**Stack**

A stack is a linear data structure in which insertions and deletions are allowed only at the end, called the top of the stack. (LIFO last in first out)

1. Any object can be accessed only from the top.

2. Any object can be added only at the top.

Primary Stack Operations:

push(data): Inserts data onto the stack.

pop (): Deletes the last inserted element from the stack.

SECONDARY STACK OPERATIONS:

Top/peek(): returns the last inserted element without removing it.

size(): returns the size or the number of elements in the stack.

isEmpty() : returns TRUE if the stack is empty otherwise it returns FALSE.

isFu11(): returns TRUE if the stack is full, otherwise it returns FALSE.

**Basic Stack program:**

class Stack:  
 def \_\_init\_\_(self):  
 self.stack = []  
 def push(self,value): #adds an element to the top of the stack  
 self.stack.append(value)  
 def pop(self): #removes the top element of the stack  
 if len(self.stack) ==0:  
 return "Stack is empty"  
 return self.stack.pop()  
 def peek(self): #returns the top element of the stack  
 if len(self.stack) ==0:  
 return "Stack is empty"  
 return self.stack[-1]  
 def size(self): #returns the size of the stack  
 return len(self.stack)  
 def is\_empty(self): #returns True if the stack is empty, False otherwise  
 if len(self.stack) ==0:  
 return True  
 return False  
 def is\_full(self): #returns True if the stack is full, False otherwise  
  
 return False #stack is never full in python because of the dynamic nature of the list  
 def \_\_str\_\_(self): #returns the elements of the stack  
 return str(self.stack)  
  
stack = Stack()  
stack.push(1)  
stack.push(2)  
stack.push(3)  
print(stack)  
print(stack.pop())  
print(stack.is\_full())  
  
  
# **time complexity** of the operations is O(1) because we are using a list to implement the stack but for \_\_str\_\_ it is O(n) where n is the size of the stack