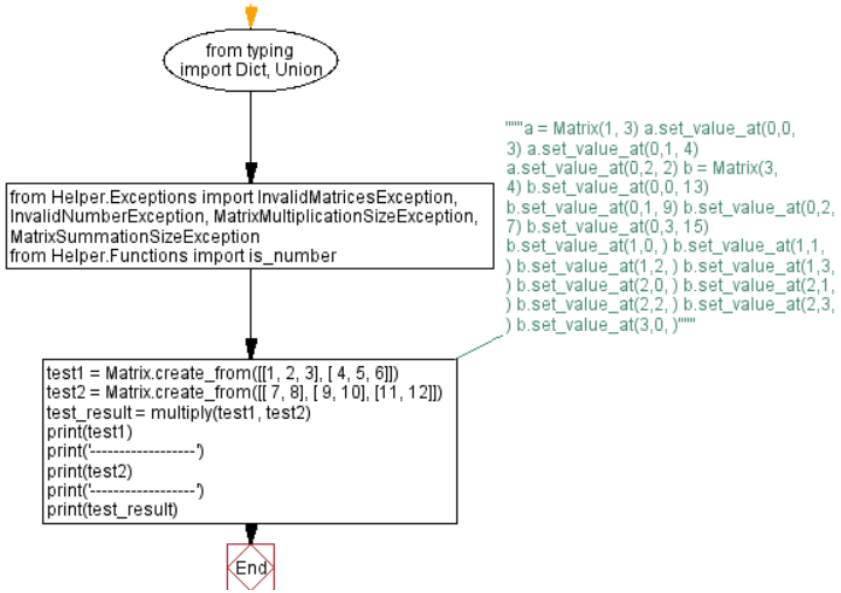
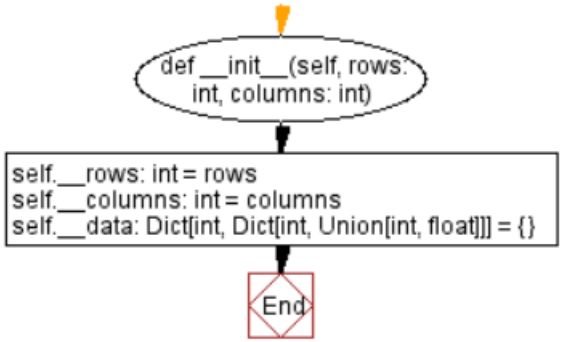
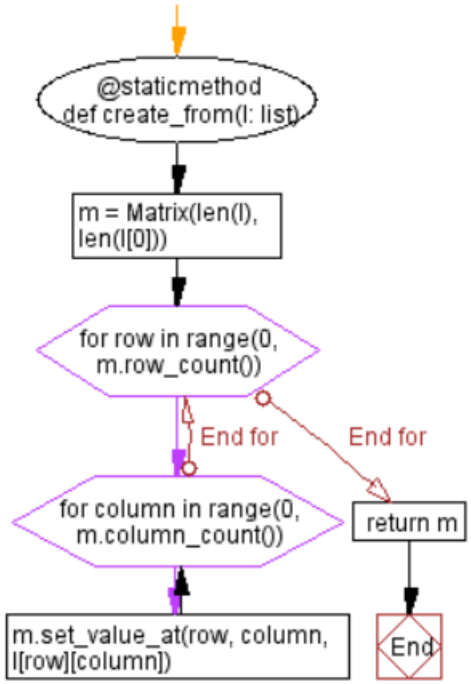
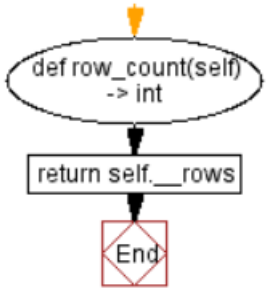
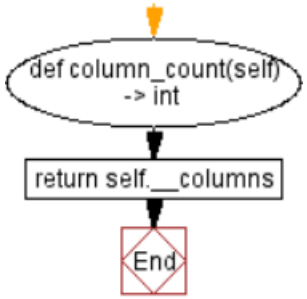

