1. class Sudoku:

def \_\_init\_\_(self,in\_string):

self.board = []

for ele in range(0,len(in\_string),9):

self.board.append([int(string) for string in in\_string[ele:ele+9]])

def get\_row(self,row):

return self.board[row]

def get\_col(self,column):

output = []

for ele in self.board:

output.append(ele[column])

return output

def get\_sqr(self,row,column=None):

output = []

square\_to\_cell = {

0 : [[0,1,2],[0,1,2]],

1 : [[0,1,2],[3,4,5]],

2 : [[0,1,2],[6,7,8]],

3 : [[3,4,5],[0,1,2]],

4 : [[3,4,5],[3,4,5]],

5 : [[3,4,5],[6,7,8]],

6 : [[6,7,8],[0,1,2]],

7 : [[6,7,8],[3,4,5]],

8 : [[6,7,8],[6,7,8]]

}

if column == None:

for i in square\_to\_cell[row][0]:

for j in square\_to\_cell[row][1]:

output.append(self.board[i][j])

else:

square = None

if row <= 2:

square = 0 if column <=2 else 1 if column <=5 else 2

elif row <= 5:

square = 3 if column <=2 else 4 if column <=5 else 5

else:

square = 6 if column <=2 else 7 if column <=5 else 8

for i in square\_to\_cell[square][0]:

for j in square\_to\_cell[square][1]:

output.append(self.board[i][j])

return (output)

game = Sudoku("417950030000000700060007000050009106800600000000003400900005000000430000200701580")

display(game.board)

print(f'game.get\_row(0) ➞ {game.get\_row(0)}')

print(f'game.get\_col(8) ➞ {game.get\_col(8)}')

print(f'game.get\_sqr(1) ➞ {game.get\_sqr(1)}')

print(f'game.get\_sqr(1,8) ➞ {game.get\_sqr(1,8)}')

print(f'game.get\_sqr(8,3) ➞ {game.get\_sqr(8,3)}')

Output:

[[4, 1, 7, 9, 5, 0, 0, 3, 0],

[0, 0, 0, 0, 0, 0, 7, 0, 0],

[0, 6, 0, 0, 0, 7, 0, 0, 0],

[0, 5, 0, 0, 0, 9, 1, 0, 6],

[8, 0, 0, 6, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 3, 4, 0, 0],

[9, 0, 0, 0, 0, 5, 0, 0, 0],

[0, 0, 0, 4, 3, 0, 0, 0, 0],

[2, 0, 0, 7, 0, 1, 5, 8, 0]]

game.get\_row(0) ➞ [4, 1, 7, 9, 5, 0, 0, 3, 0]

game.get\_col(8) ➞ [0, 0, 0, 6, 0, 0, 0, 0, 0]

game.get\_sqr(1) ➞ [9, 5, 0, 0, 0, 0, 0, 0, 7]

game.get\_sqr(1,8) ➞ [0, 3, 0, 7, 0, 0, 0, 0, 0]

game.get\_sqr(8,3) ➞ [0, 0, 5, 4, 3, 0, 7, 0, 1]

1. class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

self.num\_list = []

self.num\_list.append(val)

def add\_data(self,in\_list):

self.num\_list.extend(in\_list)

def get\_data(self):

return self.num\_list

class add\_two\_numbers:

def \_\_init\_\_(self,ob1,ob2):

self.ob1 = ob1

self.ob2 = ob2

def get\_data(self):

it1 = ''.join([str(ele) for ele in self.ob1.get\_data()[::-1]])

it2 = ''.join([str(ele) for ele in self.ob2.get\_data()[::-1]])

return [int(x) for x in str(int(it1)+int(it2))[::-1]]

lt1 = ListNode(2)

lt1.add\_data([4, 3])

lt2 = ListNode(5)

lt2.add\_data([6, 4])

print(f'lt1.get\_data() ➞ {lt1.get\_data()}')

print(f'lt2.get\_data() ➞ {lt2.get\_data()}')

print(f'add\_two\_numbers(lt1, lt2).get\_data() ➞ {add\_two\_numbers(lt1, lt2).get\_data()}',end='\n\n')

lt1 = ListNode(0)

lt2 = ListNode(0)

print(f'lt1.get\_data() ➞ {lt1.get\_data()}')

print(f'lt2.get\_data() ➞ {lt2.get\_data()}')

print(f'add\_two\_numbers(lt1, lt2).get\_data() ➞ {add\_two\_numbers(lt1, lt2).get\_data()}',end='\n\n')

lt1 = ListNode(9)

lt1.add\_data([9,9,9,9,9,9])

lt2 = ListNode(9)

lt2.add\_data([9,9,9])

print(f'lt1.get\_data() ➞ {lt1.get\_data()}')

print(f'lt2.get\_data() ➞ {lt2.get\_data()}')

print(f'add\_two\_numbers(lt1, lt2).get\_data() ➞ {add\_two\_numbers(lt1, lt2).get\_data()}')

