Review

- Module
  - Importing an entire module vs. specific functions/variables
  - Memory vs. namespace

- Class vs. Class object

- Class method

Lists – Basics

Lecture 5-1

Hyung-Sin Kim

SNU Graduate School of Data Science

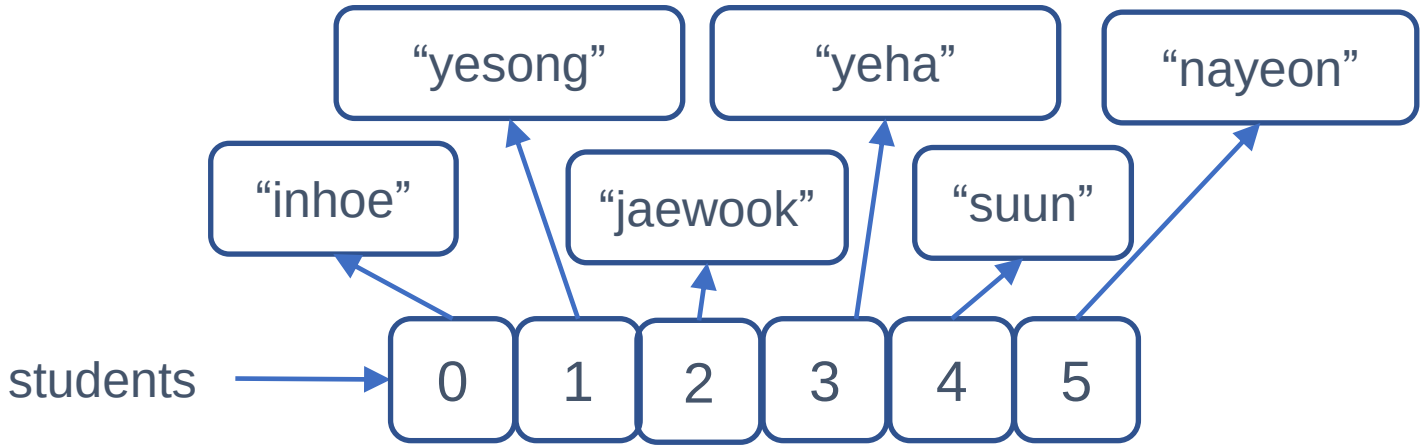
Lists

- List is a type of object (i.e., **class**) that contains a list of **ordered** items
  - [<<expression1>>, <<expression2>>, …, <<expressionN>>]
  - [] (empty list)
  - List, as a class, has its own **methods**
- A list **object** can be assigned to a variable

- Example (tens of students in this course)
  - I need to declare so many variables! (nightmare!)
    - >>> student1 = “inhoe”
    - >>> student2 = “yesong”
    - >>> student3 = “jaewook”
  - Instead, I need to declare one list, much easier to manage
    - >>> students = [“inhoe”, “yesong”, “jaewook”, “yeha”, “suun”, “nayeon”]

Lists

- Memory model



Access and Assign

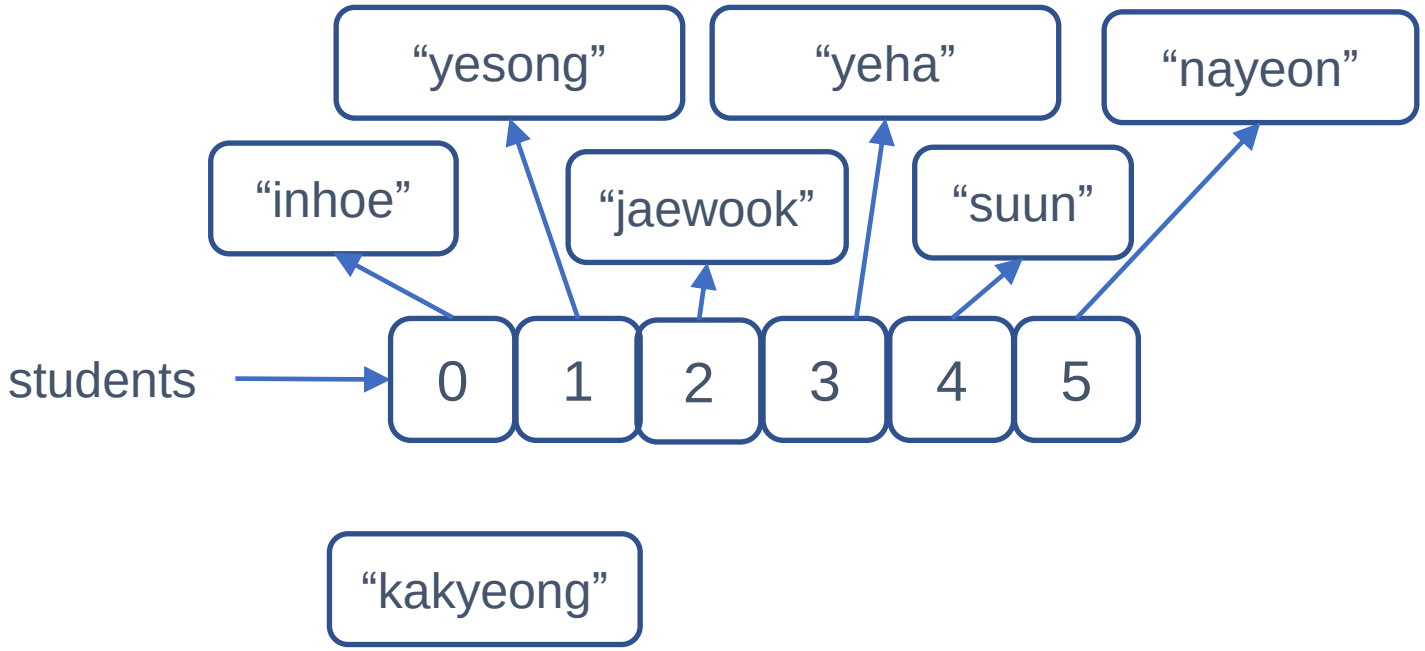
- students = [“inhoe”, “yesong”, “jaewook”, “yeha”, “suun”, “nayeon”]

- Access elements of the list
  - >>> students[0] ➡ “inhoe”
  - >>> students[5] ➡“nayeon”
  - >>> students[-1] ➡ “nayeon”
  - >>> students[-3] ➡“yeha”

- Assign elements to variables
  - >>> student0 = students[0]
  - >>> print(student0) ➡ inhoe

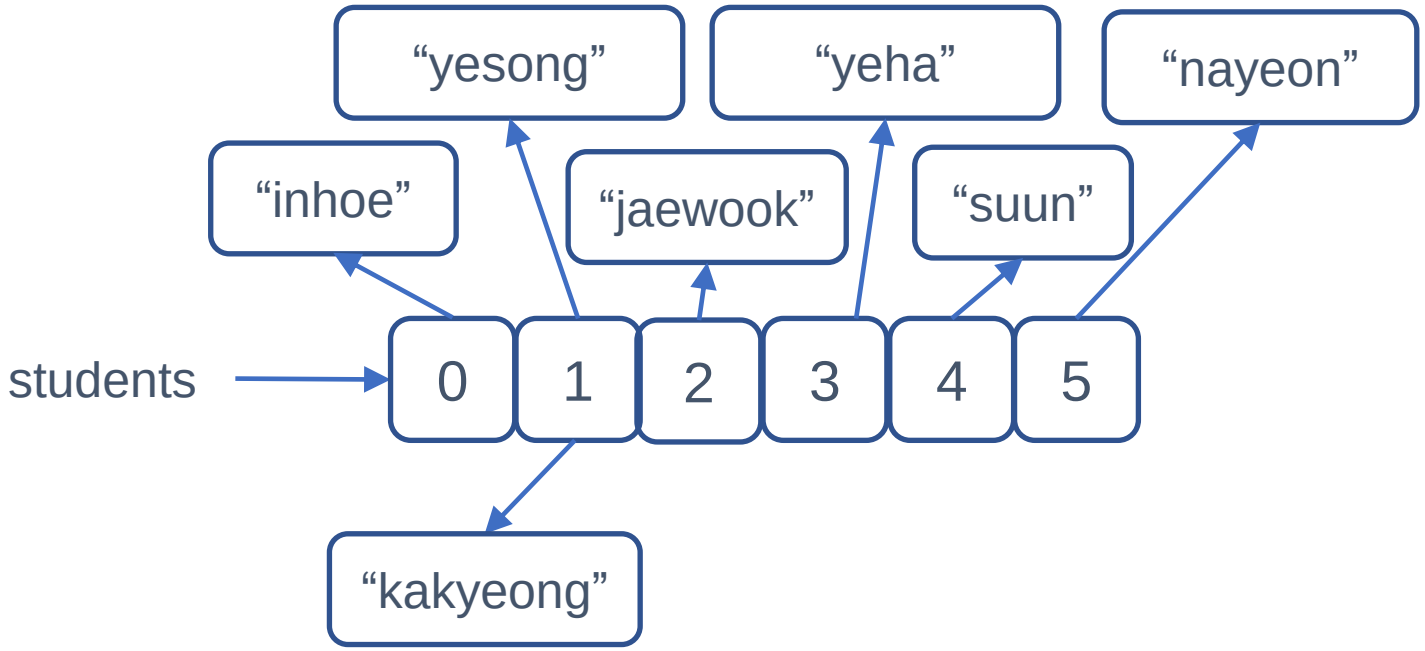
Modifying Elements

- Elements can be modified, just as variables
  - >>> students[1] = “kakyeong”