from typing import Dict, Union
from Helper.Exceptions import InvalidMatricesException, InvalidNumberException, MatrixMultiplicationSizeException, MatrixSummationSizeException
from Helper.Functions import is_number
"""a = Matrix(1, 3) a.set_value_at(0,0, 3) a.set_value_at(0,1, 4) a.set_value_at(0,2, 2) b = Matrix(3, 4) b.set_value_at(0,0, 13) b.set_value_at(0,1, 9) b.set_value_at(0,2, 7) b.set_value_at(0,3, 15) b.set_value_at(1,0,) b.set_value_at(1,1,) b.set_value_at(1,2,) b.set_value_at(1,3,) b.set_value_at(2,0,) b.set_value_at(2,1,) b.set_value_at(2,2,) b.set_value_at(2,3,) b.set_value_at(3,0,)"""
test1 = Matrix.create_from([[1, 2, 3], [4, 5, 6]])
test2 = Matrix.create_from([[7, 8], [9, 10], [11, 12]])
test_result = multiply(test1, test2)
print(test1)
print('------------------')
print(test2)
print('------------------')
print(test_result)
End

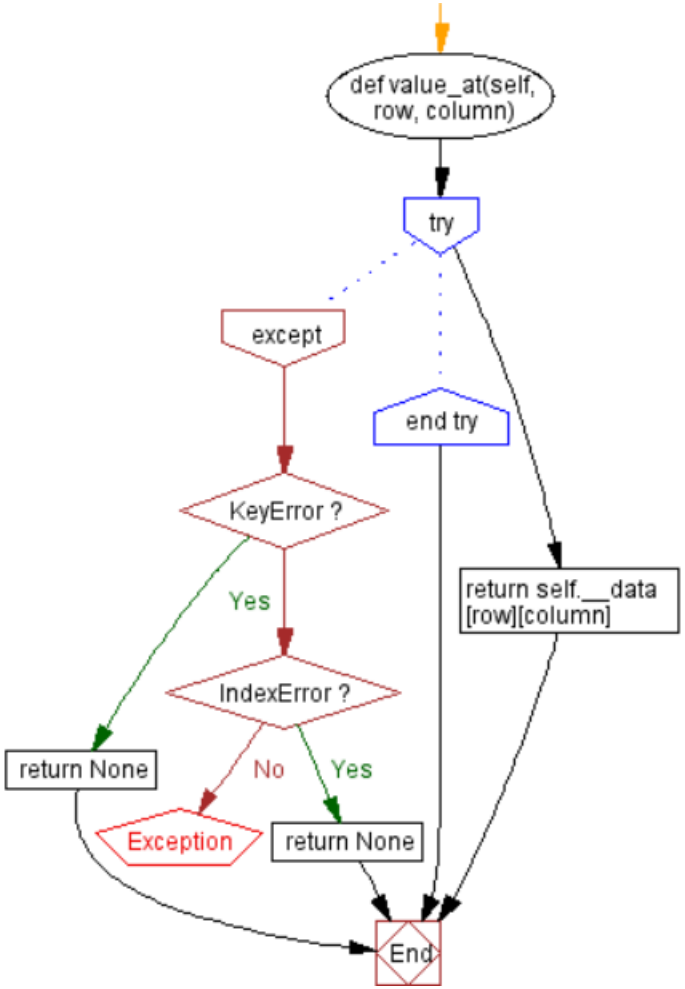
class Matrix(object)
End

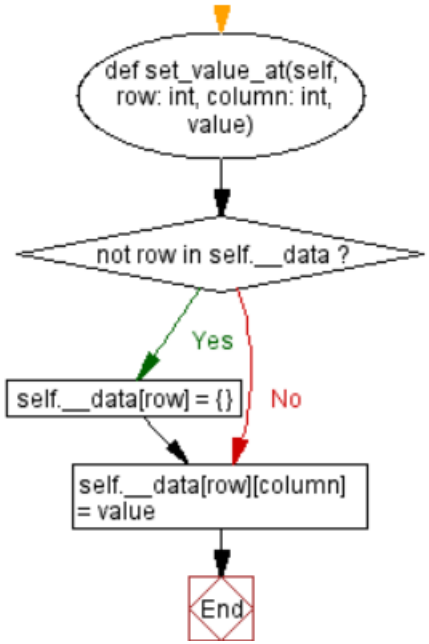

def __init__(self, rows: int, columns: int)
self.__rows: int = rows
self.__columns: int = columns
self.__data: Dict[int, Dict[int, Union[int, float]]] = {}
End


