Data Science & AI

DATA STRUCTURES Through Python

STACK

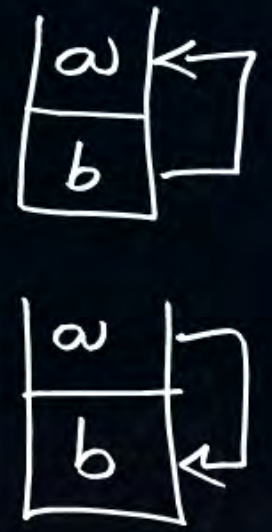
Lecture No.- 05

By- Satya sir

Recap of Previous Lecture

Expression Evaluation

- Postfix Exp Evaluation : L To R
- Prefix Exp Evaluation : R To L



Topics to be Covered

Stack Implementation

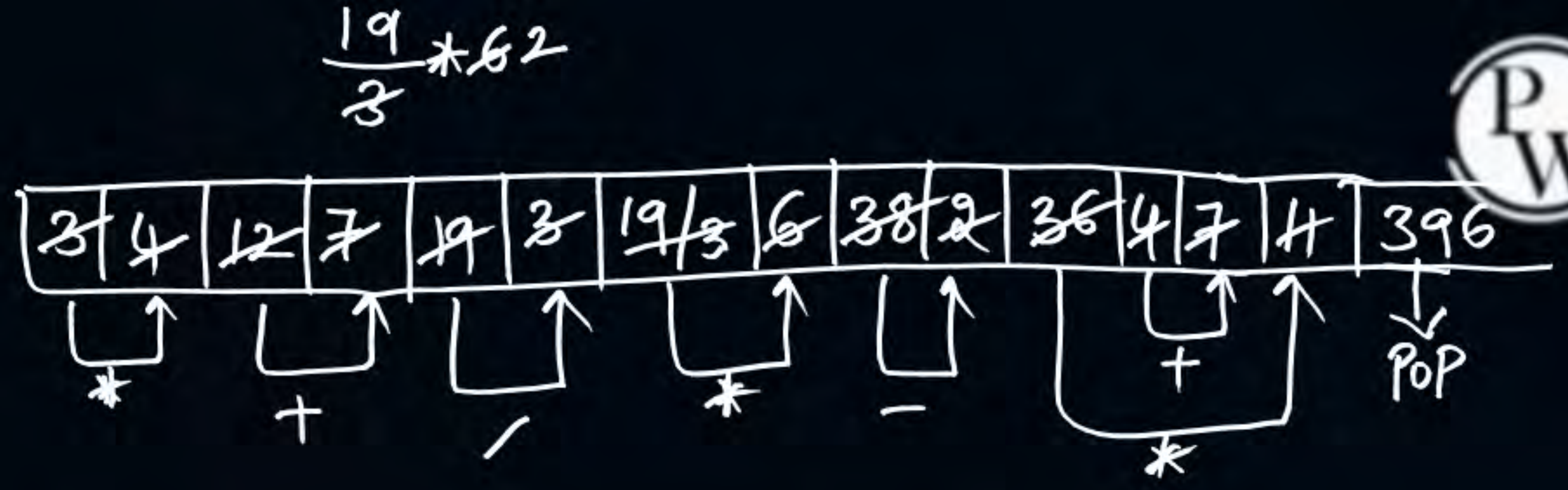
- Using Lists
- Using deque
- Using Lifoqueue

