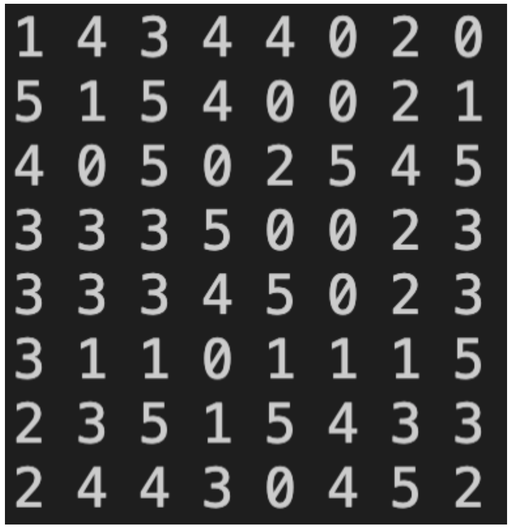


CSC 1002 Week 5

Flipping Number – Scope, Spec, Design, Implementation

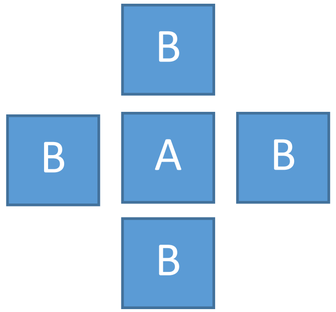
Number-Flipping Game

Start

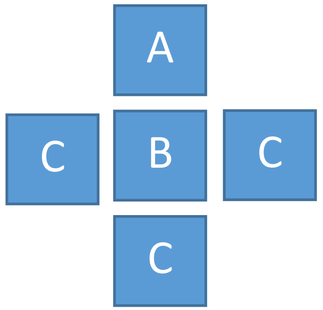
End

Connected Neighbors

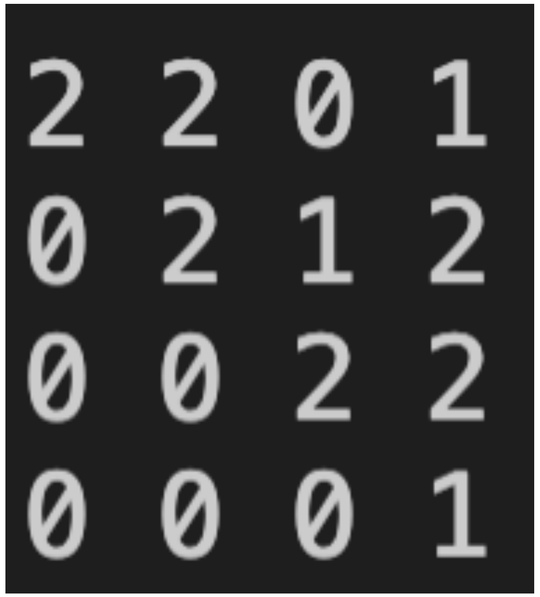
- Two positions (A,B) are connected if the digits are identical and position of B is in one of the A's direct left, right, up or down position.



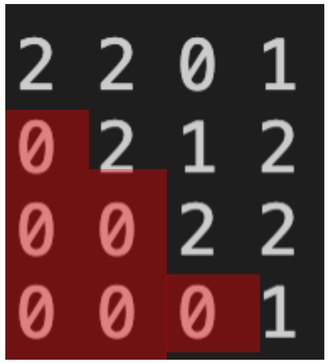
- Two positions (A,C) are connected if digits are identical and A and C are connected via other connected neighbors.



