**Documentation BWINF 2021 Round 1 Task 1: Sliding Parking Lot**

Problem....................................................................................................................................2

Solution idea.............................................................................................................................4

Implementation.........................................................................................................................6

Examples..................................................................................................................................9

Code........................................................................................................................................13

**Problem**

There is a Parking Lot with cars in every slot and cars which are blocking the Parking Lot in some positions. One car has the length 2, so one blocking car can block two cars in the Parking Lot. The code should get as input:

* In the first line, the character of the first and the last car in the Parking Lot. For example: A, D => A, B, C, D.
* In the second line, the number *n*, which is the number of blocking cars in front.
* In the following lines are first the character from a blocking car and his left position.

For example:

1 A G

2 2

3 H 2

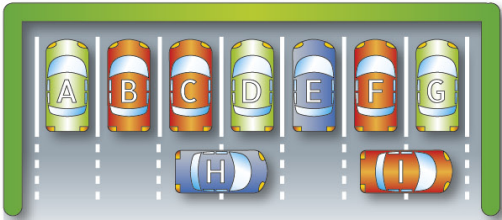
4 I 5

Line 1 -> cars in slot = {“*A*”, “*B*”, “*C*”, “*D*”, “*E*”, “*F*”, “*G*”}

Line 2 -> number of blocking cars *n* = 2

Line 3 -> blocking car one = H; positions: {2, 3}

Line 4 -> blocking car two = I; positions: {5, 6}



The code should give as output for every car in the slots, which blocking cars need to move how many steps and the direction, that the car can go out.

For example:

1 A: E 1 right

2 B: E 1 left

3 C:

4 D:

**Solution idea**

First, I need to get the data from the file and separate the data in the constants CARS\_IN\_PAKRSLOT and BLOCKING\_CARS.

CARS\_IN\_PARKSLOT is a list of all cars in the Parking Lot and BLOCKING\_CARS is a dictionary of all blocking cars in front of the Parking Lot with their character as key and their positions as their values.

Then I create a new constants list, CARS\_NEED\_MOVES. I iterate through CARS\_IN\_PARKSLOT and when one car is blocked it´s add to the list.

I iterate through CARS\_IN\_PARKSLOT and when the car needs no move I add it directly to the result list as:

*f*“*{*car*}*: ”

But if the car is in CARS\_NEED\_MOVES, I need to collect the moves, which are needed for this car to get out of his slot. First, I have to check if only one move is needed or multiple moves. For that I check if on one side of the directly blocking car is space, so the car, can move out of the way, with one move. If this is possible, I also get the direction the directly blocking car needs to move. After that, I have to find the directly blocking car and add it to the result list as:

*f*“*{*car*}*: *{*blocking\_car*}* 1 *{*direction*}*”

This only works if the cars are in these positions:

*—B—* *or* *-B—*

*—CC—* *CC—*

-> B: C 1 right -> B: C 1 left

But when one move is not possible and multiple moves are needed, I have to do more.

I get the moves needed per side for the directly blocking car to move out of the way.

For example: [1, 2]

-> This means that I need 1 free space on the left or 2 free spaces on

the right side of the directly blocking car

Then I go 1 more position to the left and right and check if there is space. If there is space, I change the moves needed per side. When one side is 0 I return the range where I need to move the cars and their direction.

After that, I collect the blocking cars in the range and look for their moves. The first car, which needs to move, always moves as far as he can (up to 2).

Finally, I add every one of them with their moves to the result and print the result list line after line.

**Implementation**

I separated my project in four files:

* *main.py*
* *get\_data.py*
* *solve\_sliding\_parking\_lot.py*
* *test\_code.py*

1) main.py:

This is my main file, which needs to be start to run the project.

2) get\_data.py:

This file get the data from the selected file (all examples in folder files) and returns CARS\_IN\_PARKSLOTS and BLOCKING\_CARS.

3) solve\_sliding\_parking\_lot.py:

This file solves the problem and has the logic in it.

4) test\_code.py:

Tests the code with 30 tests.

——————————————————————————————————————

Main file gets CARS\_IN\_PARKSLOTS and BLOCKING\_CARS from the static method get\_data\_from\_file\_return\_necessary\_data from the class GetData. This static method gets as parameter the file\_path by the main file. The data in the file is saved by the variable data. CARS\_IN\_PARKSLOTS and BLOCKING\_CARS were returned by this method and created by data.

