

# 파이썬으로 배우는 데이터 구조



한동대학교  
전산전자공학부  
김영섭 교수



# 학습 목표

---

Stack의 개념과 Stack의 ADT를 이해하고  
파이썬으로 구현할 수 있다

# **Data Structures in Python**

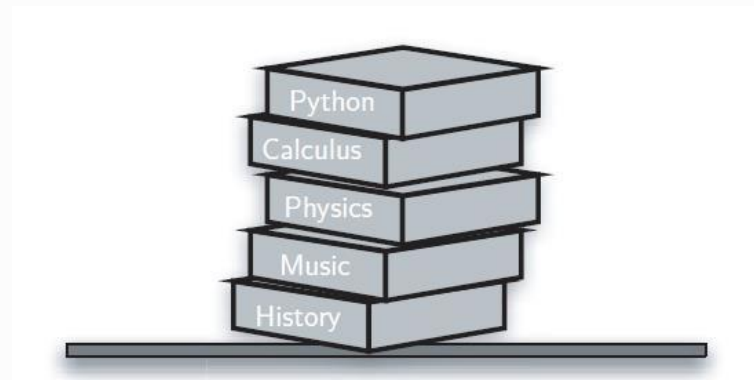
## **Chapter 3 - 1**

- Stack Concept and ADT
- Stack Example - Matching
- Stack Example - Postfix

# Software Design Principles - Modularity

---

- Agenda
  - Introduction
  - The Stack Abstract Data Type (ADT)
  - Two implementations of Stack
- Reference:
  - Textbook: Problem Solving with Algorithms and Data Structures
    - Chapter 3: Basic Data Structures



# Introduction - Linear Structures

---

- Linear structures are data collections whose items are **ordered** depending on how they are **added** or **removed** from the structure.
- Once an item is **added**, it stays in that **position** relative to the other elements that came before and came after it.
- Linear structures can be thought of as having two **ends, top** and **bottom**, (or front and end or front and back).
- What distinguishes one linear structure from another is the way in which items are added and removed, in particular the location where these additions and removals occur, e.g., add only to one end, add to both, etc.

# Introduction - What is a Stack?

---

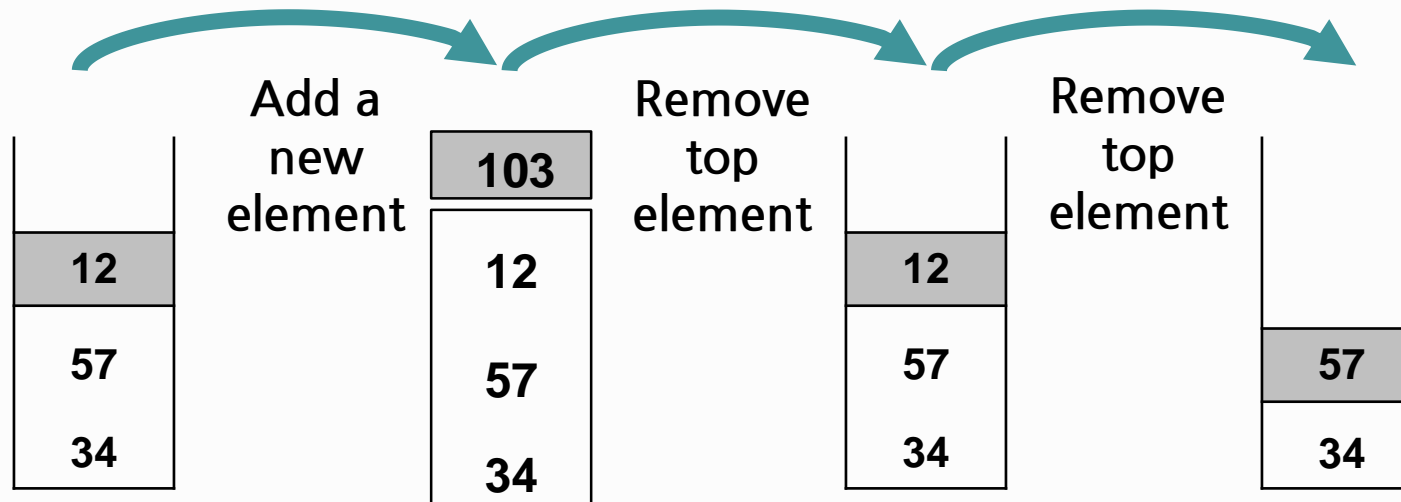
- A stack is an ordered collection of items where the addition of new items and the removal of existing items always takes place at the same end, referred to as the top of the stack.
  - i.e., add at **top**, remove from **top**
- Last-in, first-out (LIFO) property
  - The **last** item placed on the stack will be the **first** item removed.
- Example:
  - A stack of dishes in a cafeteria



