

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Stacks: Last In, First Out

Lecture 9 · Wednesday 21 October 2026 · Canning, Broder & Lafore, Ch 4 (Stacks and Queues)

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–5 min) Where you use a stack every day
2. (5–11 min) Abstract data types: what versus how
3. (11–20 min) The stack: operations, implementation, cost
4. (20–29 min) Application: matching brackets, traced
5. (29–36 min) Application: the call stack and recursion
6. (36–41 min) Application: undo and redo with two stacks
7. (41–50 min) Exercise, solution, summary

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Distinguish	an abstract data type (what) from a data structure that implements it (how)
Implement	a stack with O(1) push, pop and peek, and justify which end of a list to use
Apply	a stack to bracket matching and to undo/redo
Explain	how the call stack makes recursion work, and why deep recursion fails

Recap: Sorting Is Done; Now a New Kind of Structure

IIT Madras Zanzibar

1 Arrays

touch any item: a[i] in O(1)

2 Sorting

insertion sort: O(n + inversions)

3 Today: rules of access

a stack lets you touch only the TOP item

4 Why restrict?

the rule makes some problems easy and safe

? **Think:** can you think of a situation where being ALLOWED to touch only the most recent item is actually helpful?

You Use a Stack Many Times Every Day

IIT Madras Zanzibar

B

Browser back button

visit a page: push. Press back: pop the most recent page

U

Undo (Ctrl+Z)

every edit is pushed; undo pops the most recent edit

P

Plates in the canteen

add to the top, take from the top: never from the middle

F

Function calls

Python pushes a frame for each call and pops it on return

?

Think: you visit pages A, B, C, press Back twice, then visit D. Where does Back take you now? Where did C go?

The one rule: Last In, First Out (LIFO). You may only touch the top.

Stacks: Last In, First Out

IIT Madras Zanzibar

→ **Abstract Data Types**

The Stack

Application: Matching Brackets

Application: The Call Stack

Application: Undo and Redo

In-Class Exercise

Abstract Data Types: What It Does Versus How It Is Built

IIT Madras Zanzibar

ADT: the WHAT

- a promise of operations and behaviour
- stack: push, pop, peek, is_empty
- like a car's steering wheel and pedals
- users depend only on this

Data structure: the HOW

- one concrete way to keep the promise
- stack on an array (today) or a linked list (Week 4)
- like the engine: petrol, diesel, electric
- can change without breaking users

Why it matters: you can swap the implementation for a faster one without touching the code that uses it.

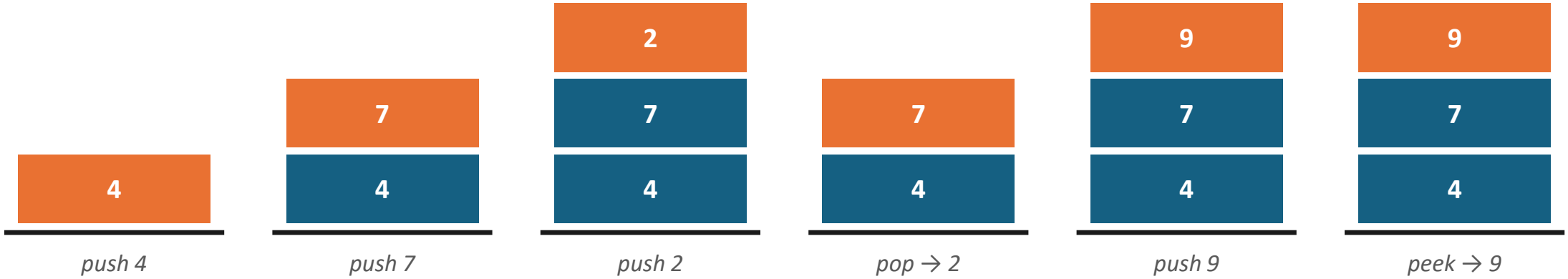
? **Think:** Python's list, set and dict: which of them is an ADT, which a data structure, or both?

Stacks: Last In, First Out

IIT Madras Zanzibar

- Abstract Data Types
- → **The Stack**
- Application: Matching Brackets
- Application: The Call Stack
- Application: Undo and Redo
- In-Class Exercise

The Stack ADT: Four Operations, All O(1)



Operation	What it does	Cost
push(x)	put x on top	O(1)*
pop()	remove and return the top	O(1)
peek()	return the top, don't remove	O(1)
is_empty()	is there anything left?	O(1)

Underflow: pop() on an empty stack is an error. Check is_empty() first.

