stack: is a linear data structure , perform the O(1) and also insert and delete.

implement by using "list" only.

arr.append()

arr.pop

arr.insert

application:undo/redo,memory management,block chain,recursion,

stack follows the "LIFO".last in first out

\*push

\*pop

\*size

\*is empty

stack=[]

stack.append(10)

stack.append(20)

stack.append(30)

def push():

top=-1

if(top==len(stack)-1):

return"stack is full"

else:

return stack.append(40)

program:

def push(value):

top=-1

if(top==4):

return"stack is full"

else:

top=top+1

return stack.append(value)

stack=[10] #size as 5

push(20)

push(30)

push(40)

push(50)

push(60)

print(stack)

STACK:

stack=[]

top=-1

def push(value):

global top

stack.append(value)

top +=1

def pop():

global top

if top==-1:

priny("Stack is empty.nothing to pop.")

return

else:

stack.pop()

top-=1

def peek():

if top ==-1:

return "Stack is empty.No top element."

else:

return f"top element={stack[top]}"

def display():

if(top==-1):

print("empty")

else:

for i in range(top,-1,-1):

print(stack[i])

top=-1

push(10)

push(30)

push(50)

push(70)

pop()

pop()

print(peek())

display()

O/P:

top element=30

30

10

stack=[]

top=-1

def push(value):

global top

stack.append(value)

top +=1

def pop():

global top

if top==-1:

priny("Stack is empty.nothing to pop.")

return

else:

stack.pop()

top-=1

def peek():

if top ==-1:

return "Stack is empty.No top element."

else:

return f"top element={stack[top]}"

def display():

if(top==-1):

print("empty")

else:

for i in range(top,-1,-1):

print(stack[i])

while True:

print("\n---welcome--")

print("1.push")

print("2.pop")

print("3.peek")

print("4.display all the elements")

print("5.exit")

choise=int(input("Enter your choise"))

if choise==1:

value=int(input("Enter the element"))

push(value)

elif choise==2:

pop()

elif choise==3:

print(peek())

elif choise==4:

display()

else:

print("exit")

break

QUEUE:

is alinear data structure follows the FIFO

APLICATIONS:

1.ROUND ROBIN

2.JOB SCHEDULING

REFERENCE VARIABLES:

1.front

2.rear

TYPES:

1.Enqueue

2.Dequeue

program:

class queue:

def \_init\_(self,Q,value):

self.Q=Q

self.value=value

def enqueue(self,Q,value):

return Q.append(value)

QEUE:

class Queue:

def \_init\_(self):

self.items = deque()

def is\_empty(self):

return len(self.items) == 0

def enqueue(self, item):

self.items.append(item)

def dequeue(self):

if not self.is\_empty():

return self.items.popleft()

else:

return "Queue is empty"

def peek(self):

if not self.is\_empty():

return self.items[0]

else:

return "Queue is empty"

def display(self):

if self.is\_empty():

print("Queue is empty")

else:

print("Queue elements:")

for item in self.items:

print(item, end=" ")

print()

# Example usage

queue = Queue()

queue.enqueue(1)

queue.enqueue(2)

queue.enqueue(3)

queue.display() # Output: Queue elements: 1 2 3

print("Front element:", queue.peek()) # Output: Front element: 1

queue.dequeue()

queue.display() # Output: Queue elements: 2 3