STACK — Exam Cheat Sheet

Class XII Computer Science (083) | LIFO, push, pop and the 5 board question types

1. What and why

- A stack is a linear data structure that follows **LIFO — Last In, First Out**. The element added last is the first one removed, like a pile of plates.
- All insertions and deletions happen at one end only, called the **TOP**.
- In the CBSE syllabus a stack is always made with a **Python list**. The end of the list is the top.

Term	Meaning	Python
Push	add an element at the top	`stk.append(x)`
Pop	remove and return the top element	`stk.pop()`
Peek / Top	see the top element without removing it	`stk[-1]`
Underflow	pop or peek on an EMPTY stack	`check len(stk) == 0 first`
Overflow	push on a FULL stack (fixed-size stacks only)	`a Python list grows, so rare`

Always check for an empty stack before pop(). stk.pop() on an empty list crashes with IndexError.

2. Stack operations using a list

Operation	Syntax	What it does
Create	`stk = []`	makes an empty stack
Push	`stk.append(x)`	adds x at the end (top)
Pop	`stk.pop()`	removes and returns the last element (top)
Peek	`stk[-1]`	returns the top element, does not remove it
Is empty?	`len(stk) == 0`	True when nothing is in the stack
Size	`len(stk)`	number of elements
Display	`print(stk[::-1])`	prints the stack from top to bottom

- Use pop() with no argument. pop(0) removes from the FRONT — that is a queue, not a stack.
- print(stk) shows bottom to top. stk[::-1] is a reversed copy, so print(stk[::-1]) shows the top first and does not change the stack.

3. Template A — basic stack functions

```
stk = []

def is_empty():
    if len(stk) == 0:
        return True
    else:
        return False

def push_element(item):
    stk.append(item)                  # add at the top

def pop_element():
    if len(stk) == 0:
        print("Underflow")            # nothing to pop
    else:
        return stk.pop()              # remove and return the top

def peek():
    if len(stk) == 0:
        print("Underflow")
```

```
    else:
        return stk[-1]                # top element, not removed

def display():
    if len(stk) == 0:
        print("Stack Empty")
    else:
        print(stk[::-1])              # top to bottom
```

4. Template B — the 3-step recipe (learn this by heart)

```
stk = []                              # STEP 1 : empty stack

def push_element(data):               # STEP 2 : push if condition is true
    for rec in data:
        if <condition>:
            stk.append(<part of rec>)

def pop_element():                    # STEP 3 : pop all, then say empty
    while len(stk) != 0:
        print(stk.pop())
    print("Stack Empty")
```

Every board stack question is this recipe. Only the data, the condition and the part pushed change.

Type	Data passed	Loop	Read a part	Push
1	numbers typed by the user	`for i in range(10):`	`n = int(input())`	`stk.append(n)`
2	one list per call	`no loop`	`rec[0], rec[2]`	`stk.append([rec[0], rec[1]])`
3	list of tuples	`for rec in L:`	`rec[0], rec[1]`	`stk.append(rec[0])`
4	list of lists	`for rec in L:`	`rec[0], rec[2]`	`stk.append([rec[0], rec[1]])`
5	list of dictionaries	`for rec in L:`	`rec['Price']`	`stk.append([rec['Name'], rec['Price']])`

pop_element() is exactly the same in all 5 types.

5. Type 1 — input from the user

Accept 10 integers. Push the three-digit even numbers onto a stack, then pop and display them.

```
stk = []

def push_element():
    for i in range(10):
        n = int(input("Enter a number: "))
        if 100 <= n <= 999 and n % 2 == 0:   # three-digit AND even
            stk.append(n)

def pop_element():
    while len(stk) != 0:              # repeat until the stack is empty
        print(stk.pop())              # removes and prints the TOP element
    print("Stack Empty")

push_element()
pop_element()

# Input  : 12 31 320 457 6 92 924 220 1 218
# Stack  : [320, 924, 220, 218]
# Output : 218  220  924  320  Stack Empty
```

input() goes INSIDE the loop, and range(10) fixes how many numbers are read.

6. Type 2 — one list passed as the parameter

Each customer record is a list [Name, Phone, City]. push_element(rec) pushes [Name, Phone] of customers who live in Goa.

```
stk = []
```

```
def push_element(rec):                # rec = ONE customer list
    if rec[2] == "Goa":
        stk.append([rec[0], rec[1]])  # push only Name and Phone

def pop_element():
    while len(stk) != 0:              # repeat until the stack is empty
        print(stk.pop())              # removes and prints the TOP element
    print("Stack Empty")

push_element(["Gurdas", "9999999999", "Goa"])
push_element(["Julee", "8888888888", "Mumbai"])
push_element(["Murugan", "7777777777", "Cochin"])
push_element(["Ashmit", "1010101010", "Goa"])
pop_element()

# Stack  : [['Gurdas', '9999999999'], ['Ashmit', '1010101010']]
# Output : ['Ashmit', '1010101010']  ['Gurdas', '9999999999']  Stack Empty
```

No loop inside the function — one record arrives per call, so the function is called once for each record.