** amortized, on a dynamic array*

Implementing a Stack: Use the END of the List

IIT Madras Zanzibar

```
class Stack:
    def __init__(self):
        self._items = []

    def push(self, x):
        self._items.append(x)      # O(1)*

    def pop(self):
        if not self._items:
            raise IndexError('pop from empty stack')
        return self._items.pop()       # O(1)

    def peek(self):
        return self._items[-1]

    def is_empty(self):
        return not self._items
```

1 **Top = END of list**

append and pop(): O(1)

2 **Top = FRONT of list**

insert(0) and pop(0): O(n) each!

3 **Same ADT**

users see no difference, except speed

? **Think:** a classmate builds the stack with insert(0, x) and pop(0). Their tests all pass. What is wrong?

Stacks: Last In, First Out

IIT Madras Zanzibar

- Abstract Data Types
- The Stack
- → **Application: Matching Brackets**
- Application: The Call Stack
- Application: Undo and Redo
- In-Class Exercise

Matching Brackets: Every Code Editor Does This

IIT Madras Zanzibar

Input: ([{ }])



1 Push

every opening bracket ([{

2 Pop and match

every closing bracket must match the top

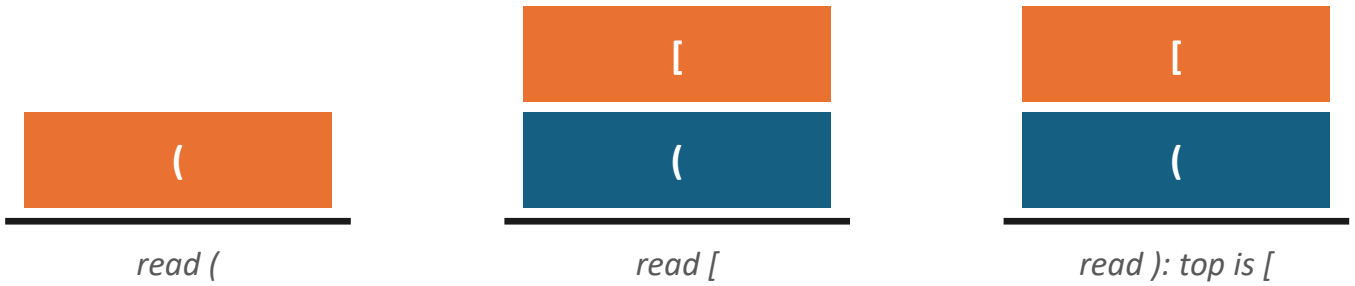
3 Fail if

mismatch, empty stack on a closer, or leftovers at the end

? **Think:** trace '([ )]'. At which character does it fail, and why? What about '(( )'?

A Failing Example, Traced: ([ )]

Input: ([ )]



Mismatch!
) needs (, but the top is [

1 **Fails at the 3rd character**

exactly where the nesting breaks

2 **Your editor**

underlines this spot in red the same way

? **Think:** trace '(( )'. Where does it fail? Which of the three failure rules applies?

Bracket Matching in Python

IIT Madras Zanzibar

```
def balanced(text):
    pairs = {')': '(', ']': '[', '}': '{'}
    stack = []
    for ch in text:
        if ch in '([{':
            stack.append(ch)
        elif ch in pairs:
            if not stack or stack.pop() != pairs[ch]:
                return False
    return not stack

balanced('([{}])')   # True
balanced('([)]')     # False
```

1 **Time**

O(n): each character pushed and popped at most once

2 **Space**

O(n) worst case: a long run of openers

3 **Also used for**

HTML/XML tags, JSON validation, compilers

Stacks: Last In, First Out

IIT Madras Zanzibar

Abstract Data Types

The Stack

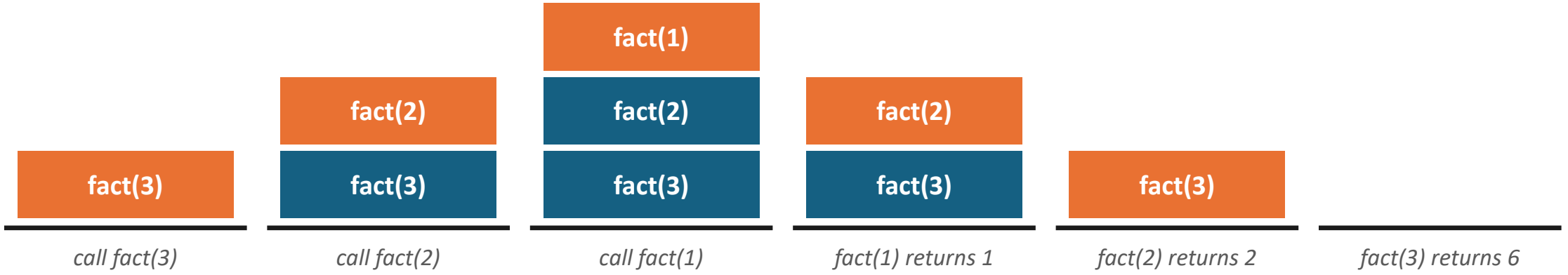
Application: Matching Brackets

→ **Application: The Call Stack**

Application: Undo and Redo

In-Class Exercise

The Call Stack: How Recursion Actually Works




```
import sys
sys.getrecursionlimit()   # 1000
fact(5000)  # RecursionError
```

Stack overflow: too many frames exhaust the call stack. That is where stackoverflow.com got its name.