Examples


2 2 0 1
0 2 1 2
0 0 2 2
0 0 0 1


2 2 0 1
0 2 1 2
0 0 2 2
0 0 0 1

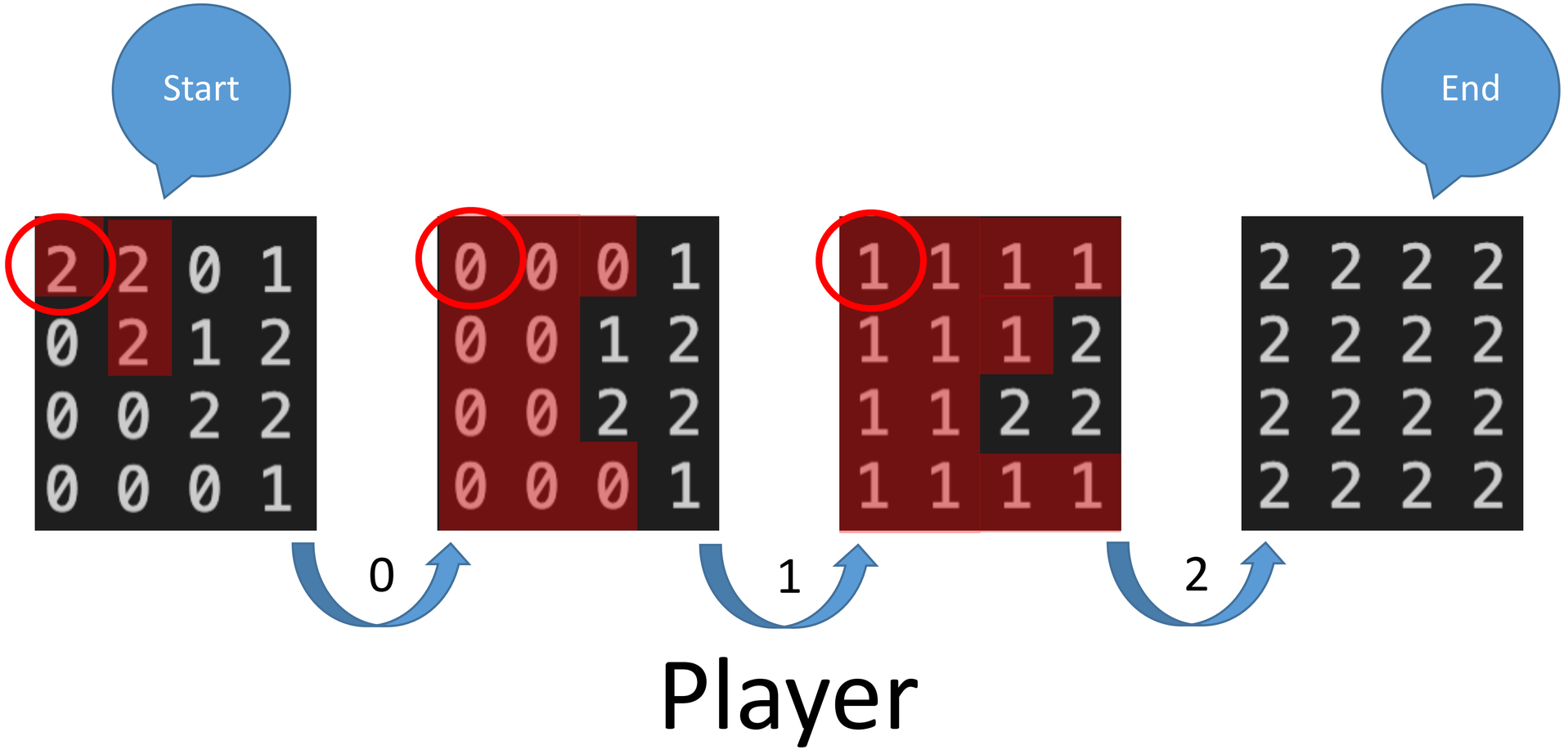

2 2 0 1
0 2 1 2
0 0 2 2
0 0 0 1


2 2 0 1
0 2 1 2
0 0 2 2
0 0 0 1

Number-Flipping Game (Console, Keyboard Based)