CARS\_IN\_PARKSLOTS is created by a list comprehension. It goes through the alphabet and add the char up to the last car char and stops then. This is accomplished by *takewhile* from the build in package *itertools*:

CARS\_IN\_PARKSLOTS = [car for car in

takewhile(lambda car: car != alphabet[last\_car\_index], alphabet)]

BLOCKING\_CARS is a dictionary with blocking\_cars char as keys and their positions as values in a list:

BLOCKING\_CARS = {cars[0]: [int(cars[1:]), int(cars[1:])+1]

for cars in data[2:]}

After collecting these two constants the main file generates an object of the class SolveSlidingParkingLot with CARS\_IN\_PARKSLOTS and BLOCKING\_CARS as parameters of the constructor.

In the constructor the attributes self.CARS\_IN\_PARKSLOTS and self.BLOCKING\_CARS are getting their values by the two parameters. In \_\_init\_\_ multiple attributes are getting also their values:

* self.BLOCKING\_CARS\_POSITIONS is a list with all blocked positions by the self.BLOCKING\_CARS.
* self.CARS\_NEED\_MOVES is a list created by list comprehension and is a collection of all self.CARS\_IN\_PARKSLOTS, which are blocked by self.BLOCKING\_CARS.
* self.result is unfilled list, for the final result.
* self.car\_position is the position of the current car, which the algorithm is trying to find moves, so the car can go out of the slot.
* self.parkslot\_border is a list with the positions of the end of the park slot.

After that, the main function will call the method assign\_moves\_to\_cars.

Here, a for loop iterate through self.CARS\_IN\_PARKSLOTS and checks if the current car is in self.CARS\_NEED\_MOVES. If not it gets directly added to the result as: f"{car}:"

But if the current car is blocked the position of the current car gets saved in the variable self.car\_position.

The method check\_if\_car\_needs\_one\_or\_multiple\_moves gets called and checks if one or multiple moves are needed, that the current car can go out of the slot.

If the directly right slot is blocked, and two slots right is also free and it´s not out of self.parkslot\_border, then it returns "right".

Else, if the directly left slot is blocked and two slots left is free and it´s not out of self.parkslot\_border, then it returns "left".

Else it returns "multiple".

If the return value of the method is not "multiple", the directly blocking car is determined by the method one\_move\_get\_blocking\_car. The method iterate through all self.BLOCKING\_CARS and checks if the self.car\_position is in the positions of the blocking car, if this is true the method returns the directly blocking car. After that, it gets added to the result as:

f"{car}: {blocking\_car} 1 {move\_status}".

But if multiple moves are needed the multiple\_moves method gets called. This method coordinated other methods and their return values to get the final result for the current car to get out the slot.

First it calls get\_moves\_needed\_per\_side which return the moves needed per side for the directly blocking car to get out of the way of the slot.

It returns either [1, 2] or [2, 1]. The first digit is the moves for the left and the second the moves for the right side to get out of the way. This is accomplished by checking if the directly blocking car starts at the position of the current blocked car or if it ends their. Also the method returns the result twice. The second one is the backup, which is needed in the next method. This next method is find\_range\_of\_blocking\_cars\_and\_side.

In an infinite while loop gets every loop the position\_changer value increased by one. Every loop self.car\_position gets increased and decrease by the position\_changer to check if their is a free space on the left or right side. If there is a free space and it´s inside of the self.parkslot\_border then moves\_needed\_per\_side the return of the method before gets decreased on the side, with the free space. If one side is zero, so all free spaces are enough to get the directly blocking car out of the way, the loop breaks by returning the range, where all blocking cars need to move, the direction they need to move and the moves the directly blocking car needs to do. The last value is return by the second return of the method before, this was the backup for this return.

The last step is to get the blocking cars and their moves. Now the method get\_multiple\_cars\_need\_to\_move gets called. Now the directly\_blocking\_car

is created. And the blocking\_cars is filled with all cars which need to move. This list is created by a list comprehension which iterate through the blocking cars and checks if the positions of the car are in range\_of\_blocking\_cars\_and\_side[0] and that the car is on the right side of the car and is not the directly blocking car. After that, the list gets cleaned, by removing the double cars and order the list in the right order.

Then a for loop iterate through the list and assign each blocking car to 1 or 2 moves, based on the space the car can get. But it´s always the highest possible move and the current blocking car is getting add to result.

At the end, the directly blocking car is getting added to the list, with his moves.

Then the result gets returned and added to self.result.

Finally, in mail file the main function iterate through the result and print for each car the result.