Output:

lt1.get\_data() ➞ [2, 4, 3]

lt2.get\_data() ➞ [5, 6, 4]

add\_two\_numbers(lt1, lt2).get\_data() ➞ [7, 0, 8]

lt1.get\_data() ➞ [0]

lt2.get\_data() ➞ [0]

add\_two\_numbers(lt1, lt2).get\_data() ➞ [0]

lt1.get\_data() ➞ [9, 9, 9, 9, 9, 9, 9]

lt2.get\_data() ➞ [9, 9, 9, 9]

add\_two\_numbers(lt1, lt2).get\_data() ➞ [8, 9, 9, 9, 0, 0, 0, 1]

1. Menu = [

{'name':'Orange Juice','type':'drink','price':25.50},

{'name':'Lemonade','type':'drink','price':10},

{'name':'Cranberry Juice','type':'drink','price':40},

{'name':'Pineapple Juice','type':'drink','price':40},

{'name':'Lemon Iced Tea','type':'drink','price':80},

{'name':'Vanilla Chai Latte','type':'drink','price':90},

{'name':'Hot Choclate','type':'drink','price':100},

{'name':'Iced Cofee','type':'drink','price':70.12},

{'name':'Tuna Sandwich','type':'food','price':120},

{'name':'Ham Cheese Sandwich','type':'food','price':180},

{'name':'Bacon And Egg','type':'food','price': 120},

{'name':'Chicken Biryani','type':'food','price':360},

{'name':'Veg Burger','type':'food','price':90},

{'name':'Cinnamon Roll','type':'food','price':60.25}

]

class Cofeeshop:

def \_\_init\_\_(self,name,menu,orders):

self.name = name

self.menu = menu

self.orders = orders

def add\_order(self, order\_name):

available\_items = [item['name'].lower() for item in self.menu]

if order\_name in available\_items:

output = "Order added!"

self.orders.append(order\_name)

else:

output = "This item is currently unavailable!"

return output

def list\_orders(self):

return self.orders

def due\_amount(self):

output = sum([item['price'] for item in self.menu if item['name'].lower() in self.orders])

return output

def fulfill\_order(self):

if len(self.orders) > 0:

output = f'The {self.orders.pop(0)} is ready!'

else:

output = 'All orders have been fulfilled!'

return output

def cheapest\_item(self):

lowest\_price = min([item['price'] for item in self.menu])

output = [item['name'] for item in self.menu if item['price'] == lowest\_price]

return output[0]

def drinks\_only(self):

output = [item['name'] for item in self.menu if item['type'] == 'drink']

return output

def food\_only(self):

output = [item['name'] for item in self.menu if item['type'] == 'food']

return output

tcs = Cofeeshop('Tesha\'s Cofee Shop',Menu,[])

print(f'tcs.add\_order("hot cocoa") ➞ {tcs.add\_order("hot cocoa")}')

print(f'tcs.add\_order("iced tea") ➞ {tcs.add\_order("iced tea")}')

print(f'tcs.add\_order("cinnamon roll") ➞ {tcs.add\_order("cinnamon roll")}')

print(f'tcs.add\_order("iced cofee") ➞ {tcs.add\_order("iced cofee")}')

print(f'tcs.list\_orders() ➞ {tcs.list\_orders()}')

print(f'tcs.due\_amount() ➞ {tcs.due\_amount()}')

print(f'tcs.fulfill\_order() ➞ {tcs.fulfill\_order()}')

print(f'tcs.fulfill\_order() ➞ {tcs.fulfill\_order()}')

print(f'tcs.fulfill\_order() ➞ {tcs.fulfill\_order()}')

print(f'tcs.list\_orders() ➞ {tcs.list\_orders()}')

print(f'tcs.due\_amount() ➞ {tcs.due\_amount()}')

print(f'tcs.cheapest\_item() ➞ {tcs.cheapest\_item()}')

print(f'tcs.food\_only() ➞ {tcs.food\_only()}')

print(f'tcs.drinks\_only() ➞ {tcs.drinks\_only()}')

Output:

tcs.add\_order("hot cocoa") ➞ This item is currently unavailable!

tcs.add\_order("iced tea") ➞ This item is currently unavailable!

tcs.add\_order("cinnamon roll") ➞ Order added!

tcs.add\_order("iced cofee") ➞ Order added!

tcs.list\_orders() ➞ ['cinnamon roll', 'iced cofee']

tcs.due\_amount() ➞ 130.37

tcs.fulfill\_order() ➞ The cinnamon roll is ready!

tcs.fulfill\_order() ➞ The iced cofee is ready!