Modifying Elements

- Elements can be modified, just as variables
  - >>> students[1] = “kakyeong”
  - >>> students ➡ [“inhoe”, “kakyeong”, “jaewook”, “yeha”, “suun”, “nayeon”]



Type

- Lists can contain **any type** of data
  - >>> jaesuk_info = [“MC”, “1972.8.14”, 178, 65]

출생 1972. 8. 14. 서울특별시, 사자자리, 쥐띠
나이 51세, 만49세
소속그룹 싹쓰리
소속사 안테나
신체 178cm, 65kg
가족 배우자 나경은
데뷔 1991년 제1회 KBS 대학개그제
종교 불교

  - But this is **error prone**, since we need to remember what is where
  - Lists are <u>usually</u> used for containing a **single type** of objects
- Recommendation: Specify what type a list expects
  - >>> from typing import List
  - >>> def average(L: List[float]) -> float:
  - … <<body>>

List of Lists

- List can have lists as its **elements**
  - >>> students = [[“2021-11111”, “inhoe”], [“2021-22222”, “yesong”], [“2021-33333”, “jaewook”], [“2021-44444”, “yeha”], [“2021-55555”, “sun”], [“2021-55555”, “nayeon”]]
  - >>> students[0] ⟹ [“2021-11111”, “inhoe”]
  - >>> students[1][0] ⟹ “2021-22222”
  - >>> students[1][1] ⟹ “yesong”

- We can assign a **sublist** to a variable (creating an **alias** for that sublist)
  - Any change to the sublist alias will change the sublist, which will also be seen when accessing the main list

Operations

- A = [2, 5, -1, 4, 3, 3, 100, -20]
  - >>> len(A) ➡ 8
  - >>> max(A) ➡ 100
  - >>> min(A) ➡ -20
  - >>> sum(A) ➡ 96
  - >>> sorted(A) ➡ [-20, -1, 2, 3, 3, 4, 5, 100]
  - >>> A + [2, 3, 5] ➡ [2, 5, -1, 4, 3, 3, 100, -20, 2, 3, 5]
  - >>> A * 2 ➡ [2, 5, -1, 4, 3, 3, 100, -20, 2, 5, -1, 4, 3, 3, 100, -20]
  - >>> del A[1] \ A ➡ [2, -1, 4, 3, 3, 100, -20]

Operations – “in” Operator

- **A** in **List-B**
  - **True** if **A** is an element of **List-B**

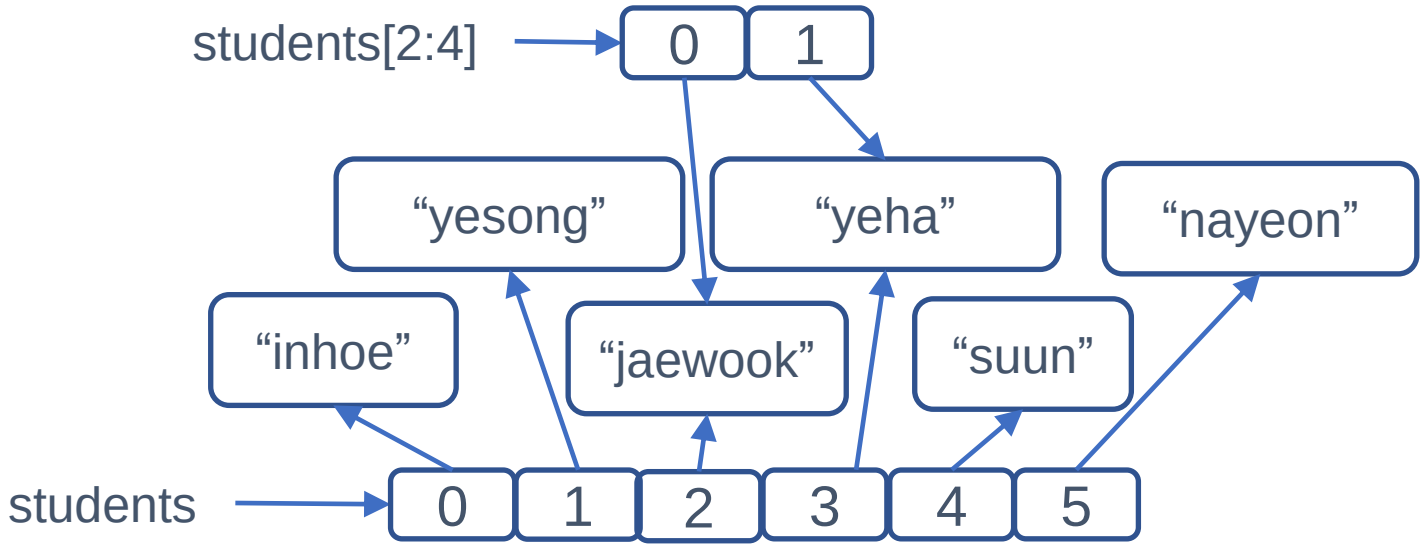
- attendance = [“inhoe”, “yesong”, “jaewook”, “yeha”, “suun”]
  - >>> “jaewook” **in** attendance
  - True
  - >>> “nayeon” **in** attendance
  - False

Methods

- List is also a **class** having several methods, such as str
  - These methods modify the list but does not return anything (None)
  - >>> students.append(“kangsuk”)
  - >>> students.clear()
  - >>> students.count(“suun”)
  - >>> students.index(“jaewook”)
  - >>> students.insert(2, “sunwoo”)
  - >>> students.pop()
  - >>> students.remove(“inhoe”)
  - >>> students.reverse()
  - >>> students.sort()
  - >>> students.sort(reverse=True)

Slicing

- A[i:j]: A list comprised of i-th element to (j-1)-th element of list A
  - students = [“inhoe”, “yesong”, “jaewook”, “yeha”, “suun”, “nayeon”]
  - students[2:4] ➡ [“jaewook”, “yeha”]
  - students[3:] ➡ [“yeha”, “suun”, “nayeon”]
  - students[:2] ➡ [“inhoe”, “yesong”]
  - students[:] ➡ [“inhoe”, “yesong”, “jaewook”, “yeha”, “suun”, “nayeon”]



Summary

- List definition
- Access, Assign, and Modify list elements
- List of lists
- Operations and Methods
- Slicing