# Introduction - Stack Example

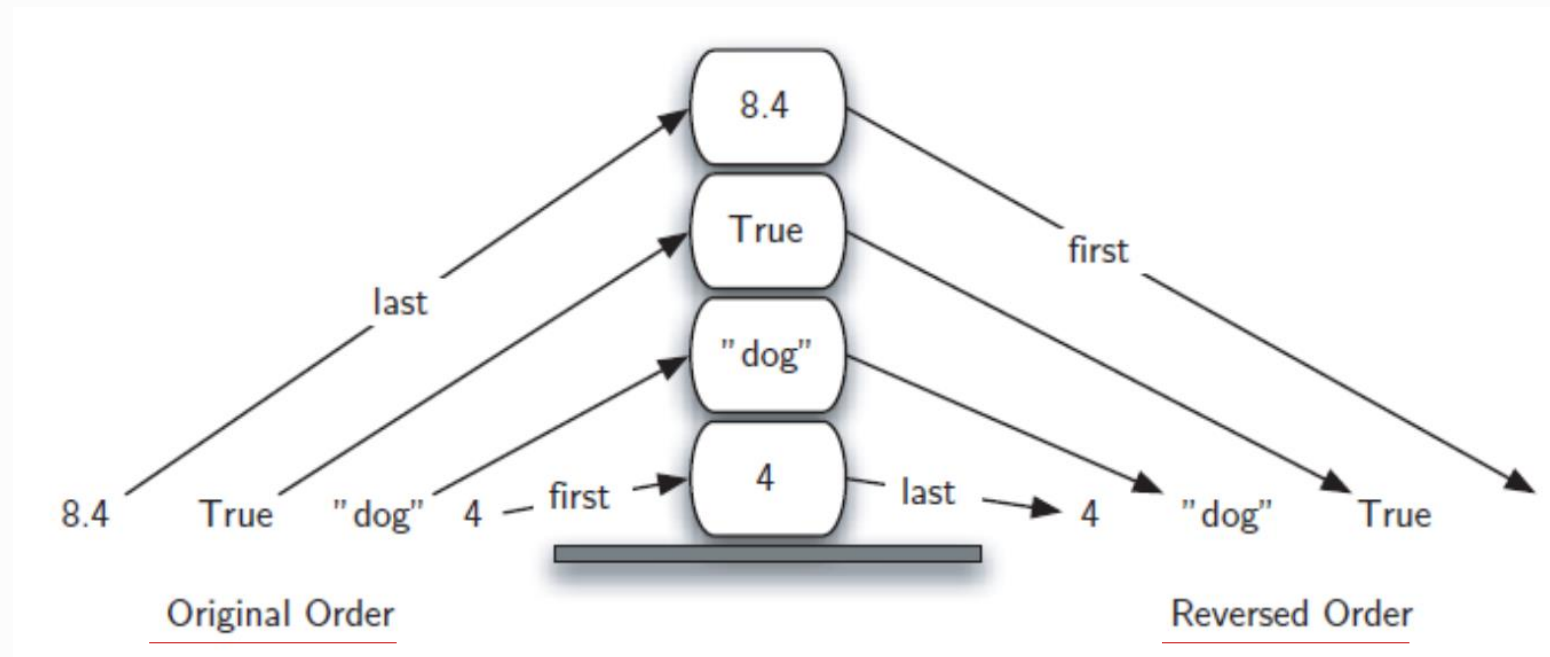
- Add only to the **top** of a Stack
- Remove only from the **top** of the Stack
  - Note: The last item placed on the stack will be the first item removed

