This is just a simple shared plaintext pad, with no execution capabilities.

When you know what language you'd like to use for your interview,

simply choose it from the dropdown in the top bar.

You can also change the default language your pads are created with

in your account settings: https://coderpad.io/settings

Enjoy your interview!

# Input: "20+12\*3+40"

# Output: 96

# only multiplication and addition

# only positive integers

# if there were only subtraction and addition then order wouldn't matter, but because we have multiplication it does

# without order mattering you could push items onto a stack and then pop and evaluate when you have an operator

# "1+12+40\*3\*2"

# 1+ calc(12+40\*3\*2)

# 1 + 12 + calc(40\*3\*2)

# "2\*3\*40+12+1"

# PSEUDO CODE - SANDBOX

acc = 0

for i in range(string\_input):

# Let's say I get current operator and next operator

if i =="+" and i+1 != "\*" :

# Then I can current value and products to the right

acc += prev

# Solution

class Solution(object):

def calculate\_add(self, input: str) -> int:

running\_total = 0

# Only addition - should give me a list of numbers

numbers = input.split('+')

for num in numbers:

running\_total += int(num)

return runnint\_total

def calculate\_addtion\_no\_split(self, input: str) -> int:

running\_total = 0

num = ""

for c in input:

if c.isdigit():

num += c

else:

num = int(num)

running\_total += num

num = ""

if num != "":

running\_total += int(num)

return runnint\_total

def calculate\_with\_multiplication\_no\_split(self, input: str) -> int:

running\_total = 0

product\_so\_far = 1

num = ""

for c in input:

if c.isdigit():

num += c

else:

if c == '+':

num = int(num)

running\_total += num

num = ""

else:

num = int(num)

product\_so\_far \*= num

num = product\_so\_far

if num != "":

running\_total += int(num)

return runnint\_total

input = "12+40\*3"

print(Solution().calculate\_add(input))

# barchart holding water

# [1, 3, 2 ,4] -> 1

# \*

# \* % \* % % % \*

# \* \* \* % \* % \*

# \* \* \* \* \* \* \* \* \*

# Scan through chart

# get first set of buckets by finding local minimums

# cases where chart[i-1] > chart[i] and chart[i] < chart[i + 1]

# fill buckets while there no more walls to left and right

# maximum of height to the left and right of you gives the height of water

# Do max values computations

from typing import List

class Solution(object):

def water(self, chart: List[int]) -> int:

water = 0

# do max heights for each bar

# Match index to max height

heights = {}

for i, num in enumerate(chart):

if i == 0:

heights[i] = 0

elif i == len(chart)-1:

heights[i] = 0

else:

heights[i] = max((min(chart[:i], chart[i+1:])-chart[i]),0)

# use heights to compute water above

for k,v in heights.items():

water += v

# return

return water

chart = [1,3,2,4]

print(Solution().water(chart))

# This is just a simple shared plaintext pad, with no execution capabilities.

# When you know what language you'd like to use for your interview,

# simply choose it from the dropdown in the top bar.

# You can also change the default language your pads are created with

# in your account settings: https://coderpad.io/settings

# Enjoy your interview!

# ######## PROBLEM #############

# div(a,b) -> Str

# a < b

# a, b > 0

# div(1,2) = "0.5"

# div(1,3) = "0.(3)"

# ######### NOTES ######################################################

# dont use floating point division

# These are okay

# // %

# always valid input integers

# ######### PSUEDOCODE ###################################################

# two steps:

# Make the right side of the digit

# 0.33

# ---------

# 3 | 1.0

# 9

# 1.0

# 0.3 (1 8)

# --------

# 22 | 7.0

# 6 6

# 4 0

# 2 2

# 1 8 0

# 1 7 6

# 4

# ######### IMPLEMENTATION ###############################################

from typing import List

def div(a: int, b: int) -> str:

result = "0."

divisor = b

dividend = a\*10

results = []

while(True):

temp = dividend // divisor

if temp > 0:

# subtract the product of temp and divisor

dividend -= temp \* divisor

if dividend == 0:

# convert results list to string output

results.append(temp)

result += "".join(str(results))

# done

break

# check if I've seen this before

if dividend not in set:

results.append((dividend,len(results)))

# if we've seen this before we are done

else:

# build our "inside parenthesis" content

# get index where repeating begins

index\_repeat = [r[1] for r in results if r[0] == dividend]

# get non-repeating characters first

results += "".join([r[0] for r in results if r[1] < index\_repeat])

# get repeating characters last

results += "".join([r[0] for r in results])

# done

break

# not big enough, multiple by 10

else:

dividend \*= 10

return result

print(div(1,2))

####### TESTS #########################

# This is just a simple shared plaintext pad, with no execution capabilities.

# When you know what language you'd like to use for your interview,

# simply choose it from the dropdown in the top bar.

# You can also change the default language your pads are created with

# in your account settings: https://coderpad.io/settings

# Enjoy your interview!

# Hello Steven, this Maxim from FB. Can you see me typing?

# Yes, should I join here or stay in bluejeans call?

# On which blue jeans are you?

# https://fb.workplace.com/meet/cmnaSfDwFX/

# I am in coderpad call now

Simple CNN convolution layer

1 input layer and 1 output layer

# thinking about parameters

input matrix

kernel\_size: dimensions of the kernel

padding = same (adding zeros to matrix edges)

stride = how many positions to move between each convolution

def get\_image\_window(i,j, kernel\_size):

# TODO

window = []

for y in range(0,kernel\_size):

for x in range(0, kernel\_size):

window.append(input[i-kernel\_size//2+x][j-kernel\_size//2+y])

return window

def compute(i: int,j: int):

image\_window = get\_image\_window(i,j)

result = 0

for img\_elem, kernel\_element in zip(image\_window, kernel\_element):

result += img\_elem\*kernel\_element

return result

def conv(input\_image: List[List[int]], kernel\_size: int, padding: int, stride: int) -> List[List[int]]:

# check if this valid

# assume image is bigger than kernel

# compute the size of output here based on input size and parameters

# TODO

out\_rows = input\_size - kernel\_size

out\_cols =

# case without padding or stride > 1

for i in range(0,len(input\_image[0])):

for j in range(0, len(input\_image)):

out = compute\_conv(i,j)

out[f(i)][f(j)] = out

# return output

return out