Lists – Copy and Alias

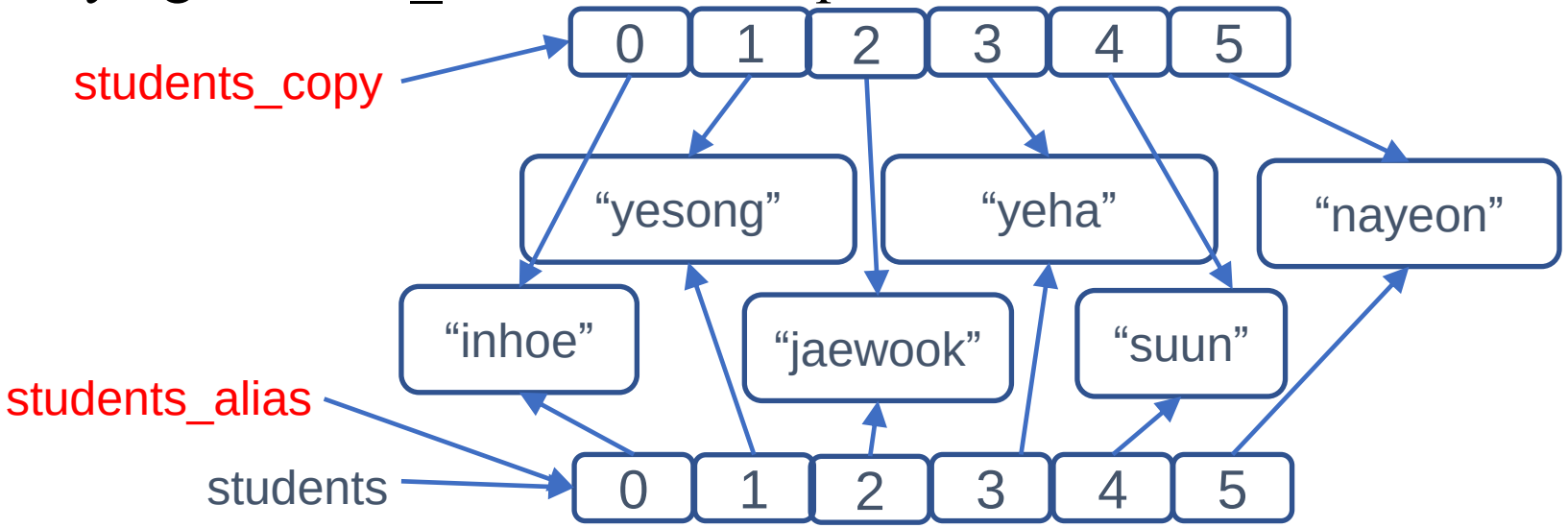
Lecture 5-2

Hyung-Sin Kim

SNU Graduate School of Data Science

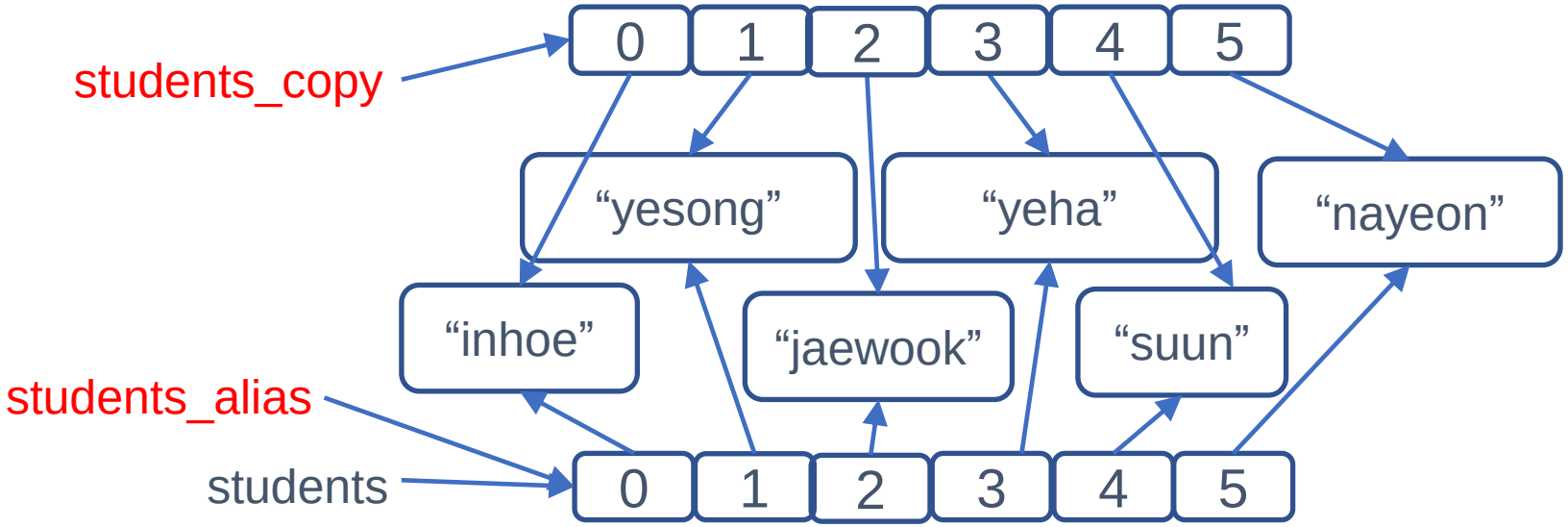
Copy vs. Alias

- **Copy**: students_copy = students[:]
  - An **independent** list with the same elements
  - Modifying *students_copy* does **NOT** impact *students*
- **Alias**: students_alias = students (having many alias is not a good idea!)
  - Another name referring to the **same** list
  - Modifying students_alias **DOES** impact *students*



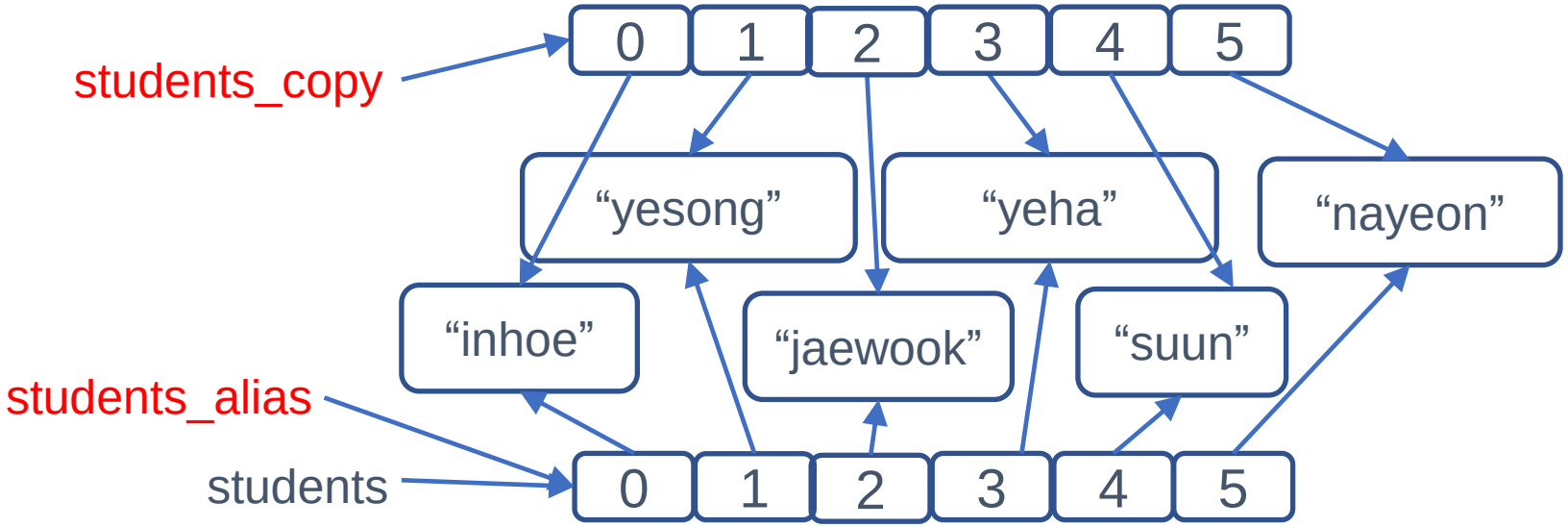
Copy vs. Alias

- del students_copy[-1]
  - *students* is not changed at all
- del students_alias[-1]
  - stduents[-1] is now removed!



