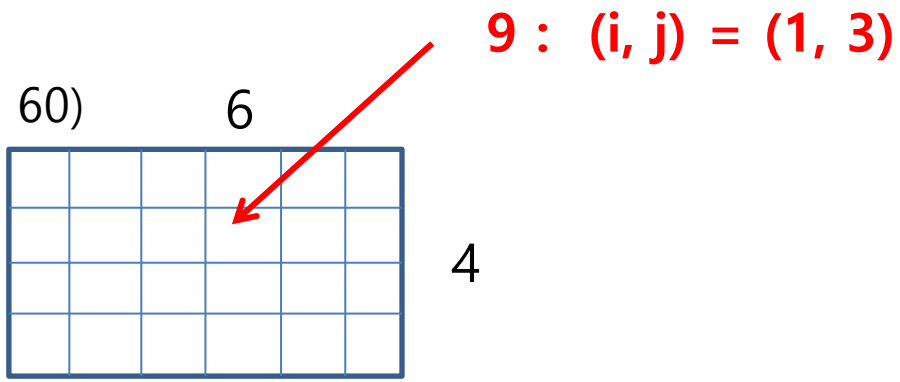
```
def initialize_cards_and_board():
    for photo_id in range(12):
        if photo_id < 10:
            filename = "faces/face0" + str(photo_id) + ".jpg"
        else:
            filename = "faces/face" + str(photo_id)  + ".jpg"
        photos.append(filename)
        cards.append(photo_id)
        cards.append(photo_id)
    random.shuffle(cards)
    for i in range(24):
        hidden = True
        board.append([cards[i], hidden])
```

photos = [photo file name, ...]

cards = [photo_id, photo_id, ...]

board = [(photo_id, status), ...]

```
def show_initial_screen():
    x0 = 50
    y0 = 60
    for index in range(24):
        i = index // 6
        j = index % 6
        rect = Rectangle(90,120)
        rect.moveTo(x0 + 90 * j, y0 + 120 * i)
        canvas.add(rect)
        label = Text(str(index))
        label.moveTo(x0 + 10 +  90 * j, y0 + 120 * i)
        canvas.add(label)
```



```
def get_a_card(new):
    while(True):
        if  new == 1:
            card = input("Enter the 1st card number: ")
        else:
            card = input("Enter the 2nd card number: ")
        try:
            card = int(card)
            return card
        except:
            print (card, ": invalid input“)
```

```
def is_valid(card1, card2):
    if card1 < 0 or card1 > 23:
        return False
    if card2 < 0 or card2 > 23:
        return False
    if is_hidden(card1) and is_hidden(card2):
        return True
    return False

def is_hidden(card):
    photo_id, hidden = board[card]
    return hidden
```

```
def check_cards(card1, card2, layer):
    photo_id1, hidden1 = board[card1]
    photo1 = add_to_layer(card1, photo_id1, layer)
    photo_id2, hidden2 = board[card2]
    photo2 = add_to_layer(card2, photo_id2, layer)
    time.sleep(3)
    if photo_id1 == photo_id2:
        update_board(card1)
        update_board(card2)
        return True
    layer.remove(photo1)
    layer.remove(photo2)
    return False
```

```
def add_to_layer(card, photo_id, layer):
    x0 = 50
    y0 = 60
    i = card // 6
    j = card % 6
    filename = photos[photo_id]
    photo = Image(filename)
    photo.moveTo(x0 + 90*j, y0 + 120 * i)
    layer.add(photo)
    return photo

def update_board(card):
    board[card][1] = False
```