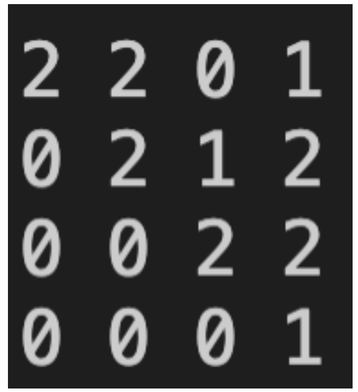
- A rectangular board of an NxN array of randomly chosen, single-digit numbers (0-9).
- Goal: change all the numbers (digits) to be same
- Rule:
  - Player chooses one digit to replace the digit at the upper left corner position
  - The changes will spread to all its connected neighbors
  - After the change is completed you are prompted for another digit.
  - The process repeats until all digits are identical

Demo (4x4) with digits 0, 1 & 2



Show a demo

Design – Data Model



- How do we model the game board ?

String

"2201021200220001"

Flatted List

[2, 2, 0, 1, 0, 2, 1, 2, 0, 0, 2, 2, 0, 0, 0, 1]

Nested List

[[2, 2, 0, 1], [0, 2, 1, 2], [0, 0, 2, 2], [0, 0, 0, 1]]

Dictionary

[0:[2, 2, 0, 1], 1:[0, 2, 1, 2], 2:[0, 0, 2, 2], 3:[0, 0, 0, 1]]

Data Model - Quick Comparison

Data Model	Remark
String	Need to convert into rows and columns; therefore another pcs of info needed (dimensions)
Flat List	Same as string.
Nested List	The number of elements in the list will match the row count, and the length of each sub-element is the column count. The dimension is implicit.
Dictionary	The number of keys in the dictionary will match the row count, and the length of the value is the column count. The dimension is implicit.

Design - Functionality

- Create new Game
- Refresh the screen (display the game on the screen)
- Prompt player for a number (single-digit)
- Flip neighboring digits to the new digit
- Check completion of the game

Design – Function Parameters

- Create new Game - createGame(dim, range) returns the data model (game)
- Refresh screen with the Game – refreshScreen(game)
- Prompt player for a digit to flip to – promptNumber() returns the digit
- Flip neighboring digits to the new digit - flipNumber(row, col, game, from, to) returns the updated data model
- Check if game is completed, all digits identical – checkGame(game) returns a Boolean, True or False

create_game(dim, num_range) -> model

- It creates the data model (game) in given dimension (dim x dim) and fill in the data model with random digits chosen based on the given number range (from 0 to num_range, inclusive)

- create_game(5,3)

```
1 0 3 0 1
0 1 0 0 1
3 1 1 2 3
2 3 2 0 0
1 0 1 1 2
```

create_game(5,5)

```
1 0 2 1 2
4 1 4 0 4
0 5 1 1 0
5 4 5 2 3
0 2 5 4 0
```

create_game(5,1)

```
0 0 0 0 0
0 1 0 1 1
0 0 1 1 1
0 0 0 1 0
1 0 1 0 1
```

refresh_screen(game) -> None

- Display the game on the console in rows and columns according to digits in the given data model (game), separating each digit in the same row with a single whitespace.

- refresh_screen(game)

```
1 0 2 1 2
4 1 4 0 4
0 5 1 1 0
5 4 5 2 3
0 2 5 4 0
```