**Examples**

parkplatz00:

A:

B:

C: H 1 right

D: H 1 left

E:

F: H 2 left, I 2 left

G: I 1 left

parkplatz01:

A:

B: Q 2 right, P 2 right, O 1 right

C: O 1 left

D: P 1 right

E: O 1 left, P 1 left

F:

G: Q 1 right

H: Q 1 left

I:

J:

K: R 1 right

L: R 1 left

M:

N:

parkplatz02:

A:

B:

C: O 1 right

D: O 1 left

E:

F: O 2 left, P 2 left

G: P 1 left

H: S 1 right, R 2 right, Q 1 right

I: O 2 left, P 2 left, Q 1 left

J: R 1 right

K: O 2 left, P 2 left, Q 1 left, R 1 left

L:

M: O 2 left, P 2 left, Q 1 left, R 1 left, S 2 left

N: S 1 left

parkplatz03:

A:

B: O 1 right

C: O 1 left

D:

E: P 1 right

F: P 1 left

G:

H:

I: P 2 left, Q 2 left

J: Q 1 left

K: P 2 left, Q 2 left, R 2 left

L: Q 2 left, R 1 left

M: P 2 left, Q 2 left, R 1 left, S 2 left

N: Q 2 left, R 1 left, S 1 left

parkplatz04:

A: R 2 right, Q 1 right

B: S 2 right, R 2 right, Q 2 right

C: R 1 right

D: S 2 right, R 2 right

E:

F:

G: S 1 right

H: S 1 left

I:

J:

K: T 1 right

L: T 1 left

M:

N: U 1 right

O: U 1 left

P:

parkplatz05:

A:

B:

C: P 2 left

D: P 1 left

E: Q 1 right

F: R 2 right, Q 2 right

G:

H:

I: R 1 right

J: R 1 left

K:

L:

M: S 1 right

N: S 1 left

O:

**Code**

main.py:

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

from get\_data import GetData

from solve\_sliding\_parking\_lot import SolveSlidingParkingLot

\_\_author\_\_ = "Marius Wörfel"

\_\_email\_\_ = "raborogit@gmail.com"

\_\_status\_\_ = "Finished 06/07/2022"

def main() -> None:

CARS\_IN\_PARKSLOTS, BLOCKING\_CARS = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz0.txt")

solve\_sliding\_parking\_lot = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS, BLOCKING\_CARS)

solve\_sliding\_parking\_lot.assign\_moves\_to\_cars()

for car\_and\_result in solve\_sliding\_parking\_lot.result:

print(car\_and\_result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

get\_data.py:

from itertools import takewhile

class GetData():

@staticmethod

def get\_data\_from\_file\_return\_necessary\_data(file\_path: str) -> list[str] and dict[str: list[int]]:

"""

return CARS\_IN\_PARKSLOTS -> list of all cars in parkslots

return BLOCKING\_CARS -> dict of cars in the way of parkslots as keys and their positions in a list as their values

e.g: A: [1, 2] => car A is in front of parkslot 2 and 3 (because counting starts at 0)

"""

with open(file\_path, "r") as file:

data = file.read().split("\n")

data.pop(-1)

alphabet = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

last\_car = data[0][2]

last\_car\_index = [index+1 for index, char in enumerate(alphabet) if char == last\_car][0]

# add car to CARS\_IN\_PARKSLOTS as long it´s not the last car, if it´s the last car stop list comprehension

CARS\_IN\_PARKSLOTS = [car for car in takewhile(lambda car: car != alphabet[last\_car\_index], alphabet)]

# car: [position\_1, position\_2]

BLOCKING\_CARS = {cars[0]: [int(cars[1:]), int(cars[1:])+1]

for cars in data[2:]}

return CARS\_IN\_PARKSLOTS, BLOCKING\_CARS

solve\_sliding\_parking\_lot.py:

class SolveSlidingParkingLot():

def \_\_init\_\_(self, CARS\_IN\_PARKSLOTS: list[str], BLOCKING\_CARS: dict[str: [int, int]]) -> None:

self.CARS\_IN\_PARKSLOTS = CARS\_IN\_PARKSLOTS

self.BLOCKING\_CARS = BLOCKING\_CARS

self.BLOCKING\_CARS\_POSITIONS = [position for positions in self.BLOCKING\_CARS.values() for position in positions]

# add car to the list if car in parkslot is blocked by one car

self.CARS\_NEED\_MOVES = [car for car\_position, car in enumerate(self.CARS\_IN\_PARKSLOTS)

for blocked\_positions in self.BLOCKING\_CARS.values()

if car\_position in blocked\_positions]

self.result = []

self.car\_position = 0

self.parkslot\_border = [len(self.CARS\_IN\_PARKSLOTS), 0]

def assign\_moves\_to\_cars(self) -> None:

"""

assign each car to moves, which are needed, that the car can move out of the slot

e.g A: B 1 right => For car A, car B needs to be moves one time to the right

"""

for car in self.CARS\_IN\_PARKSLOTS:

if car not in self.CARS\_NEED\_MOVES:

self.result.append(f"{car}:")

continue

self.car\_position = [position for position, value in enumerate(self.CARS\_IN\_PARKSLOTS) if value == car][0]

# if only one move is needed, it contains the direction of the move ["left", "right"], else it´s "multiple"

move\_status = self.check\_if\_car\_needs\_one\_or\_multiple\_moves()

if move\_status == "multiple":

blocking\_cars = self.multiple\_moves()

self.result.append(f"{car}: {blocking\_cars}")

continue

blocking\_car = self.one\_move\_get\_blocking\_car()

self.result.append(f"{car}: {blocking\_car} 1 {move\_status}")

def check\_if\_car\_needs\_one\_or\_multiple\_moves(self) -> str:

"""

return if the selected car, needs multiple car moves to come out; if not return the direction the blocking car needs to be moved

"""

# one slot right blocked && two slots right free && two slots right inside parkslot border

if self.car\_position + 1 in self.BLOCKING\_CARS\_POSITIONS and self.car\_position + 2 not in self.BLOCKING\_CARS\_POSITIONS and self.car\_position + 2 < self.parkslot\_border[0]:

return "right"

# one slot left blocked && two slots left free && two slots left inside parkslot border

elif self.car\_position - 1 in self.BLOCKING\_CARS\_POSITIONS and self.car\_position - 2 not in self.BLOCKING\_CARS\_POSITIONS and self.car\_position - 2 >= self.parkslot\_border[1]:

return "left"

return "multiple"

def one\_move\_get\_blocking\_car(self) -> str:

"""

return for one move the blocking car, which needs to be moved

"""

return [car for car, current\_blocking\_car\_position in self.BLOCKING\_CARS.items() if self.car\_position in current\_blocking\_car\_position][0]

def multiple\_moves(self) -> str:

"""

return multiple moves and the cars, which needs to be moved

"""

moves\_needed\_per\_side, backup\_moves\_needed\_per\_side = self.get\_moves\_needed\_per\_side()

range\_of\_blocking\_cars\_and\_side = self.find\_range\_of\_blocking\_cars\_and\_side(moves\_needed\_per\_side, backup\_moves\_needed\_per\_side)

blocking\_cars = self.get\_multiple\_cars\_need\_to\_move(range\_of\_blocking\_cars\_and\_side)

return blocking\_cars

def get\_moves\_needed\_per\_side(self) -> list[int, int]:

"""

return moves needed per side that the selected car can go out and backup, which is the same

e.g if return [1, 2] => left side one or right side two spaces are needed

"""

blocking\_car\_positions = [blocked\_positions for blocked\_positions in self.BLOCKING\_CARS.values() if self.car\_position in blocked\_positions][0]

return [[1, 2], [1, 2]] if self.car\_position == blocking\_car\_positions[1] else [[2, 1], [2, 1]]

def find\_range\_of\_blocking\_cars\_and\_side(self, moves\_needed\_per\_side: list[int, int], backup\_moves\_needed\_per\_side: list[int, int]) -> list[int, str, int]:

"""

return if the side, where the cars, which needs to be moved are (left or right) and return up to which position these cars are and the moves the directly blocking car needs to do

e.g if return [1, "left", 1] => up to position 1 from the position of the current car on the left side, these cars and the directly blocking car needs to be moved, directly blocking car needs to move 1 field

"""

position\_changer = 0

while True:

# car\_position - position\_changer is free space && inside parkslot border (left)

if self.car\_position - position\_changer not in self.BLOCKING\_CARS\_POSITIONS and self.car\_position - position\_changer >= self.parkslot\_border[1]:

moves\_needed\_per\_side[0] -= 1

# car\_position + position\_changer is free space && inside parkslot border (right)

if self.car\_position + position\_changer not in self.BLOCKING\_CARS\_POSITIONS and self.car\_position + position\_changer < self.parkslot\_border[0]:

moves\_needed\_per\_side[1] -= 1

# if one side got enough space and is the fastest one

if moves\_needed\_per\_side[0] == 0 or moves\_needed\_per\_side[1] == 0:

return [self.car\_position - position\_changer - 1, "left", backup\_moves\_needed\_per\_side[0]] if moves\_needed\_per\_side[0] == 0 else [self.car\_position + position\_changer + 1, "right", backup\_moves\_needed\_per\_side[1]]

position\_changer += 1

def get\_multiple\_cars\_need\_to\_move(self, range\_of\_blocking\_cars\_and\_side: list[int ,str, int]) -> str:

"""

return the cars, which need to be moved

e.g "E 1 left, F 2 left" => car E needs to move 1 left and car F needs to move 2 left

"""

directly\_blocking\_car = [[car, positions] for car, positions in self.BLOCKING\_CARS.items()

for position in positions

if position == self.car\_position][0]

if range\_of\_blocking\_cars\_and\_side[1] == "left":

# add car to blocking\_cars if their positions in the range\_of\_blocking\_cars\_and\_side[0] && on the left side from the current car and their not the directly\_blocking\_car

blocking\_cars = [car for car, positions in self.BLOCKING\_CARS.items()

for position in positions

if position >= range\_of\_blocking\_cars\_and\_side[0] and position < self.car\_position and car != directly\_blocking\_car[0]]

else:

# add car to blocking\_cars if their positions in the range\_of\_blocking\_cars\_and\_side[0] && on the right side from the current car and their not the directly\_blocking\_car

blocking\_cars = [car for car, positions in self.BLOCKING\_CARS.items()

for position in positions

if position <= range\_of\_blocking\_cars\_and\_side[0] and position > self.car\_position and car != directly\_blocking\_car[0]]

car\_result = []

cleaned\_blocking\_cars = list(dict.fromkeys(blocking\_cars)) # remove double elements

if range\_of\_blocking\_cars\_and\_side[1] == "right":

cleaned\_blocking\_cars = cleaned\_blocking\_cars[::-1]

for blocking\_car in cleaned\_blocking\_cars:

if range\_of\_blocking\_cars\_and\_side[1] == "right":

# if one right field of the blocking\_car is free or two right fields are the parkslot border

if [positions for car, positions in self.BLOCKING\_CARS.items() if car == blocking\_car][0][1] + 1 in self.BLOCKING\_CARS\_POSITIONS or [positions for car, positions in self.BLOCKING\_CARS.items() if car == blocking\_car][0][1] + 2 >= self.parkslot\_border[0]:

car\_result.append(f"{blocking\_car} 1 {range\_of\_blocking\_cars\_and\_side[1]}, ")

continue

car\_result.append(f"{blocking\_car} 2 {range\_of\_blocking\_cars\_and\_side[1]}, ") # two right fields of the blocking\_car are free

else:

# if one left field of the blocking\_car is free or two left fields is the parkslot border

if [positions for car, positions in self.BLOCKING\_CARS.items() if car == blocking\_car][0][0] - 1 in self.BLOCKING\_CARS\_POSITIONS or [positions for car, positions in self.BLOCKING\_CARS.items() if car == blocking\_car][0][0] - 2 < self.parkslot\_border[1]:

car\_result.append(f"{blocking\_car} 1 {range\_of\_blocking\_cars\_and\_side[1]}, ")

continue

car\_result.append(f"{blocking\_car} 2 {range\_of\_blocking\_cars\_and\_side[1]}, ") # two left fields of the blocking\_car are free

car\_result.append(f"{directly\_blocking\_car[0]} {range\_of\_blocking\_cars\_and\_side[2]} {range\_of\_blocking\_cars\_and\_side[1]}")

return "".join(car\_result)

test\_code.py:

import time

from get\_data import GetData

from solve\_sliding\_parking\_lot import SolveSlidingParkingLot

class TestSolveSlidingParkingLot():

def \_\_init\_\_(self) -> None:

self.start\_time = time.time()

self.test\_counter = 30

self.passed\_tests = 0

self.failed\_tests = 0

self.test\_result = 0

self.start\_tests()

result = self.get\_test\_result()

print(f"\033[97mResult: {result:.0%}\nTest passed: {self.passed\_tests}\nTest failed {self.failed\_tests}\033[97m")

print(f"\n--- {time.time() - self.start\_time} seconds executing time ---")

def start\_tests(self) -> None:

# GetData

self.get\_data\_test\_input\_file\_0()

self.get\_data\_test\_input\_file\_1()

self.get\_data\_test\_input\_file\_2()

self.get\_data\_test\_input\_file\_3()

self.get\_data\_test\_input\_file\_4()

self.get\_data\_test\_input\_file\_5()