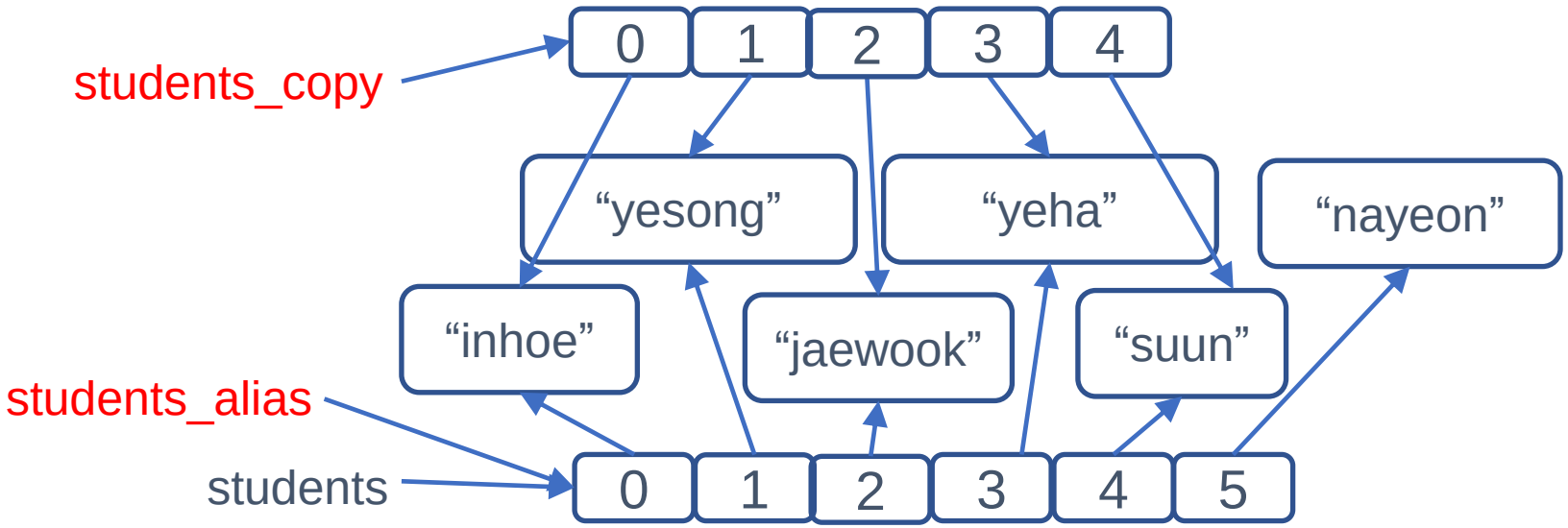
Copy vs. Alias

- del students_copy[-1]



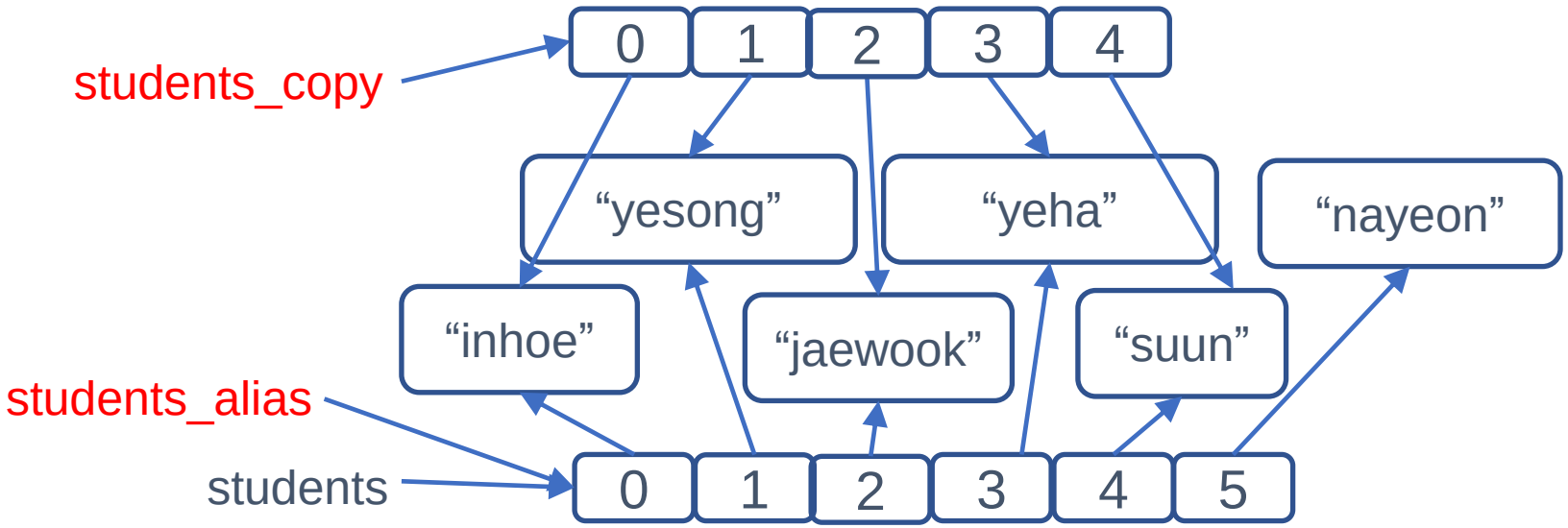
Copy vs. Alias

- del students_copy[-1]
  - *students* is not changed at all



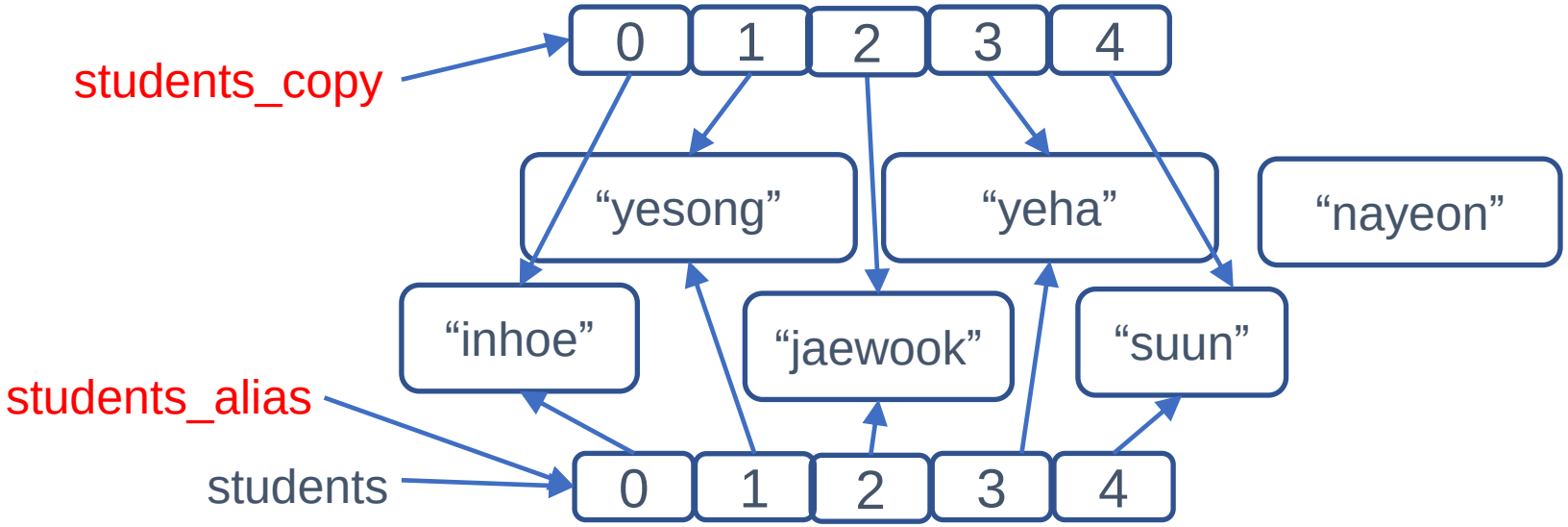
Copy vs. Alias

- del students_copy[-1]
  - *students* is not changed at all
- del students_alias[-1]



Copy vs. Alias

- del students_copy[-1]
  - *students* is not changed at all
- del students_alias[-1]
  - stduents[-1] is now removed!