? **Think:** a function calls itself 500 times in a chain. How many frames are on the call stack at the deepest point?

Stacks: Last In, First Out

IIT Madras Zanzibar

Abstract Data Types

The Stack

Application: Matching Brackets

Application: The Call Stack

→ **Application: Undo and Redo**

In-Class Exercise

Undo and Redo With Two Stacks

IIT Madras Zanzibar

1 Edit

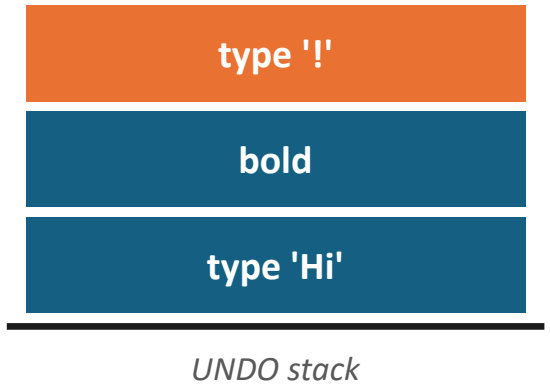
push the edit onto UNDO; clear REDO

2 Undo

pop from UNDO, reverse it, push onto REDO

3 Redo

pop from REDO, reapply it, push onto UNDO



UNDO stack

undo moves the top edit across →



REDO stack

? **Think:** why must a NEW edit clear the redo stack? What would go wrong if it didn't?

Common Mistakes to Avoid

IIT Madras Zanzibar

1. **Using the front of a list**
   insert(0)/pop(0) make every operation O(n)

2. **Popping an empty stack**
   check is_empty() first, or catch the error

3. **Forgetting leftovers**
   after the last bracket, the stack must be EMPTY

4. **Mixing up peek and pop**
   peek looks; pop removes

5. **Confusing ADT and code**
   the stack ADT is the promise; the list is one way to build it

6. **Deep recursion in Python**
   about 1,000 frames, then RecursionError

In the lab: Lab 2: build a Stack class and a bracket checker; the tests include all three failure cases.

In-Class Exercise (6 minutes, in pairs)

IIT Madras Zanzibar

IN-CLASS TASK: trace the stack, show its contents

1. Start empty: push 5, push 3, pop, push 8, push 1, pop, pop, push 6. What is popped each time? What remains?

2. Which of these are balanced? (a) {[()()]} (b) ([)] (c) ((((d))(

3. How can you reverse the string 'ZANZIBAR' using a stack? What is the cost?

4. In an editor you type A, type B, type C, undo, undo, type D. What is on the undo stack? On the redo stack?

Solution

IIT Madras Zanzibar

1 **1: popped 3, 1, 8; remaining [5, 6]**

after pops the stack is 5 (bottom) then 6 (top).

2 **2: only (a) is balanced**

(b) ']' meets '(' on top: mismatch; (c) one '(' left over; (d) ')' arrives with the stack empty.

3 **3: push all, then pop all**

push Z, A, N, Z, I, B, A, R; popping gives RABIZNAZ. O(n) time, O(n) space.

4 **4: undo = [A, D], redo = []**

two undos move C then B to redo; typing D clears redo.

Quiz 1: expect a push/pop trace and a bracket-matching question.

Summary and Exam Checkpoints

IIT Madras Zanzibar

ADT vs data structure

what it promises vs how it is built

Stack = LIFO

push, pop, peek, is_empty: all O(1) at the END of a list

Brackets and undo

one pass with a stack; two stacks for undo/redo

Call stack

recursion = a stack of frames; too deep = RecursionError

Next

Thu 22 Oct, 10:00: Stacks at work: calculators and stock prices

Thu 22 Oct, 2:00 pm: Lab 2: sorts, a Stack class, a bracket checker and a calculator

Read: Canning, Broder & Lafore, Chapter 4, Stacks

References and Further Reading

Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 4, Stacks and Queues.

Liskov, B. & Zilles, S. (1974). Programming with abstract data types. ACM SIGPLAN Notices, 9(4), 50–59.

Python Software Foundation. sys.getrecursionlimit. docs.python.org/3/library/sys.html

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Section 10.1.