Last In - First Out (LIFO)



# Introduction - Orders

- The base of the stack contains the oldest item, the one which has been there the longest.
- For a stack, the order in which items are removed is exactly the reverse of the order that they were placed.



The reversal property of stacks



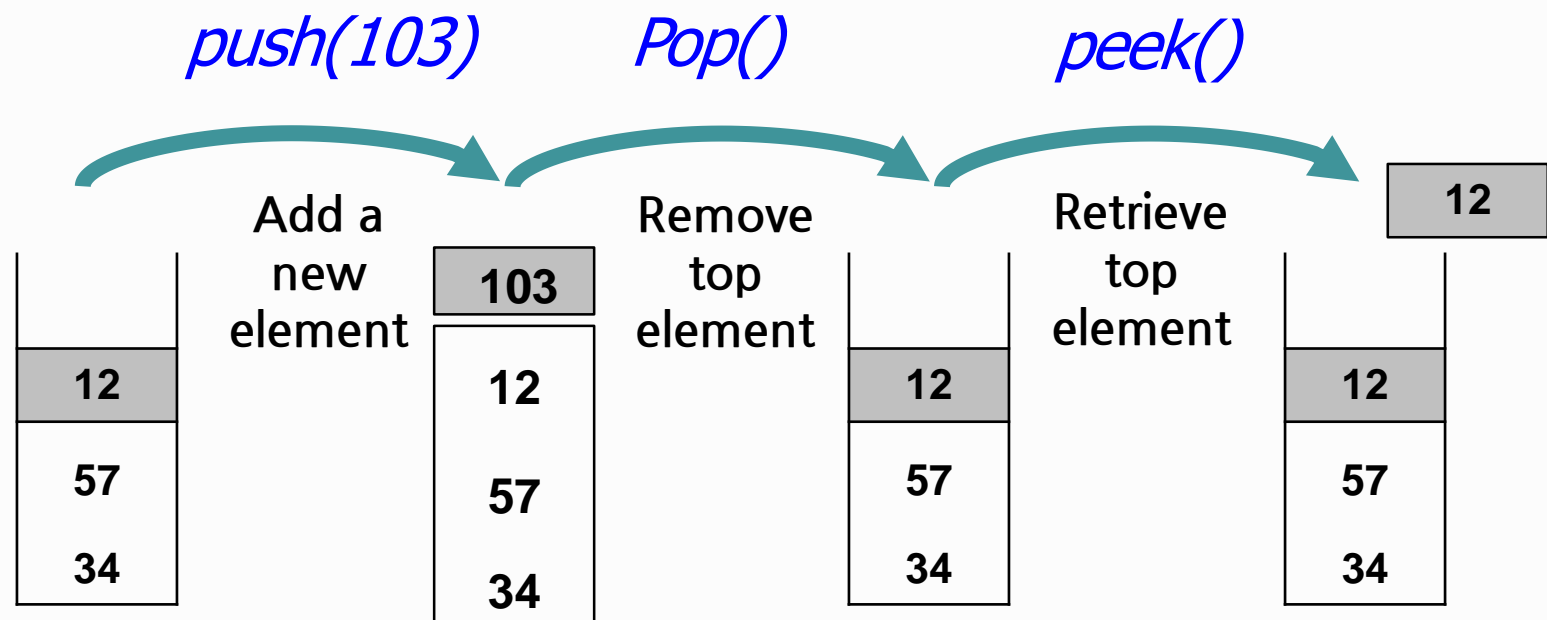
# Introduction - ADT Stack Operations

---

- What are the operations which can be used with a **Stack Abstract Data**?
  - Create an **empty** stack.
  - Determine whether a stack is empty.
    - `is_empty()`
  - Count the number of items in the stack.
    - `size`
  - Add a new item to the stack.
    - `push`
  - Remove from the stack the item that was added most recently.
    - `pop`
  - Retrieve from the stack the item that was added most recently.
    - `peek`

# Introduction - ADT Stack Operations

- What are the operations which can be used with a Stack Abstract Data?



# The Stack Abstract Data Type

---

- **Stack()** creates a new stack that is empty.
  - It needs no parameters and returns an empty stack.
- **push(item)** adds a new item to the top of the stack.
  - It needs the item and returns nothing.
  - The stack is modified.
- **pop()** removes the top item from the stack.
  - It needs no parameters and returns the item.
  - The stack is modified.

**Stack(), push(item) and pop()** are critical operations in order to manipulate the elements of the stack

# The Stack Abstract Data Type

---

- **peek()** returns the top item from the stack but does not remove it.
  - It needs no parameters.
  - The stack is not modified.