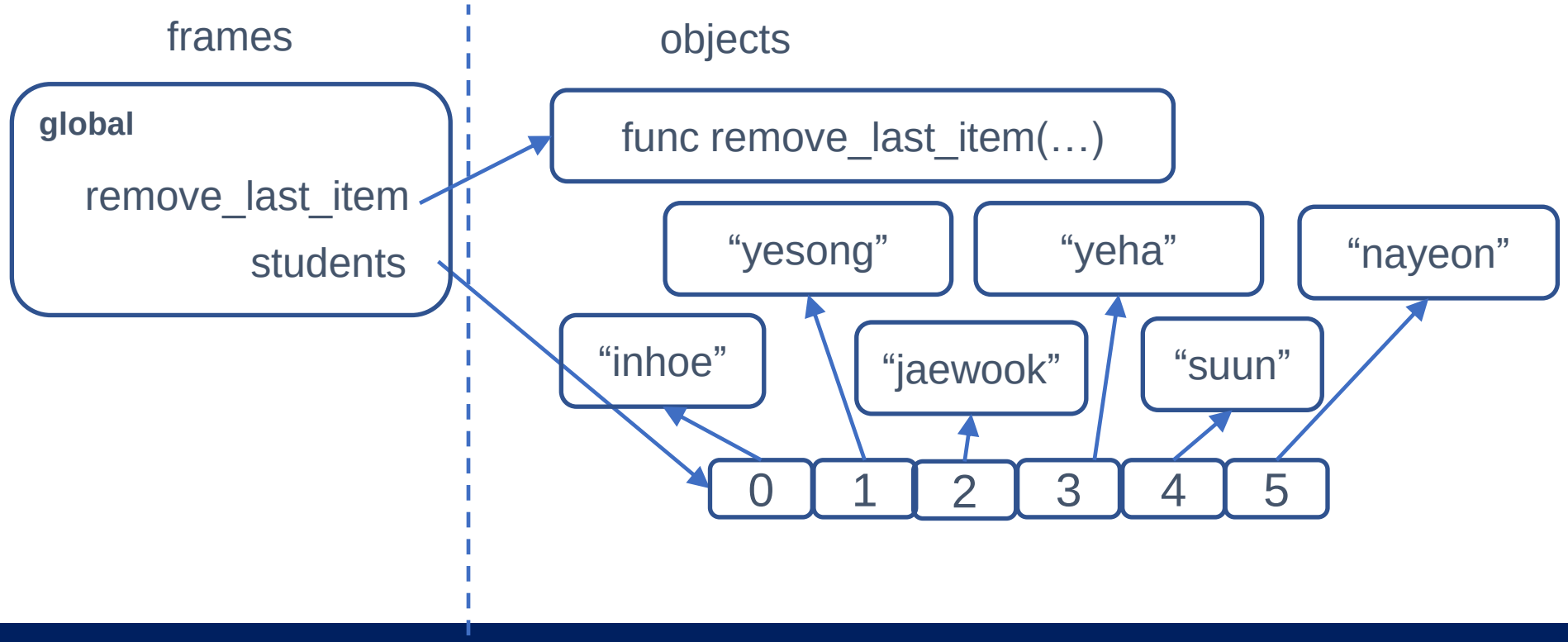
Parameter as Alias

- Since function parameters are **variables**, list parameters cause **aliasing**
  - >>> def remove_last_item(**L**: list) -> list:
  - >>> if len(L) > 0:
  - >>> del L[-1]
  - >>> else:
  - >>> print("The list is empty.")
  - >>> remove_last_item(**students**)

- The function does not have **return**, but *students* is changed since list *L* in the function is *students'* **alias**

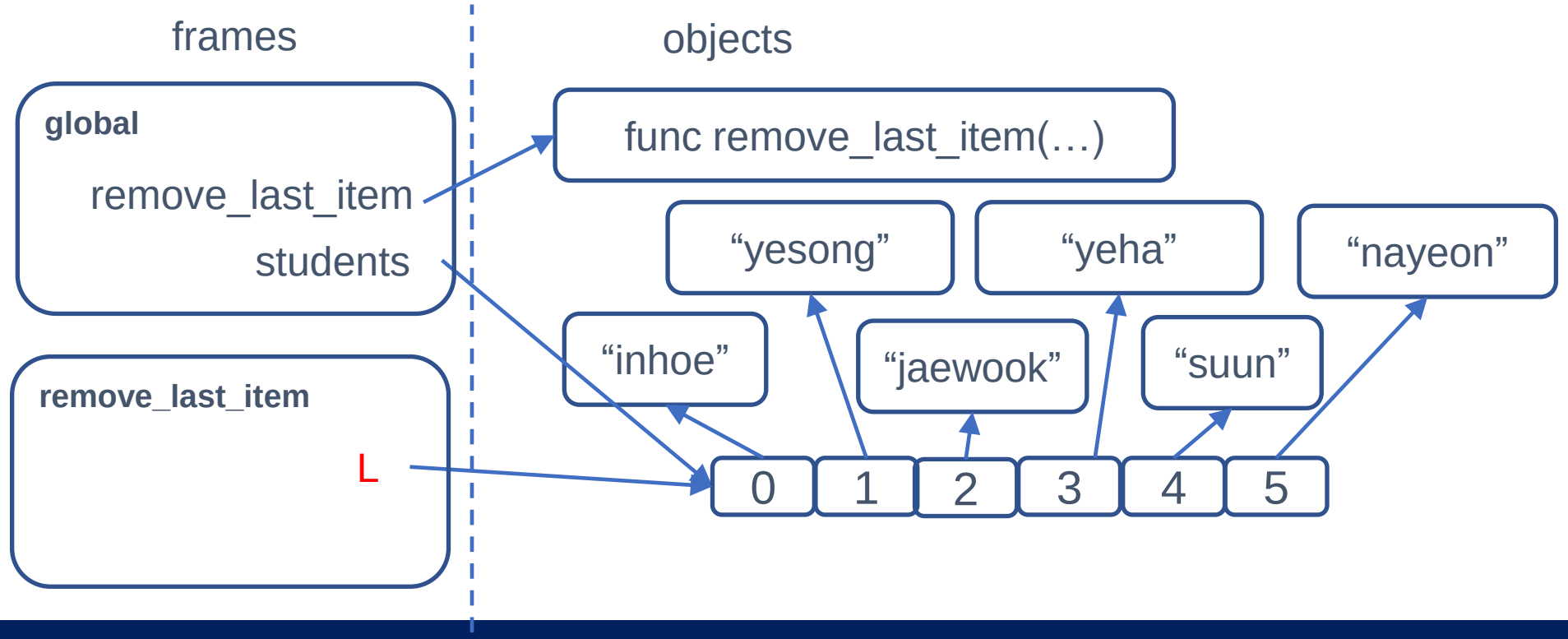
Parameter as Alias – Memory Model

- Memory model after **defining** list *students* and function *remove_last_item*