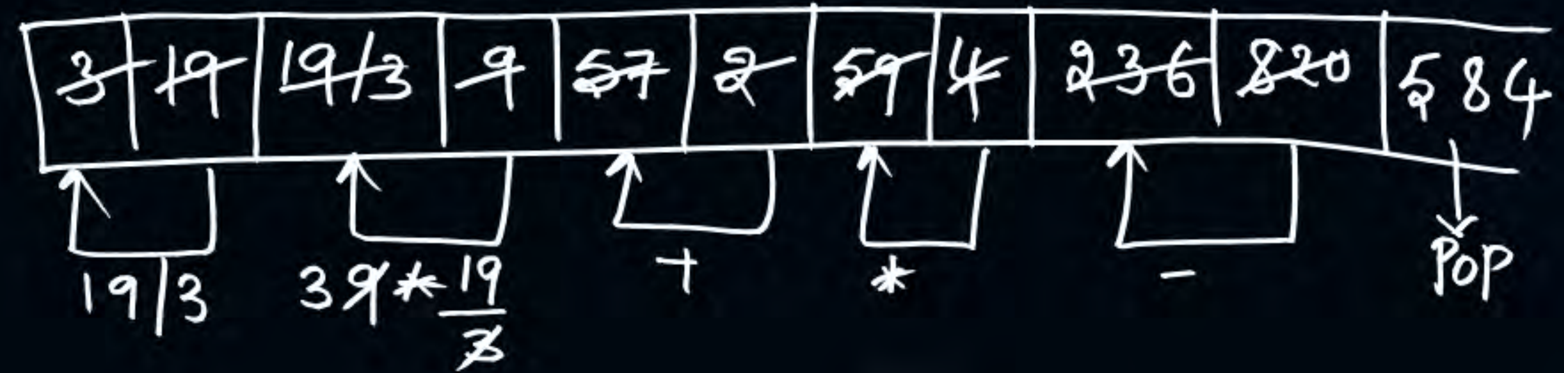
- Stack Using Queue (simple queue)

Expression Evaluation

— Postfix Exp

— Prefix Exp.





H/W: Evaluate

✓ Postfix Expression : 3 4 * 7 + 3 / 6 * 2 − 4 7 + * = 396

Prefix Expression : − 820 * 4 + 2 * 9 / 19 3 = 584

Topic : Stack Implementation

Stack (Last-In-first-out) : Insertion (Push), Deletion (Pop), Access (Peek)

overflow   Underflow

Stack can be Implemented using Python in either of 3 ways:

1) Using Lists (arrays)
2) Using Lifoqueue class
3) Using deque class

Topic : Stack Implementation

Drawback: Memory wastage : Internal fragmentation

In other lang : Arrays ⇒ Static Memory

In Python : Arrays (lists) ⇒ Dynamic Memory (No Need of checking overflow while Push)

\# Stack implementation in python Using Lists

[Contiguous memory Block-wise]

1 | 2 | 3 | 4

```
# Creating a stack
def create_stack():
    stack = []
    return stack
```

Push Time Complexity: O(1)

Pop Time Complexity: O(1)

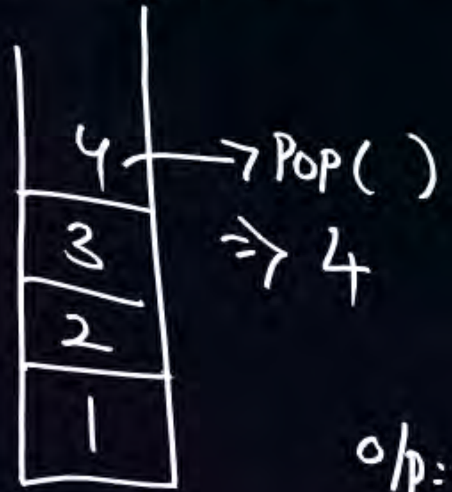
Peek Time Complexity: O(1)

```
# Creating an empty stack
def check_empty(stack):
    return len(stack) == 0
```

returns True if Empty, False otherwise

```
# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("pushed item: " + item)
```

→ will Insert Element at the top of Stack.




```
# Removing an element from the stack
def pop(stack):
    if (check_empty(stack)):
        return "stack is empty"
    return stack.pop()
~~stack = create_stack()~~
```

But, In Pop() or Peek(), Empty Condition need to be verified.