7. Type 3 — list of tuples passed

A list of tuples holds (Product, Price). Push the names of products costing more than 50.

```
stk = []

def push_element(L):
    for rec in L:                     # rec = one tuple (Product, Price)
        if rec[1] > 50:
            stk.append(rec[0])        # push only the name

def pop_element():
    while len(stk) != 0:              # repeat until the stack is empty
        print(stk.pop())              # removes and prints the TOP element
    print("Stack Empty")

L = [("Laptop", 90000), ("Mobile", 30000), ("Pen", 50), ("Headphones", 1500)]
push_element(L)
pop_element()

# Stack  : ['Laptop', 'Mobile', 'Headphones']
# Output : Headphones  Mobile  Laptop  Stack Empty
```

rec[0] is the name, rec[1] is the price. To push the whole tuple write stk.append(rec).

8. Type 4 — list of lists (nested list) passed

A nested list holds [City, Country, Distance from Delhi]. Push [City, Country] for cities outside India less than 3500 km from Delhi.

```
stk = []

def push_element(L):
    for rec in L:                     # rec = [City, Country, Distance]
        if rec[1] != "India" and rec[2] < 3500:
            stk.append([rec[0], rec[1]])

def pop_element():
    while len(stk) != 0:              # repeat until the stack is empty
        print(stk.pop())              # removes and prints the TOP element
    print("Stack Empty")

L = [["New York", "U.S.A.", 11734], ["Naypyidaw", "Myanmar", 3219],
     ["Dubai", "UAE", 2194],         ["London", "England", 6693],
     ["Gangtok", "India", 1580],     ["Colombo", "Sri Lanka", 3405]]
push_element(L)
pop_element()

# Stack  : [['Naypyidaw', 'Myanmar'], ['Dubai', 'UAE'], ['Colombo', 'Sri Lanka']]
# Output : ['Colombo', 'Sri Lanka']  ['Dubai', 'UAE']  ['Naypyidaw', 'Myanmar']
#          Stack Empty
```

Same as Type 3, but each rec is a list. Two conditions are joined with and.

9. Type 5 — list of dictionaries passed

Each fruit record is a dictionary with keys 'Name', 'Origin', 'Price'. Push [Name, Price] of fruits priced below 100.

```
stk = []

def push_element(L):
    for rec in L:                    # rec = one dictionary
        if rec['Price'] < 100:       # read by KEY, not by position
            stk.append([rec['Name'], rec['Price']])

def pop_element():
    while len(stk) != 0:             # repeat until the stack is empty
        print(stk.pop())             # removes and prints the TOP element
    print("Stack Empty")

L = [{'Name': 'Apple',  'Origin': 'France',      'Price': 120},
     {'Name': 'Banana', 'Origin': 'India',       'Price': 60},
     {'Name': 'Mango',  'Origin': 'India',       'Price': 90},
     {'Name': 'Kiwi',   'Origin': 'New Zealand', 'Price': 150}]
push_element(L)
pop_element()

# Stack  : [['Banana', 60], ['Mango', 90]]
# Output : ['Mango', 90]  ['Banana', 60]  Stack Empty
```

A dictionary is read with rec['Price'], never rec[2].

Also asked — one dictionary with key : value pairs

```
def push_element(D):                 # D = {"Pen": 106, "Pencil": 59, ...}
    for k in D:                      # loop over the KEYS
        if D[k] > 75:                # D[k] is the value
            stk.append(k)            # push the key
```

10. Quick reference — what happens to the stack

Statement	Stack after it	Returned / printed
`stk = []`	`[]`	—
`stk.append(10)`	`[10]`	—
`stk.append(20)`	`[10, 20]`	—
`stk.append(30)`	`[10, 20, 30]`	—
`stk[-1]`	`[10, 20, 30]`	30 (peek, nothing removed)
`print(stk[::-1])`	`[10, 20, 30]`	[30, 20, 10] (stack unchanged)
`stk.pop()`	`[10, 20]`	30
`stk.pop()`	`[10]`	20
`len(stk)`	`[10]`	1

Applications: reversing a string, undo / redo, browser back button, checking balanced brackets, function calls (recursion).

11. Marks are lost here — check every time

- Reading the condition wrong — “more than 75” is > 75, “75 or more” is >= 75.
- Calling pop() without checking for an empty stack — the program crashes with IndexError.
- Changing the message — write exactly what the question says: “Stack Empty”, “UNDERFLOW”, “EMPTY STACK”.
- Using print when the question says the function returns the element — write return stk.pop().
- Using pop(0) instead of pop() — pop(0) removes from the front.
- Pushing the whole record when only some fields are asked — build a new list: [rec[0], rec[1]].

- Reading a dictionary by position (rec[2]) instead of by key (rec['Price']).
- Putting print("Stack Empty") inside the while loop — keep it after the loop so it prints once.
- Writing stk.reverse() to display — it changes the stack itself. print(stk[::-1]) leaves it unchanged.
- Using the global stk when the function header gives the stack as a parameter — use the parameter name.
- Forgetting to call the functions at the end when the question says “call both functions”.