# SolveSlidingParking\_lot

self.solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_no\_blocking\_car()

self.solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_01()

self.solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_02()

self.solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_03()

self.solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_04()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_right\_move\_01()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_right\_move\_02()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_right\_move\_03()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_left\_move\_01()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_left\_move\_02()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_left\_move\_03()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_one\_multiple\_move\_01()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_one\_multiple\_move\_02()

self.solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_one\_multiple\_move\_03()

self.solve\_sliding\_parking\_lot\_test\_one\_move\_get\_blocking\_car\_blocking\_car()

self.solve\_sliding\_parking\_lot\_test\_get\_moves\_needed\_per\_side\_1\_left\_2\_right()

self.solve\_sliding\_parking\_lot\_test\_get\_moves\_needed\_per\_side\_2\_left\_1\_right()

self.solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_left\_01()

self.solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_left\_02()

self.solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_right\_01()

self.solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_right\_02()

self.solve\_sliding\_parking\_lot\_test\_get\_multiple\_cars\_need\_to\_move\_one\_car\_two\_moves()

self.solve\_sliding\_parking\_lot\_test\_get\_multiple\_cars\_need\_to\_move\_two\_cars\_both\_two\_moves()

self.solve\_sliding\_parking\_lot\_test\_get\_multiple\_cars\_need\_move\_one\_car\_two\_moves()

def check\_if\_equal(self, test\_result, result, test\_num) -> None:

if test\_result == result:

self.passed\_tests += 1

if test\_num < 10:

print("\033[92m=================================\033[92m")

print(f"\033[92m========= Test {test\_num} passed =========\033[92m")

print("\033[92m=================================\033[92m\n")

else:

print("\033[92m=================================\033[92m")

print(f"\033[92m======== Test {test\_num} passed =========\033[92m")

print("\033[92m=================================\033[92m\n")

else:

self.failed\_tests += 1

if test\_num < 10:

print("\033[91m=================================\033[91m")

print(f"\033[91m========= Test {test\_num} failed =========\033[91m")

print("\033[91m=================================\033[91m\n")

else:

print("\033[91m=================================\033[91m")

print(f"\033[91m======== Test {test\_num} failed =========\033[91m")

print("\033[91m=================================\033[91m\n")

print(f"\033[91mYour result: {test\_result}\033[91m")

print(f"\033[92mCorrect result: {result}\033[92m\n")

# GetData tests

def get\_data\_test\_input\_file\_0(self) -> None:

test\_result = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz0.txt")

self.check\_if\_equal(test\_result=test\_result, result=(['A', 'B', 'C', 'D', 'E', 'F', 'G'], {'H': [2, 3], 'I': [5, 6]}), test\_num=1)

def get\_data\_test\_input\_file\_1(self) -> None:

test\_result = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz1.txt")

self.check\_if\_equal(test\_result=test\_result, result=(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N'], {'O': [1, 2], 'P': [3, 4], 'Q': [6, 7], 'R': [10, 11]}), test\_num=2)

def get\_data\_test\_input\_file\_2(self) -> None:

test\_result = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz2.txt")

self.check\_if\_equal(test\_result=test\_result, result=(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N'], {'O': [2, 3], 'P': [5, 6], 'Q': [7, 8], 'R': [9, 10], 'S': [12, 13]}), test\_num=3)

def get\_data\_test\_input\_file\_3(self) -> None:

test\_result = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz3.txt")

self.check\_if\_equal(test\_result=test\_result, result=(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N'], {'O': [1, 2], 'P': [4, 5], 'Q': [8, 9], 'R': [10, 11], 'S': [12, 13]}), test\_num=4)

def get\_data\_test\_input\_file\_4(self) -> None:

test\_result = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz4.txt")

self.check\_if\_equal(test\_result=test\_result, result=(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P'], {'Q': [0, 1], 'R': [2, 3], 'S': [6, 7], 'T': [10, 11], 'U': [13, 14]}), test\_num=5)

def get\_data\_test\_input\_file\_5(self) -> None:

test\_result = GetData.get\_data\_from\_file\_return\_necessary\_data(file\_path="/home/marius/Documents/Bundeswettbewerb Informatik 2021/Round 1/task 01/files/parkplatz5.txt")

self.check\_if\_equal(test\_result=test\_result, result=(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O'], {'P': [2, 3], 'Q': [4, 5], 'R': [8, 9], 'S': [12, 13]}), test\_num=6)