- **is\_empty()** tests to see whether the stack is empty.
  - It needs no parameters and returns a Boolean value.
  - The stack is not modified.
- **size()** returns the number of items on the stack.
  - It needs no parameters and returns an integer.
  - The stack is not modified

**peek()**, **is\_empty()** and **size()** are useful to allow the users to retrieve the properties of the stack but the stack is not modified.

# The Stack Abstract Data Type - Exercise 1

- What is the output of the following code snippet?

Code	Stack Implementation	Output
<pre>s = Stack() print(s.is_empty()) s.push(5) print(s.peek()) s.pop() s.push(8.5) print(s.size()) s.push('cat') print(s.peek()) s.pop() print(s.size())</pre>	<pre>s = s = s = s = s = s = s = s = s = s =</pre>	

# The Stack Abstract Data Type - Exercise 1

- What is the output of the following code snippet?

Code	Stack Implementation	Output
<pre>s = Stack() print(s.is_empty()) s.push(5) print(s.peek()) s.pop() s.push(8.5) print(s.size()) s.push('cat') print(s.peek()) s.pop() print(s.size())</pre>	<pre>s = [] s = [] s = [5] s = [5] s = [] s = [8.5] s = [8.5] s = [8.5, 'cat'] s = [8.5, 'cat'] s = [8.5] s = [8.5]</pre>	<pre>True 5 1 cat cat 1</pre>

## The Stack Abstract Data Type - Exercise 2

- What is the output of the following code snippet?

Code

```
s = Stack()
print(s.is_empty())
s.push(4)
print(s.peak())
s.push(8.4)
print(s.size())
s.push('dog')
print(s.peak())
print(s.size())
```

Stack Implementation

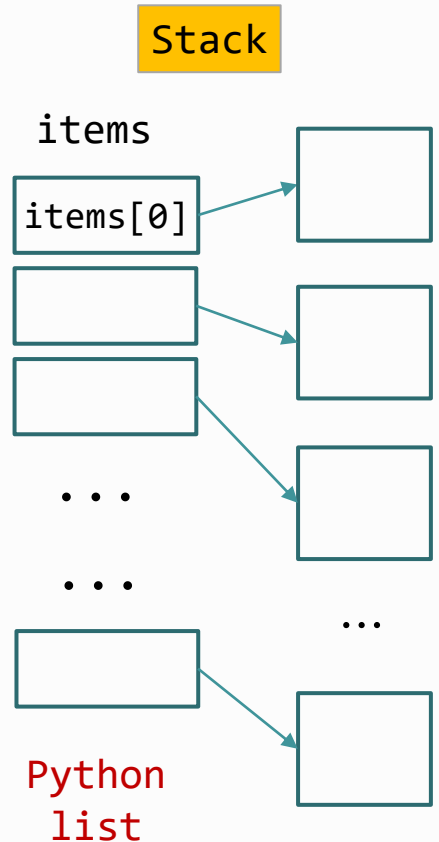
```
s =
s =
s =
s =
s =
s =
s =
s =
s =
```