prompt_player() -> the entered digit

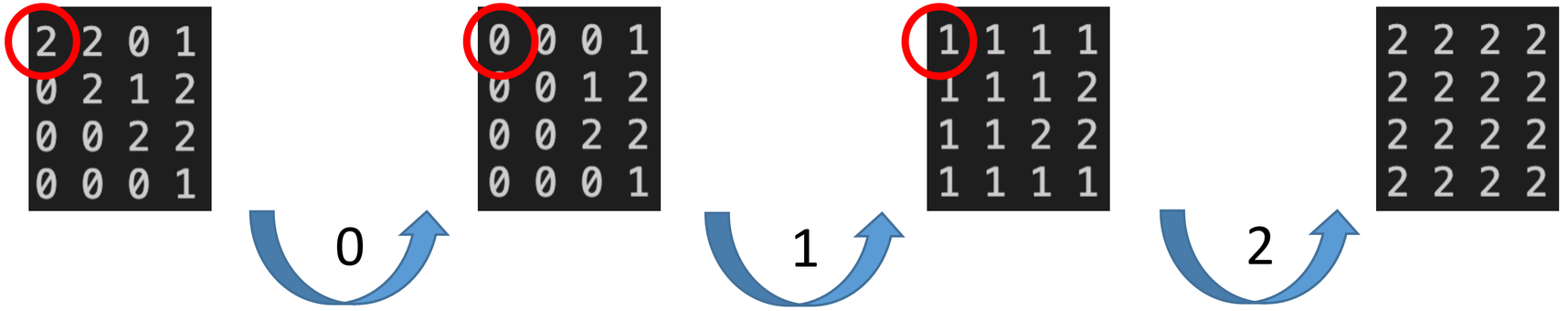
- It displays a prompt message on the screen, prompting the player to enter a digit (from 0 to 9).

```
0 0 0 0 0
0 1 0 1 1
0 0 1 1 1
0 0 0 1 0
1 0 1 0 1
Enter number to flip to?
```

flip_number(row, col, game, orig, to) -> None

- Parameters:
  - row, col is the starting position to start number-flipping process.
  - orig, to are the original and target digits respectively
- It will replace all the neighboring digits (orig) with the newly selected digit (to), starting at the given position (row,col).

- flipNumber(0,0,game,2,0) flipNumber(0,0,game,0,1) flipNumber(0,0,game,1,2)



check_game(game) -> boolean

- It checks if all the digits from the given data model are identical.

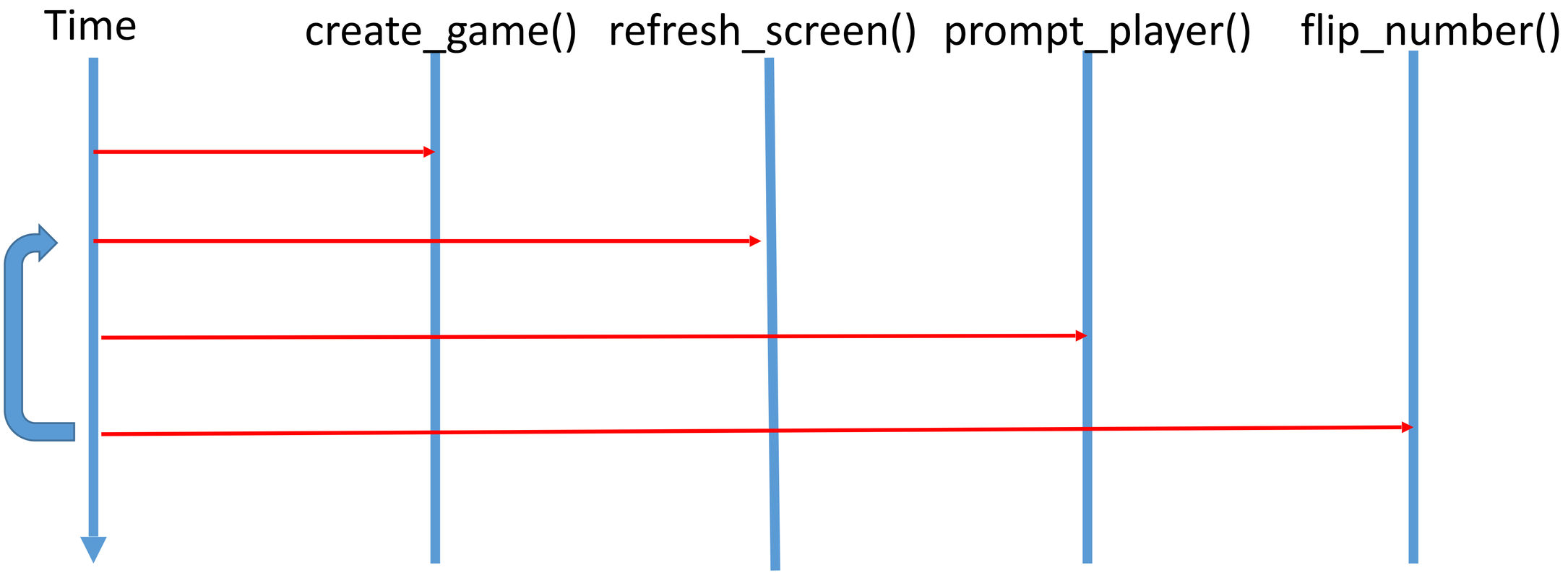
- check_game(game) = True

```
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

check_game(game) = False