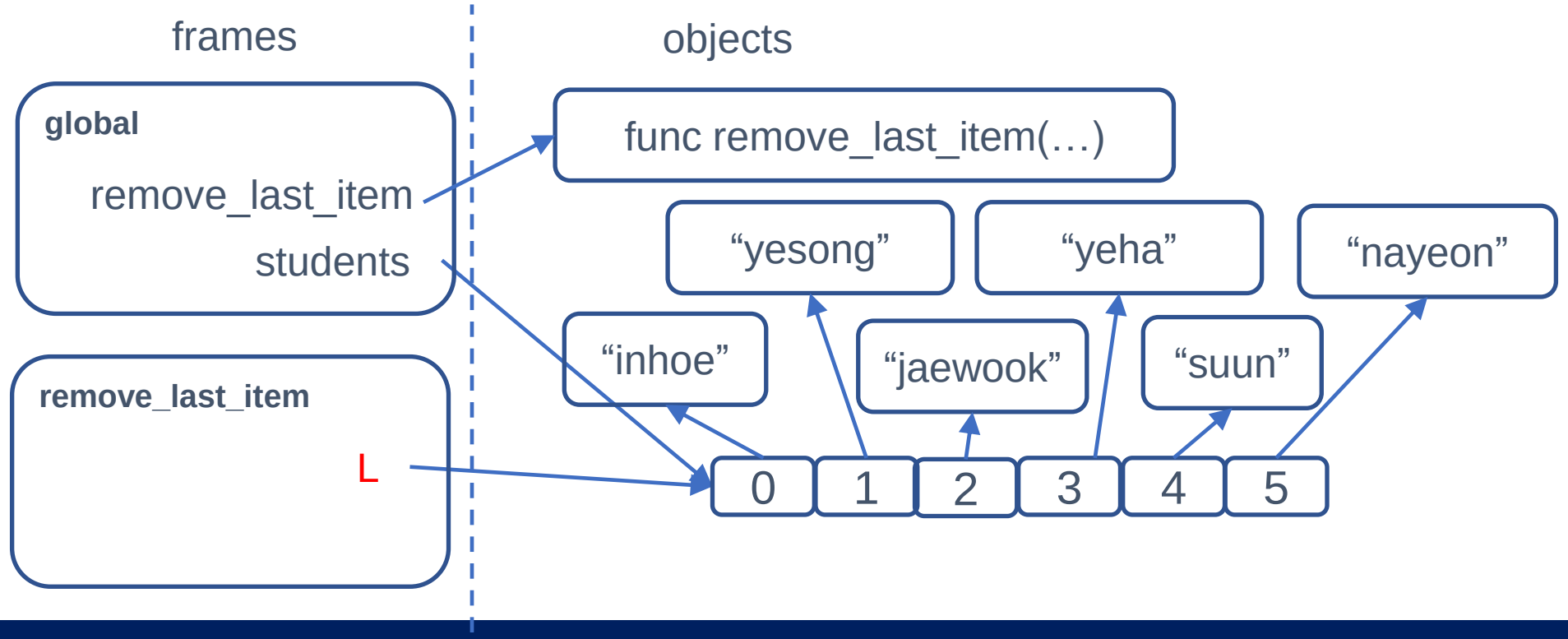
Parameter as Alias – Memory Model

- Memory model when function *remove_last_item(students)* starts to be **executed**
  - List *L* is an **alias** of *students*



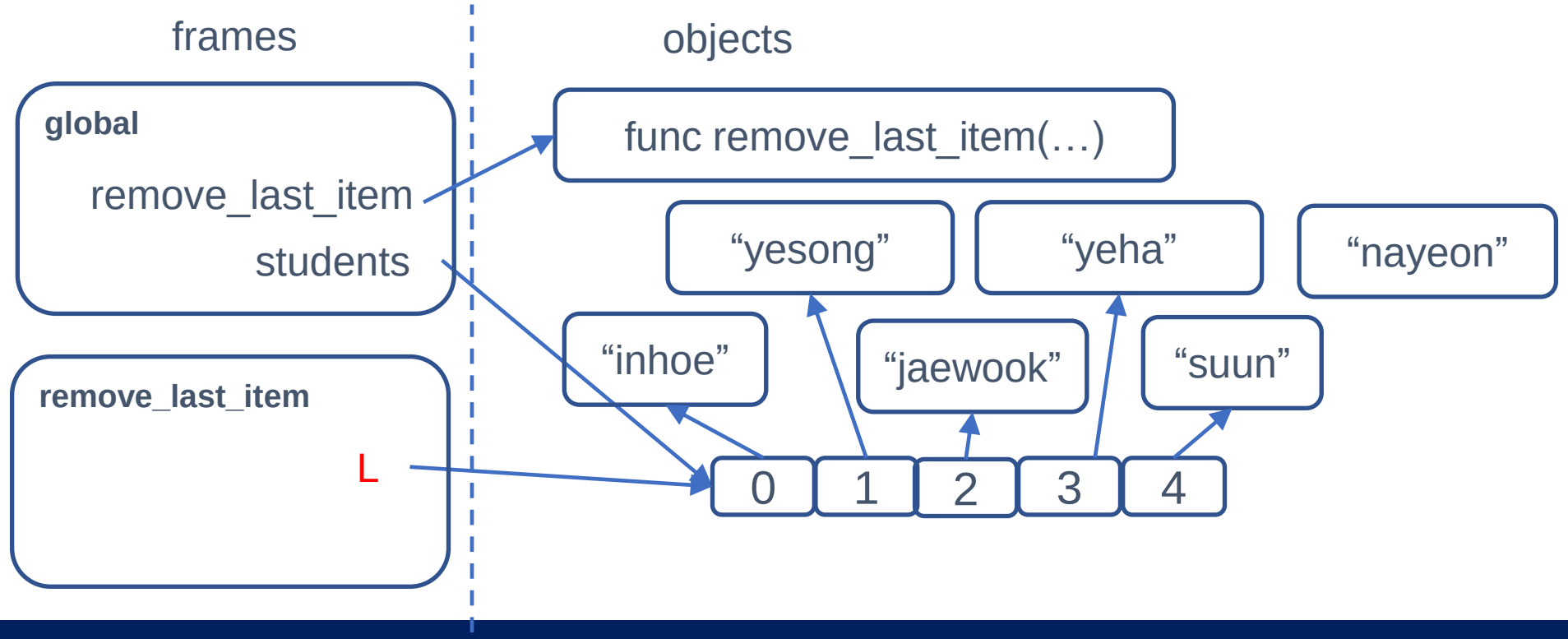
Parameter as Alias – Memory Model

- Memory model after function *remove_last_item(students)* executes "**del L[-1]**"



Parameter as Alias – Memory Model

- After function *remove_last_item(students)* **terminates**
  - List *L* (alias) is gone and there is no return value, but *students* is **changed**



Summary

- Using a slice: Copying
- Using a name: Alias
- A function can change a list through an alias

Loops – For loop

Lecture 5-3

Hyung-Sin Kim

SNU Graduate School of Data Science

Repetition is Tedious

- You **DON'T** want to write an instruction a thousand times to repeat this a thousand times
- Recall that lists were invented for you to not create a thousand variables to store a thousand values
  - Now you have a list of a thousand elements. How can you process all the elements, more efficiently compared to processing a thousand independent variables?
- Solution: Write the instruction **once** and use **loops** to repeat!

For Loop

- General form
  - **for** <<variable>> **in** <<list>>:
  - <<block>>
  - Note that <<block>> is **indented** again

- Execution
  - The loop variable is assigned the **first** item in the list, and the loop block is executed
  - The loop variable is assigned the **second** item in the list, and the loop block is executed
  - …
  - The loop variable is assigned the **last** item in the list, and the loop block is executed

For Loop – List

- Looping over a **list**
  - >>> gsds_courses = [“Math/Stat”, “CFDS”, “ML/DL 1”, “Computing 1”, “Big data 1”]
  - >>> for course in gsds_courses:
  - >>> print(“GSDS offers”, course, “course in Spring 2022.”)
  - GSDS offers Math/Stat course in Spring 2022.
  - GSDS offers CFDS course in Spring 2022.
  - GSDS offers ML/DL 1 course in Spring 2022.
  - GSDS offers Computing 1 course in Spring 2022.
  - GSDS offers Big data 1 course in Spring 2022.

For Loop – String

- Looping over a **string** (the loop variable is assigned to each character)
  - >>> name = “Hyung-Sin Kim”
  - >>> for ch in name:
  - >>> if ch.isupper():
  - >>> print(ch)
  - H
  - S
  - K

For Loop – A Range of Numbers

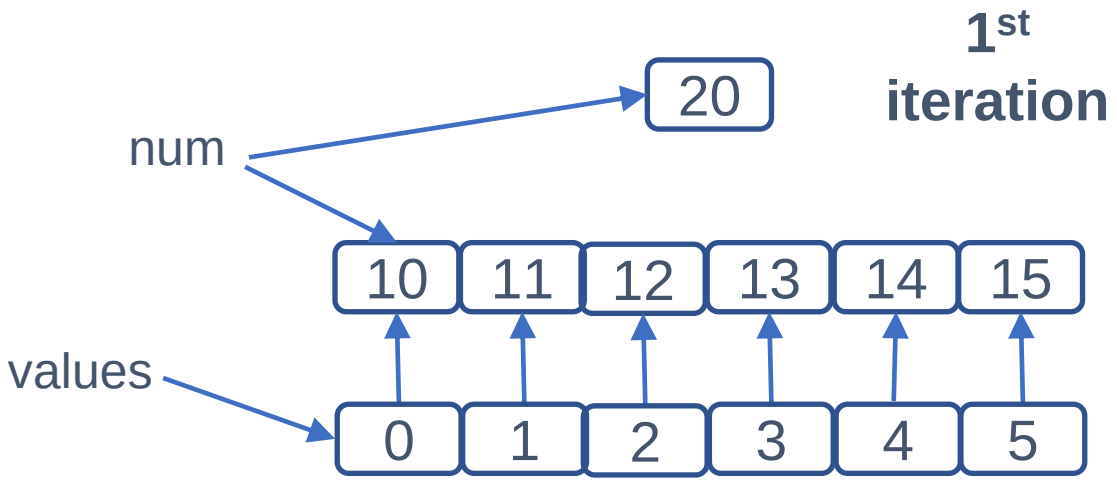
- Looping over **a range of numbers**
  - **range** function
    - range(stop) : an object that will generate a sequence of integers, 0, 1, 2, …, stop-1
    - range(start, stop) : an object that will generate a sequence of integers, start, start+1, start+2, …, stop-1
    - range(start, stop, step): an object that will generate a sequence of integers, start, start+step, start+2*step, …, (until the value becomes larger than stop -1)
    - If start > stop and step < 0, you can get a number sequence in a decreasing order
  - Try some now!
    - You can get the result of range function as a list by doing **list(range(x,y,z))**