Output

# The Stack Implementation - The Stack In Python

- We use a python **List** data structure to implement the stack.
  - Remember:  
The addition of new items and the removal of existing items always takes place at the same end, referred to as the top of the stack.

```
class Stack:  
    def __init__(self):  
        self.items = []  
  
    def is_empty(self):  
        return self.items == []  
  
    def size(self):  
        return len(self.items)  
    ...
```

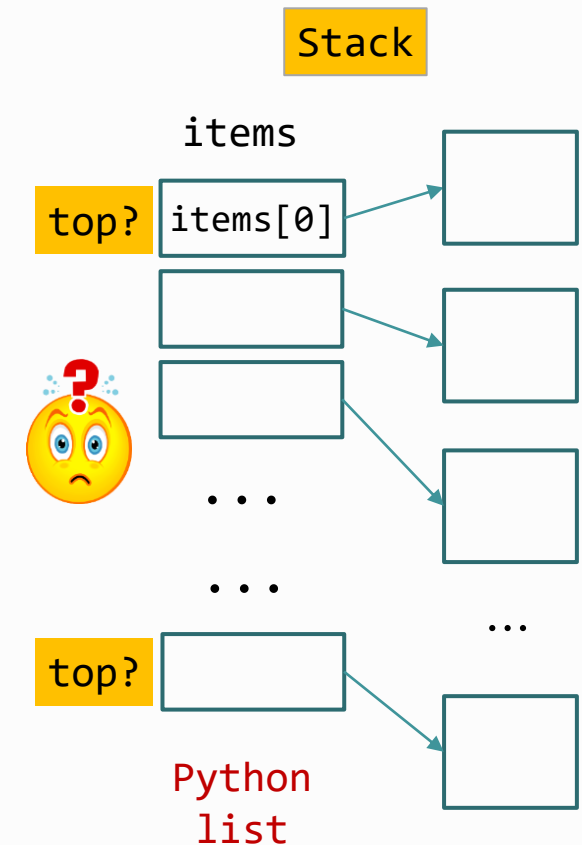




# The Stack Implementation - The Stack In Python

- We use a python List data structure to implement the stack
  - Question:
    - Which “end” of the Python list is better for our Stack implementation?

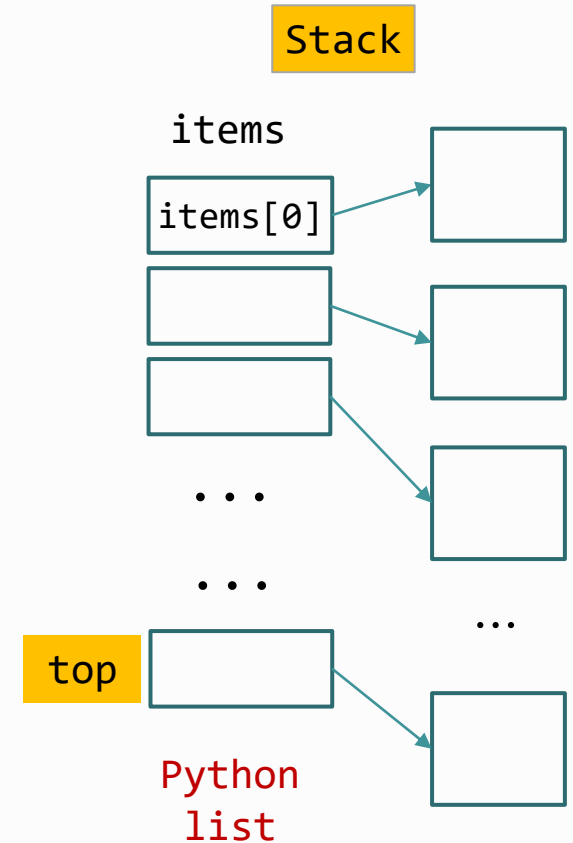
```
class Stack:  
    def __init__(self):  
        self.items = []  
  
    def is_empty(self):  
        return self.items == []  
  
    def size(self):  
        return len(self.items)  
  
    ...
```