```
2 2 0 1
0 2 1 2
0 0 2 2
0 0 0 1
```

Design – Sequence Diagram (Single Game)



Sequence Diagram: interaction diagram detailing how operations are carried out, in our case, what functions are called and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The functions involved in the operation are listed on the top, the red lines show the function invocation according to when they take part in the execution sequence.

Design – Main Processing Logic

```
game = create_game(dim, num_range)
while True:
        display_game(game)
        to = prompt_player(game)
        flip_number(0, 0, game, from, to)
```

Implementation

create_game(dim, num_range)

```
def create_game(dim, digit_range):
    b = []
    for _ in range(dim*dim):
        b.append( random.randint(0,digit_range) )
    return b
```

refresh_screen(game)

```
def refresh_screen(game):
    print('')
    for r in range(g_dim):
        for c in range(g_dim):
            print(game[r*g_dim+c], end=' ')
        print('')
```

flip_number(row, col, game, orig, to)

```
30  def flip_number(row, col, game, orig, to):
31
32      # assert that we got 2 different values to flip
33      if orig == to:
34          return
35
36      # assure both row and column within boundaries
37      if row < 0 or row >= g_dim:
38          return
39      if col < 0 or col >= g_dim:
40          return
41
42      # get the cell position
43      idx = row*g_dim+col
44
45      # check for connected cell
46      if game[idx] != orig:
47          return
48
49      # flip the cell to the given value "to"
50      game[idx] = to
51
52      # recursion calls
53      # flip other cells in up, down, left and right directions.
54      flip_number(row-1, col, game, orig, to)
55      flip_number(row+1, col, game, orig, to)
56      flip_number(row, col-1, game, orig, to)
57      flip_number(row, col+1, game, orig, to)
58
59      return
```

prompt_player()

```
def prompt_player():
    while True:
        n = input('Enter number to flip to?')
        if n.isdigit() and len(n) == 1:
            break
    return int(n)
```

Main Body

```
g_game = create_game(g_dim, g_digit_range)
while True:
    refresh_screen(g_game)
    n = prompt_player()
    flip_number(0, 0, g_game, g_game[0], n)
```

Recursion – quick overview

Recursion provides a clean and simple way to write code

Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function
- Recursion provides a clean and simple way to write code

Reverse a string

- reverseString(“hello world”) returns “dlrow olleh”

```
def reverseString(s):
    if len(s) < 2:
        return s
    else:
        return s[-1] + reverseString(s[:-1])
```

```
def reverseString(s):
    return s[-1] + reverseString(s[:-1]) if len(s) > 1 else s
```

summation - a simple list of numbers

- Given [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and return 55

```
def sumList(s):
    if len(s) == 1:
        return s[0]
    else:
        return s[0] + sumList(s[1:])
```