For Loop – A Range of Numbers

- Looping over **a range of numbers**
  - >>> total = 0
  - >>> for i in range(1,101):
  - >>> total = total + i
  - >>> total
  - 5050

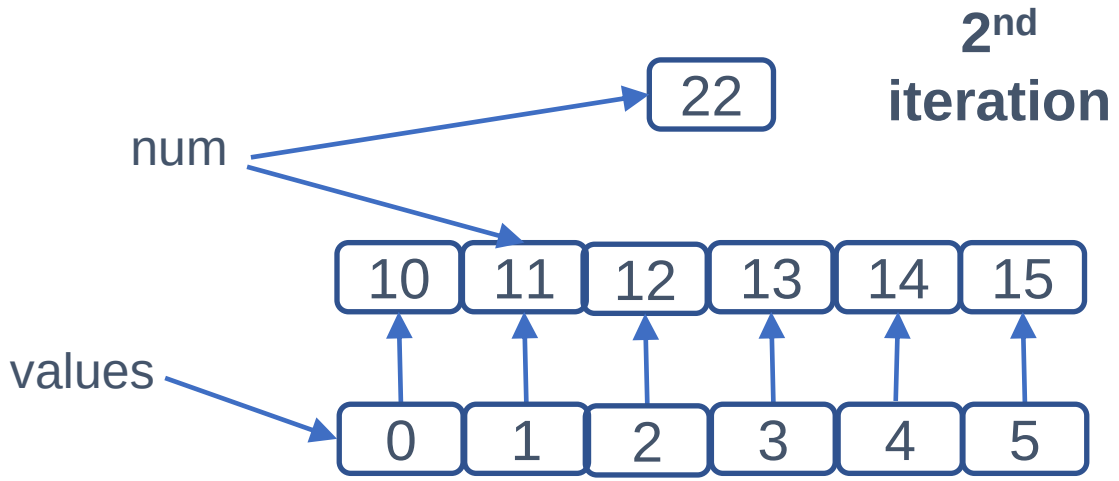
For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables



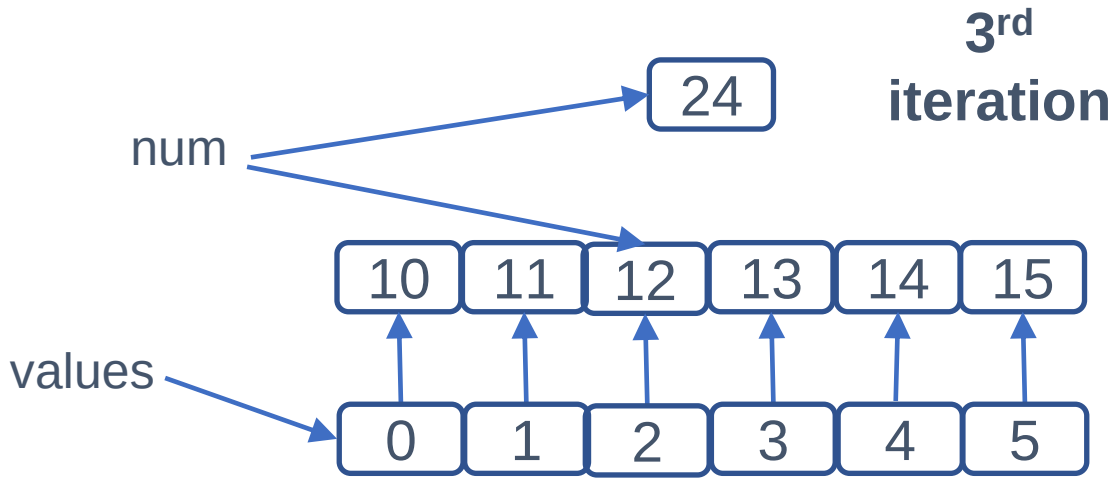
For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables



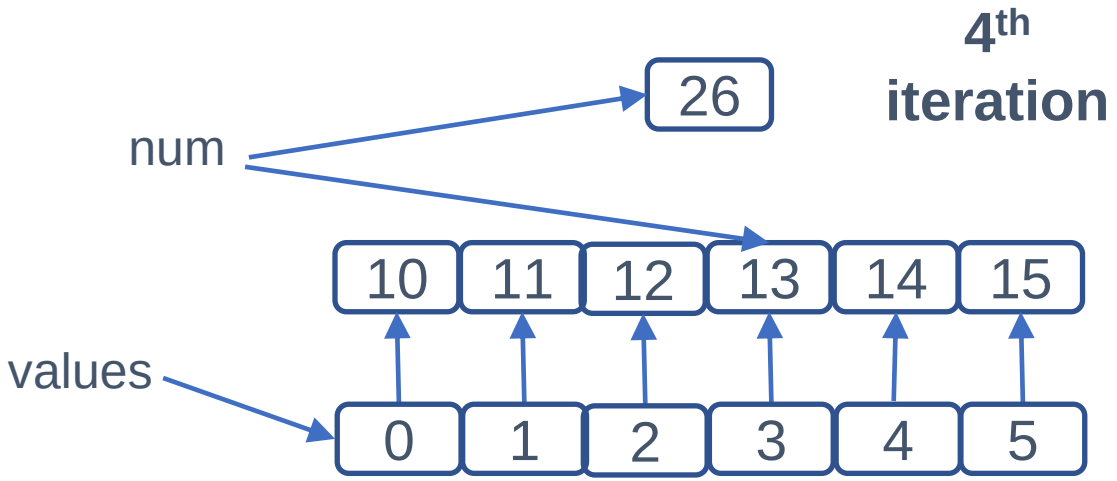
For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables



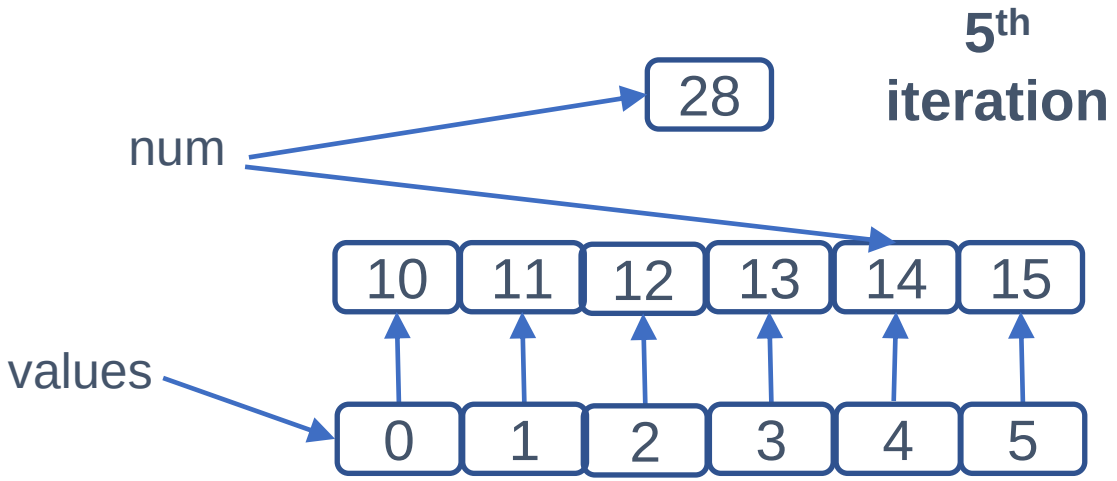
For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables



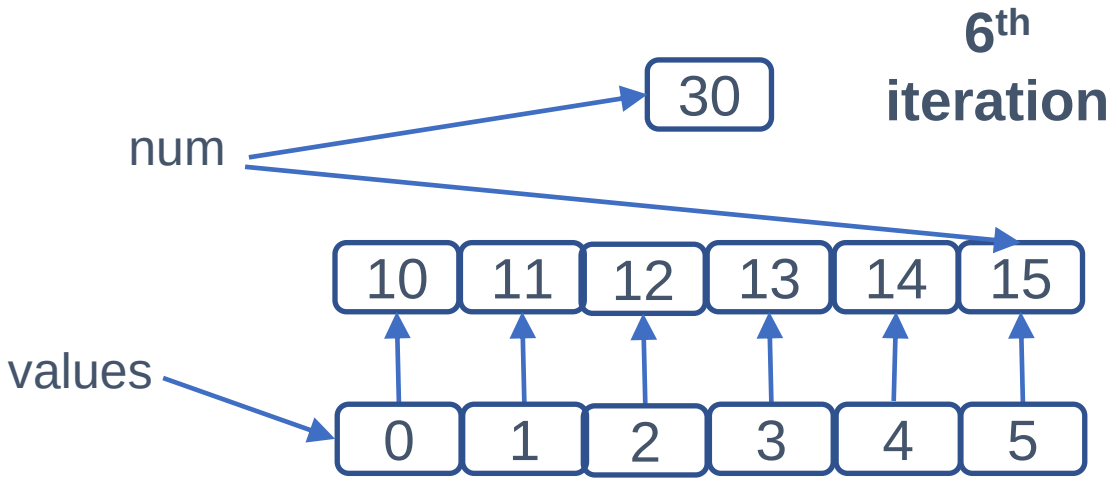
For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables



For Loop – List Values vs. List Indices

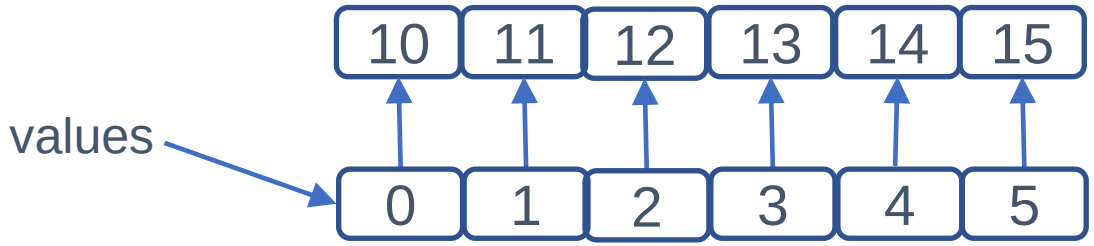
- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables



For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables

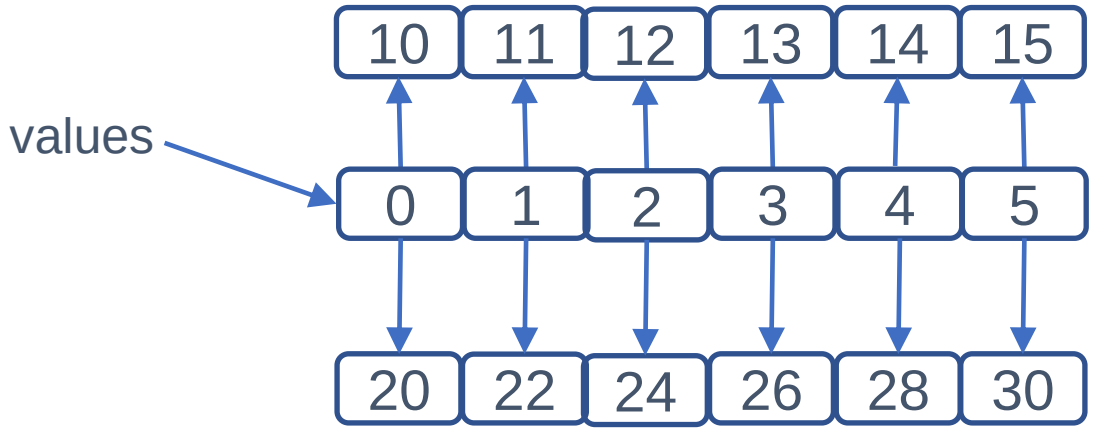
- Looping over **list indices**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for i in range(len(values)):
  - >>> values[i] = values[i] * 2
  - >>> values ➡ [20, 22, 24, 26, 28, 30]
    - Each element in *values* is directly accessed



For Loop – List Values vs. List Indices

- Looping over **each element's value**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for num in values:
  - >>> num = num * 2
  - >>> values ➡ [10, 11, 12, 13, 14, 15]
    - *num* points the same object that *value[x]* points
    - However, *values* does not change since *num* and *values*[x] are still **separate** variables

- Looping over **list indices**
  - >>> values = [10, 11, 12, 13, 14, 15]
  - >>> for i in range(len(values)):
  - >>> values[i] = values[i] * 2
  - >>> values ➡ [20, 22, 24, 26, 28, 30]
    - Each element in *values* is directly accessed



For Loop – Parallel Lists

- Indices are useful when using parallel lists
  - >>> name = [“inhoe”, “yesong”, “jaewook”, “yeha”, “suun”]
  - >>> student_id = [“2021-11”, “2021-12”, “2021-13”, “2021-14”, “2021-15”]
  - >>> for i in range(len(name)):
  - >>> print(i+1, “-th student taking this course is”, name[i], “with id”, student_id[i])

  - 1 –th student taking this course is inhoe with id 2021-11
  - 2 –th student taking this course is yesong with id 2021-12
  - 3 –th student taking this course is jaewook with id 2021-13
  - 4 –th student taking this course is yeha with id 2021-14
  - 5 –th student taking this course is sun with id 2021-15

For Loop – Loops in Loops

- Looping over each combination of multiple lists
  - >>> men = [“hangyeol”, “inwoo”, “uidong”, “kangyeol”]
  - >>> women = [“jihyun”, “anna”, “minjae”, “gaheun”]
  - >>> for man in men:
  - >>> for woman in women:
  - >>> print(man, “and”, woman, “might become a couple.”)
  - 16 outputs…

	jihyun	anna	minjae	gaheun
hangyeol				
inwoo				
uidong				
kangyeol				

Summary

- For loop
  - In a list, a string, and a range of values

- Loops in loops

Loops – While Loop and Loop Control

Lecture 5-4

Hyung-Sin Kim

SNU Graduate School of Data Science

While Loop

- General form
  - while <<expression - condition>>:
  - <<block>>
  - Note that <<block>> is **indented** again

- Execution
  - Execute <<block>> repetitively **as long as** <<expression – condition>> is **True**

While Loop

- Example
  - >>> i = 0
  - >>> while i < 10:
  - >>> i = i+1

- **Infinite loop** is a very typical error
  - Condition variables must be updated properly in <<block>>
  - Condition should not be very specific (i < 10 is better than i != 10)
  - When you experience an infinite loop, just type Ctrl-C to terminate the program

While Loop – Interaction with a User

- Example
  - text = “”
  - while text != “quit”:
  - text = input(“Enter your MBTI type (or ‘quit’ to exit): ”)
  - if text == “ESFP”:
  - print(“You are an Entertainer.”)
  - elif text == “ENFP”:
  - print(“You are a Campaigner.”)
  - elif text == “ISFP”:
  - print(“You are an Adventurer.”)
  - else:
  - print(“You are not like SSAK3.”)

Controlling Loops – Break and Continue

- You might not want to the whole block of for/while loop but control what to execute

- **Break**: The program escape from the loop **right away**
  - >>> first_upper_index = -1
  - >>> randomString = “soimoijSJosijoijAAsBsl”
  - >>> for i in range(len(randomString)):
  - >>> if randomString[i].isupper():
  - >>> first_upper_index = i
  - >>> **break**

  - first_upper_index ⟹ 7

Controlling Loops – Break and Continue

- **Continue**: The loop stops the current iteration and starts next iteration
  - >>> first_upper_index = -1
  - >>> randomString = “soimoijSJosijoijAAsBsl”
  - >>> for i in range(len(randomString)):
  - >>> if randomString[i].islower():
  - >>> **continue**
  - >>> first_upper_index = i
  - >>> **break**

  - first_upper_index ⟹ 7

- Break/Continue can be useful but make code harder to understand. Do NOT abuse it!

Summary

- While loop
  - A common error?

- Break vs. Continue

Thanks!