# The Stack Implementation - The Stack In Python

- We use a python List data structure to implement the stack
  - Question:
    - Which “end” of the Python list is better for our Stack implementation?
  - Version 1: Big-O?
    - push() and pop():  $O(1)$
    - search():  $O(n)$

```
class Stack:  
    ...  
    def push(self, item):  
        self.items.append(item)  
  
    def pop(self):  
        return self.items.pop()  
    ...
```

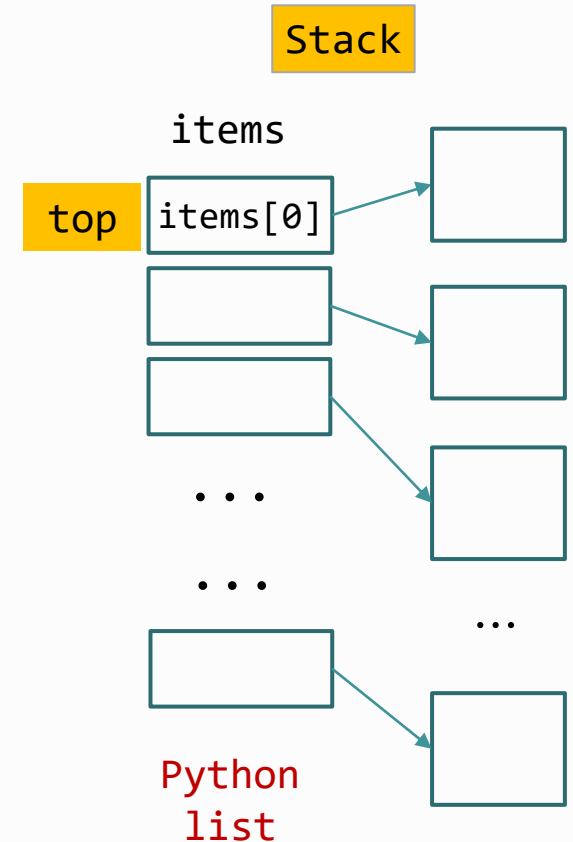


# The Stack Implementation - The Stack In Python

- We use a python List data structure to implement the stack
  - Question:
    - Which “end” of the Python list is better for our Stack implementation?
  - Version 1: Big-O?
    - push() and pop():  $O(1)$
    - search():  $O(n)$
  - Version 2: Big-O?
    - push() and pop():  $O(n)$
    - search():  $O(n)$

```
class Stack:  
    ...  
    def push(self, item):  
        self.items.append(item)  
  
    def pop(self):  
        return self.items.pop()  
    ...
```

```
class Stack:  
    ...  
    def push(self, item):  
        self.items.insert(0, item)  
  
    def pop(self):  
        return self.item.pop(0)  
    ...
```



## Summary

---

- Last-in, first-out data structure (push, pop)
- Access is at one point (top of the stack)
- Python lists support simple implementations of stacks

# 학습 정리

- 1) Stack 자료형은 선형 자료구조이며, Last-in, first-out(LIFO) 특성을 가지고 있다
- 2) Stack 자료형의 메소드에는 `push()`, `pop()`, `peek()` 등이 있다

# 파이썬으로 배우는 데이터 구조

수고했습니다  
곧 다음 시간에  
다시 뵙겠습니다