Driver code

```
stack = create_stack()      [ ] list
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
print("popped item: " + pop(stack))      4
print("stack after popping an element: " + str(stack))
```

o/p: Popped item : 4, Stack after Popping 1 2 3

Topic : Stack Implementation

Executable form of Program : Process

Thread : Light weight Process

Not Suitable for multithreading applications

```
from collections import deque
```
(Random memory allocation – No Internal fragmentation)

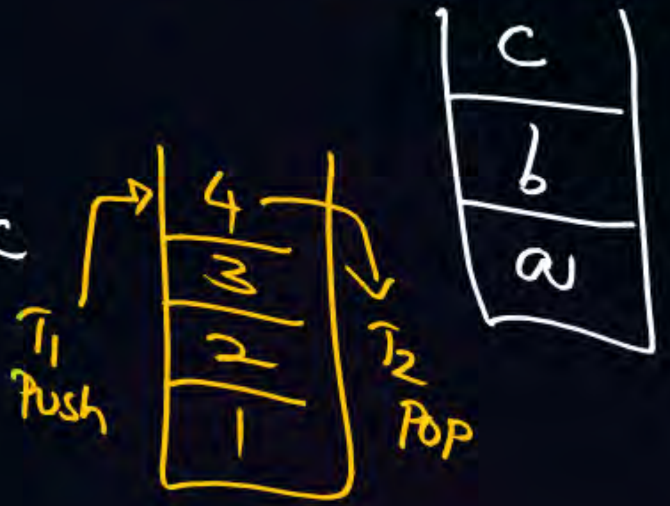
```
my_stack = deque() # constructor

# append() function is used to push
# element in the my_stack
my_stack.append('a')
my_stack.append('b')
my_stack.append('c')

print('Initial my_stack:')
print(my_stack) # a b c
```

Recursive Process

T1, T2, T3, T4, T5

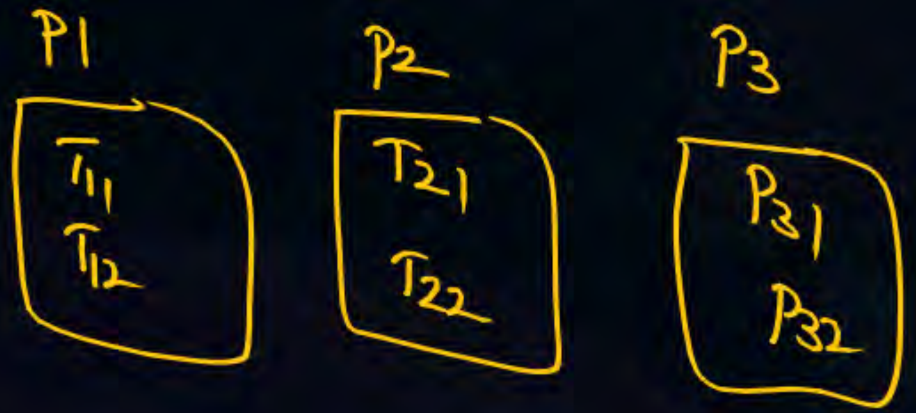
$\boxed{f(5) \rightarrow f(4) \rightarrow f(3) \rightarrow f(2) \rightarrow f(1)}$

Count : Race Condition | data Inconsistency




```
print('\nElements poped from my_stack:')
print(my_stack.pop()) # c
print(my_stack.pop()) # b
print(my_stack.pop()) # a

print('\nmy_stack after elements are poped:')
print(my_stack) # Empty, Nothing is Printed.
```

T.C : Push | Pop | Peek : $O(1)$

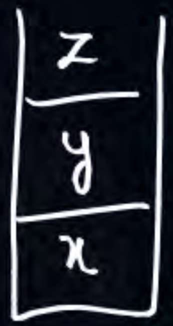


when thread of different Process access Memory – No issue.

Topic : Stack Implementation

qsize() returns Number of Elements Present

```
from queue import LifoQueue (Handle Race Condition issue, with Synchronization Tools)

my_stack = LifoQueue(maxsize = 5)

print(my_stack.qsize()) # 0

my_stack.put('x')
my_stack.put('y')
my_stack.put('z')

print("Stack is Full: ", my_stack.full()) # Stack is Full : False
print("Size of Stack: ", my_stack.qsize()) # 3
```




```
print('\nElements poped from the my_stack')
print(my_stack.get()) # z
print(my_stack.get()) # y
print(my_stack.get()) # x

print("\nStack is Empty: ", my_stack.empty())
    # Stack is Empty : True
```

T.C : Push | Pop | Peek() : O(1)

Topic : Stack Implementation

Stack Implementation

- Using Lists : Contiguous memory allocation, Memory wastage (or) Less Utilization

- Using deque : Not Suitable for multithreading applications, but fast in performing operations

- Using LifoQueue : Handle multithreading Effectively, but slow compared with deque Implementation.