# SolveSlidingParkingLot tests

def solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_no\_blocking\_car(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D"], BLOCKING\_CARS={})

test\_obj.assign\_moves\_to\_cars()

self.check\_if\_equal(test\_result=test\_obj.result, result=['A:', 'B:', 'C:', 'D:'], test\_num=7)

def solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_01(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D"], BLOCKING\_CARS={"E": [3, 4]})

test\_obj.assign\_moves\_to\_cars()

self.check\_if\_equal(test\_result=test\_obj.result, result=['A:', 'B:', 'C:', 'D: E 2 left'], test\_num=8)

def solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_02(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P"], BLOCKING\_CARS={"Q": [15, 16]})

test\_obj.assign\_moves\_to\_cars()

self.check\_if\_equal(test\_result=test\_obj.result, result=['A:', 'B:', 'C:', 'D:', 'E:', 'F:', 'G:', 'H:', 'I:', 'J:', 'K:', 'L:', 'M:', 'N:', 'O:', 'P: Q 2 left'], test\_num=9)

def solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_03(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "P", "X", "Y", "Z"], BLOCKING\_CARS={"Ü": [25, 26]})

test\_obj.assign\_moves\_to\_cars()

self.check\_if\_equal(test\_result=test\_obj.result, result=['A:', 'B:', 'C:', 'D:', 'E:', 'F:', 'G:', 'H:', 'I:', 'J:', 'K:', 'L:', 'M:', 'N:', 'O:', 'P:', 'Q:', 'R:', 'S:', 'T:', 'U:', 'V:', 'P:', 'X:', 'Y:', 'Z: Ü 2 left'], test\_num=10)

def solve\_sliding\_parking\_lot\_test\_assign\_moves\_to\_cars\_one\_car\_gets\_blocked\_04(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "P", "X", "Y", "Z", "AA", "AB", "AC", "AD", "AE", "AF", "AG", "AH", "AI", "AJ", "AK", "AL", "AM", "AN", "AO", "AP", "AQ", "AR", "AS", "AT", "AU", "AV", "AP", "AX", "AY", "AZ"], BLOCKING\_CARS={"BA": [51, 52]})

test\_obj.assign\_moves\_to\_cars()

self.check\_if\_equal(test\_result=test\_obj.result, result=['A:', 'B:', 'C:', 'D:', 'E:', 'F:', 'G:', 'H:', 'I:', 'J:', 'K:', 'L:', 'M:', 'N:', 'O:', 'P:', 'Q:', 'R:', 'S:', 'T:', 'U:', 'V:', 'P:', 'X:', 'Y:', 'Z:', 'AA:', 'AB:', 'AC:', 'AD:', 'AE:', 'AF:', 'AG:', 'AH:', 'AI:', 'AJ:', 'AK:', 'AL:', 'AM:', 'AN:', 'AO:', 'AP:', 'AQ:', 'AR:', 'AS:', 'AT:', 'AU:', 'AV:', 'AP:', 'AX:', 'AY:', 'AZ: BA 2 left'], test\_num=11)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_right\_move\_01(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [0, 1]})

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="right", test\_num=12)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_right\_move\_02(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [1, 2]})

test\_obj.car\_position = 1

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="right", test\_num=13)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_right\_move\_03(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [2, 3]})

test\_obj.car\_position = 2

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="right", test\_num=14)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_left\_move\_01(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [3, 4]})

test\_obj.car\_position = 4

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="left", test\_num=15)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_left\_move\_02(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [2, 3]})

test\_obj.car\_position = 3

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="left", test\_num=16)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_single\_left\_move\_03(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [1, 2]})

test\_obj.car\_position = 2

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="left", test\_num=17)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_one\_multiple\_move\_01(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [0, 1], "G": [2, 3]})

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="multiple", test\_num=18)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_one\_multiple\_move\_02(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [1, 2], "G": [3, 4]})

test\_obj.car\_position = 3

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="multiple", test\_num=19)

def solve\_sliding\_parking\_lot\_test\_check\_if\_car\_needs\_one\_or\_multiple\_moves\_one\_multiple\_move\_03(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [1, 2], "G": [3, 4]})

test\_obj.car\_position = 4

test\_result = test\_obj.check\_if\_car\_needs\_one\_or\_multiple\_moves()

self.check\_if\_equal(test\_result=test\_result, result="multiple", test\_num=20)

def solve\_sliding\_parking\_lot\_test\_one\_move\_get\_blocking\_car\_blocking\_car(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E"], BLOCKING\_CARS={"F": [0, 1]})

test\_result = test\_obj.one\_move\_get\_blocking\_car()

self.check\_if\_equal(test\_result=test\_result, result="F", test\_num=21)

def solve\_sliding\_parking\_lot\_test\_get\_moves\_needed\_per\_side\_1\_left\_2\_right(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H"], BLOCKING\_CARS={"I": [1, 2], "J": [3, 4], "K": [5, 6]})