```
def sumList(s):
    return s[0] + sumList(s[1:]) if len(s) > 1 else s[0]
```

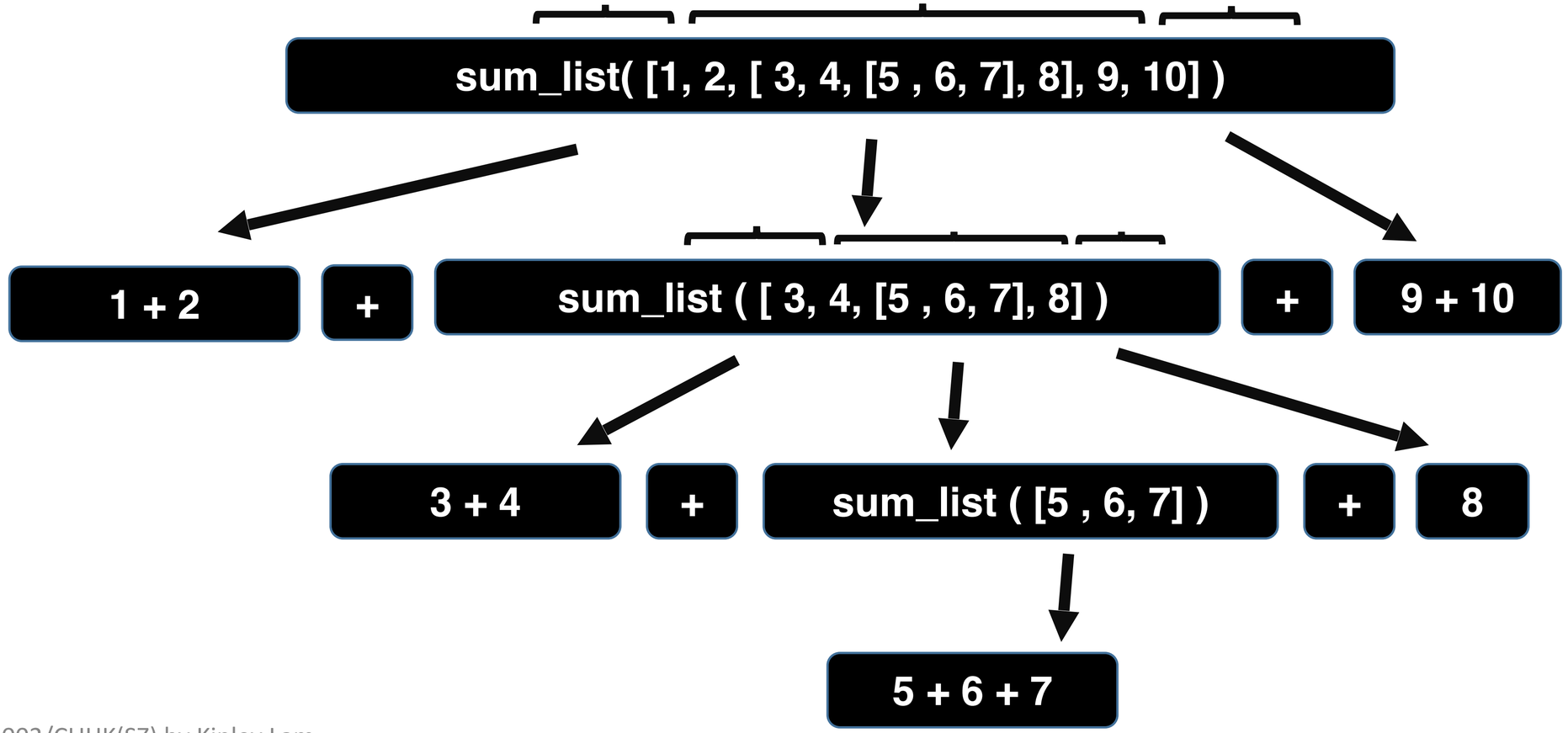
summation - a nested list of numbers

- Given [1, [2, 3], 4, [5, [6, 7], 8, 9], 10] and return 55

```
def sumNestedList(s):
    if type(s) is int:
        return s
    elif len(s) == 1:
        return sumNestedList(s[0])
    else:
        return sumNestedList(s[0]) + sumNestedList(s[1:])
```

summation - nested list of integers

- Given [1, 2, [3, 4, [5 , 6, 7], 8], 9, 10] and return 55



summation - non-recursion version

```
22    def sum_list(listitem):
23        tot = 0
24        for x in listitem:
25            if type(x) is int:
26                tot += x
27            elif type(x) is list:
28                tot += sum_list(x)
29        return tot
```

The End – Thank You