Topic : Stack Implementation

(LIFO)

Stack Implementation Using Simple Queue (FIFO)

It uses 2 Queues for Implementing stack Nature

It can be Performed in either of 2 ways:

1) By making Push operation Costly → Push : $O(n)$, Pop : $O(1)$

2) By making Pop operation costly → Push : $O(1)$, Pop : $O(n)$

Topic : Stack Implementation

Let Q1, Q2 be 2 Queues, S be stack

Push(10), Push(20), Push(30), POP(), Push(40), POP(), POP(), POP()

Expected o/p : 30, 40, 20, 10

Push costly : Push : O(n)

1) dequeue all elements from Q1 } shift from
2) Enqueue them into Q2 } Q1 to Q2
3) enqueue new Element into Q1
4) shift Elements back from Q2 to Q1

Pop Operation Pop : O(1)

dequeue from Q1.
and Print it.

Push(10)
Q1 | 10 |
Q2 | |

Push(20)
1. Q1→Q2 Q1 | | Q2 | 10 |
2. Q1←20 Q1 | 20 | Q2 | 10 |
3. Q2→Q1 Q1 | 20 | 10 | Q2 | |

Push(30)
1. Q1→Q2 Q1 | | Q2 | 20 | 10 |
2. Q1←30 Q1 | 30 | Q2 | 20 | 10 |
3. Q2→Q1
Q1 | 30 | 20 | 10 |
Q2 | |

POP()
dequeue from Q1 and Print
$\boxed{\text{O/p: 30}}$
Q1 | 20 | 10 | Q2 | |

Push(40)
1. Q1→Q2 Q1 | | Q2 | 20 | 10 |
2. 40→Q1 Q1 | 40 | Q2 | 20 | 10 |
3. Q2→Q1 Q1 | 40 | 20 | 10 | Q2 | |

POP()
dequeue from Q1 and Print
$\boxed{\text{O/p: 40}}$ Q1 | 20 | 10 | Q2 | |

POP()
dequeue from Q1 and Print
$\boxed{\text{O/p: 20}}$ Q1 | 10 | Q2 | |

POP()
Dequeue from Q1 and Print
$\boxed{\text{O/p: 10}}$ Q1 | | Q2 | |

Actual o/p sequence

30, 40, 20, 10 == Expected o/p.

2 mins Summary

Push(10), Push(20), Push(30), Pop(), Push(40), Pop(), Pop(), Pop()

Expected o/p: 30, 40, 20, 10

Pop operation Costly :

Push() : enqueue Element into Q1

O(1)

Pop() operation O(n)

1. Shift (n-1) Elements from Q1 to Q2
2. Dequeue last Element from Q1 and Print
3. Shift (n-1) Elements back to Q1 from Q2

Q1 [ ] Q2 [ ]

Push(10)

Q1 [10] Q2 [ ]

Push(20)

Q1 [10|20] Q2 [ ]

Push(30)

Q1 [10|20|30] Q2 [ ]

Pop()

1. Q1→Q2 Q1 [30] Q2 [10|20]
2. Dequeue from Q1 and Print $\boxed{\text{o/p: 30}}$ ✓
3. Q2→Q1 Q1 [10|20] Q2 [ ]

Push(40)

Q1 [10|20|40] Q2 [ ]

Pop()

1. Q1 [40] Q2 [10|20]
2. $\boxed{\text{o/p: 40}}$ ✓
3. Q1 [10|20] Q2 [ ]

Pop()

1. Q1 [20] Q2 [10]
2. $\boxed{\text{o/p: 20}}$ Q1 [ ] Q2 [10]
3. Q1 [10] Q2 [ ]

Pop() $\boxed{\text{o/p: 10}}$ Q1 [ ] Q2 [ ]

Actual o/p: 30, 40, 20, 10

THANK - YOU