tcs.fulfill\_order() ➞ All orders have been fulfilled!

tcs.list\_orders() ➞ []

tcs.due\_amount() ➞ 0

tcs.cheapest\_item() ➞ Lemonade

tcs.food\_only() ➞ ['Tuna Sandwich', 'Ham Cheese Sandwich', 'Bacon And Egg', 'Chicken Biryani', 'Veg Burger', 'Cinnamon Roll']

tcs.drinks\_only() ➞ ['Orange Juice', 'Lemonade', 'Cranberry Juice', 'Pineapple Juice', 'Lemon Iced Tea', 'Vanilla Chai Latte', 'Hot Choclate', 'Iced Cofee']

1. def loneliest\_number(start,end):

prime\_list = []

output = {'number': 0, 'distance': 0, 'closest': 0}

temp = []

if start <=3: prime\_list.extend([2,3])

for ele in range(start,end+1):

if (ele-1)%6 == 0 or (ele+1)%6 == 0: prime\_list.append(ele) # initial check

for ele in prime\_list:

for item in range(2,ele):

if ele%item == 0 :

temp.append(ele)

break

prime\_list = sorted(set(prime\_list)-set(temp))

if start in [4,5] : print(3) ; prime\_list.insert(0,3) # Logic to get first prime number before start

else:

for ele in range(start-1,0,-1):

if (ele-1)%6 == 0 or (ele+1)%6 == 0:

prime\_list.insert(0,ele)

break

while True: # Logic to get first prime number after end

if (end-1)%6 == 0 or (end+1)%6 == 0:

out\_num = None

for ele in range(2,end):

if end%ele == 0:

out\_num = ele

break

if out\_num == None:prime\_list.append(end) ; break

else: end +=1

else:

end+=1

if 1 in prime\_list: prime\_list.remove(1)

for ele in range(start,end):

org\_ele = ele

while True:

if ele in prime\_list:

n\_f\_prime = ele if ele != org\_ele else prime\_list[prime\_list.index(ele)+1]

n\_b\_prime = prime\_list[prime\_list.index(ele)-1] if prime\_list.index(ele) > 0 else 0

closest\_distance = min(org\_ele-n\_b\_prime,n\_f\_prime-org\_ele) if n\_b\_prime !=0 else n\_f\_prime-org\_ele

closest\_prime = n\_f\_prime if n\_b\_prime == 0 or closest\_distance == n\_f\_prime-org\_ele else n\_b\_prime

if output['distance'] < closest\_distance:

output = {'number': org\_ele, 'distance': closest\_distance, 'closest': closest\_prime}

break

else:

ele +=1

print(f'loneliest\_number{start,end} ➞ {output}')

loneliest\_number(0,22)

loneliest\_number(8, 123)

loneliest\_number(938, 1190)

loneliest\_number(120, 1190)

Output:

loneliest\_number(0, 23) ➞ {'number': 0, 'distance': 2, 'closest': 2}

loneliest\_number(8, 127) ➞ {'number': 120, 'distance': 7, 'closest': 127}

loneliest\_number(938, 1193) ➞ {'number': 1140, 'distance': 11, 'closest': 1151}

loneliest\_number(120, 1193) ➞ {'number': 211, 'distance': 12, 'closest': 223}

1. class Selfie:

def \_\_init\_\_(self,x=None):

self.x = x

self.archived\_states = []

def save\_state(self):

self.archived\_states.append(self.x)

self.x = None

return self.archived\_states

def recover\_state(self,in\_num):

if in\_num >=0 and in\_num <= len(self.archived\_states):

self.x = self.archived\_states[in\_num]

return self

p = Selfie()

p.x = 2

print(f'p.\_\_dict\_\_ ➞ {p.\_\_dict\_\_}')

p.save\_state()

print(f'p.\_\_dict\_\_ ➞ {p.\_\_dict\_\_}')

p.x = 5

print(f'p.\_\_dict\_\_ ➞ {p.\_\_dict\_\_}')

p.save\_state()

print(f'p.\_\_dict\_\_ ➞ {p.\_\_dict\_\_}')

p = p.recover\_state(0)

print(f'p.\_\_dict\_\_ ➞ {p.\_\_dict\_\_}')

print(f'p.x ➞ {p.x}')

Output:

p.\_\_dict\_\_ ➞ {'x': 2, 'archived\_states': []}

p.\_\_dict\_\_ ➞ {'x': None, 'archived\_states': [2]}

p.\_\_dict\_\_ ➞ {'x': 5, 'archived\_states': [2]}

p.\_\_dict\_\_ ➞ {'x': None, 'archived\_states': [2, 5]}

p.\_\_dict\_\_ ➞ {'x': 2, 'archived\_states': [2, 5]}

p.x ➞ 2