test\_obj.car\_position = 2

test\_result = test\_obj.get\_moves\_needed\_per\_side()

self.check\_if\_equal(test\_result=test\_result, result=[[1, 2], [1, 2]], test\_num=22)

def solve\_sliding\_parking\_lot\_test\_get\_moves\_needed\_per\_side\_2\_left\_1\_right(self) -> None:

test\_obj = SolveSlidingParkingLot(["A", "B", "C", "D", "E", "F", "G", "H"], BLOCKING\_CARS={"I": [1, 2], "J": [3, 4], "K": [5, 6]})

test\_obj.car\_position = 1

test\_result = test\_obj.get\_moves\_needed\_per\_side()

self.check\_if\_equal(test\_result=test\_result, result=[[2, 1], [2, 1]], test\_num=23)

def solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_left\_01(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H"], BLOCKING\_CARS={"I": [1, 2], "J": [3, 4], "K": [5, 6]})

test\_obj.car\_position = 2

test\_result = test\_obj.find\_range\_of\_blocking\_cars\_and\_side(moves\_needed\_per\_side=[1, 2], backup\_moves\_needed\_per\_side=[1,2])

self.check\_if\_equal(test\_result=test\_result, result=[-1, 'left', 1], test\_num=24)

def solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_left\_02(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H"], BLOCKING\_CARS={"I": [3, 4], "J": [5, 6]})

test\_obj.car\_position = 5

test\_result = test\_obj.find\_range\_of\_blocking\_cars\_and\_side(moves\_needed\_per\_side=[1, 2], backup\_moves\_needed\_per\_side=[1, 2])

self.check\_if\_equal(test\_result=test\_result, result=[1, 'left', 1], test\_num=25)

def solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_right\_01(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H"], BLOCKING\_CARS={"I": [3, 4], "J": [5, 6]})

test\_obj.car\_position = 5

test\_result = test\_obj.find\_range\_of\_blocking\_cars\_and\_side(moves\_needed\_per\_side=[2, 1], backup\_moves\_needed\_per\_side=[2,1])

self.check\_if\_equal(test\_result=test\_result, result=[8, 'right', 1], test\_num=26)

def solve\_sliding\_parking\_lot\_test\_find\_range\_of\_blocking\_cars\_and\_side\_right\_02(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F", "G", "H"], BLOCKING\_CARS={"I": [0, 1], "J": [2, 3], "K": [4, 5]})

test\_obj.car\_position = 4

test\_result = test\_obj.find\_range\_of\_blocking\_cars\_and\_side(moves\_needed\_per\_side=[1, 2], backup\_moves\_needed\_per\_side=[1, 2])

self.check\_if\_equal(test\_result=test\_result, result=[8, 'right', 2], test\_num=27)

def solve\_sliding\_parking\_lot\_test\_get\_multiple\_cars\_need\_to\_move\_one\_car\_two\_moves(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F"], BLOCKING\_CARS={"G": [4, 5]})

test\_obj.car\_position = 5

test\_result = test\_obj.get\_multiple\_cars\_need\_to\_move([3, "left", 1])

self.check\_if\_equal(test\_result=test\_result, result="G 1 left", test\_num=28)

def solve\_sliding\_parking\_lot\_test\_get\_multiple\_cars\_need\_to\_move\_two\_cars\_both\_two\_moves(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F"], BLOCKING\_CARS={"G": [4, 5], "H": [2, 3]})

test\_obj.car\_position = 4

test\_result = test\_obj.get\_multiple\_cars\_need\_to\_move([1, "left", 2])

self.check\_if\_equal(test\_result=test\_result, result= "H 2 left, G 2 left", test\_num=29)

def solve\_sliding\_parking\_lot\_test\_get\_multiple\_cars\_need\_move\_one\_car\_two\_moves(self) -> None:

test\_obj = SolveSlidingParkingLot(CARS\_IN\_PARKSLOTS=["A", "B", "C", "D", "E", "F"], BLOCKING\_CARS={"G": [0, 1], "H": [2, 3]})

test\_obj.car\_position = 2

test\_result = test\_obj.get\_multiple\_cars\_need\_to\_move([2, "right", 2])

self.check\_if\_equal(test\_result=test\_result, result="H 2 right", test\_num=30)

def get\_test\_result(self) -> float:

return self.passed\_tests / self.test\_counter

if \_\_name\_\_ == "\_\_main\_\_":

TestSolveSlidingParkingLot()