@staticmethod
def create_from(l: list)
m = Matrix(len(l), len(l[0]))
for row in range(0, m.row_count())
End for
End for
for column in range(0, m.column_count())
return m
m.set_value_at(row, column, l[row][column])
End


def row_count(self) -> int
return self.__rows
End

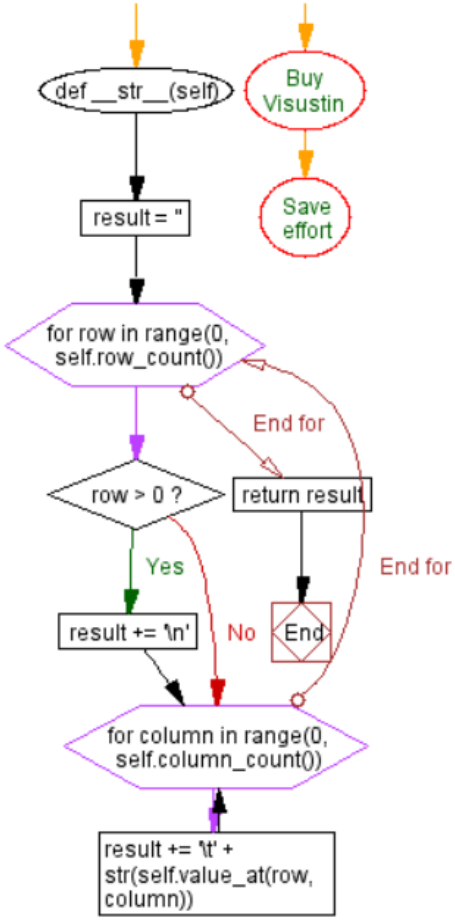

def column_count(self) -> int
return self.__columns
End

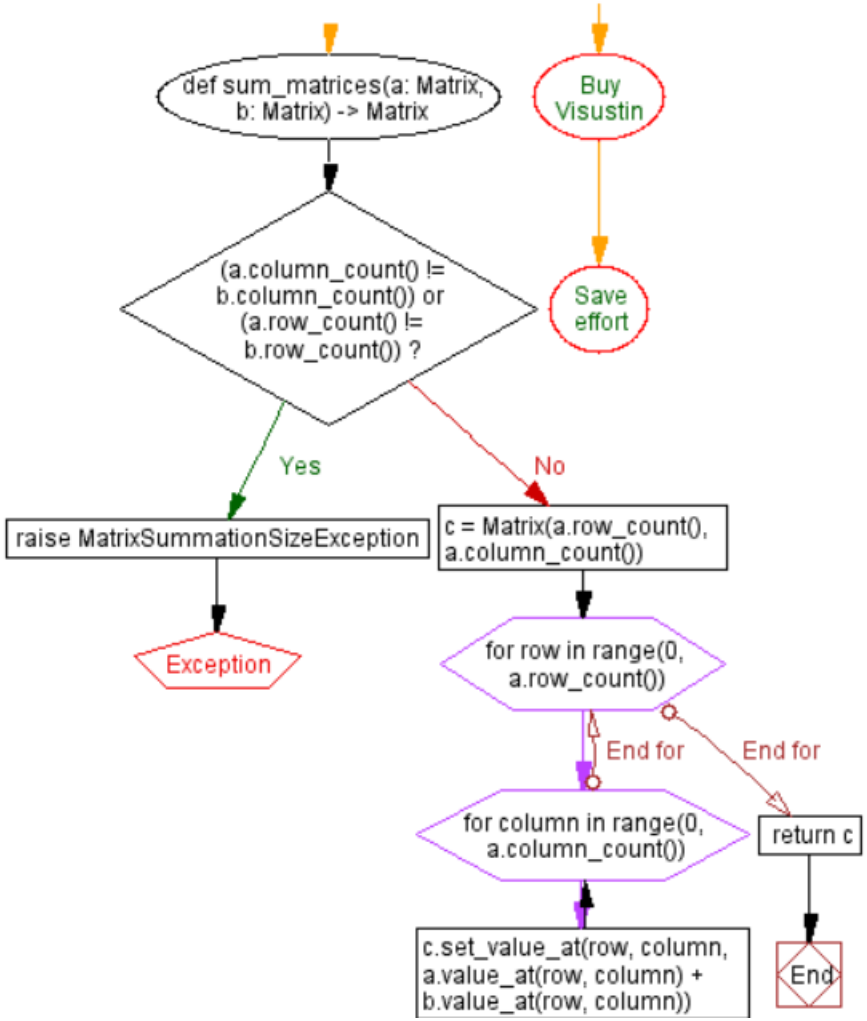

def value_at(self, row, column)
try
except
end try
KeyError ?
Yes
IndexError ?
No
Yes
return self.__data [row][column]
return None
Exception
return None
End

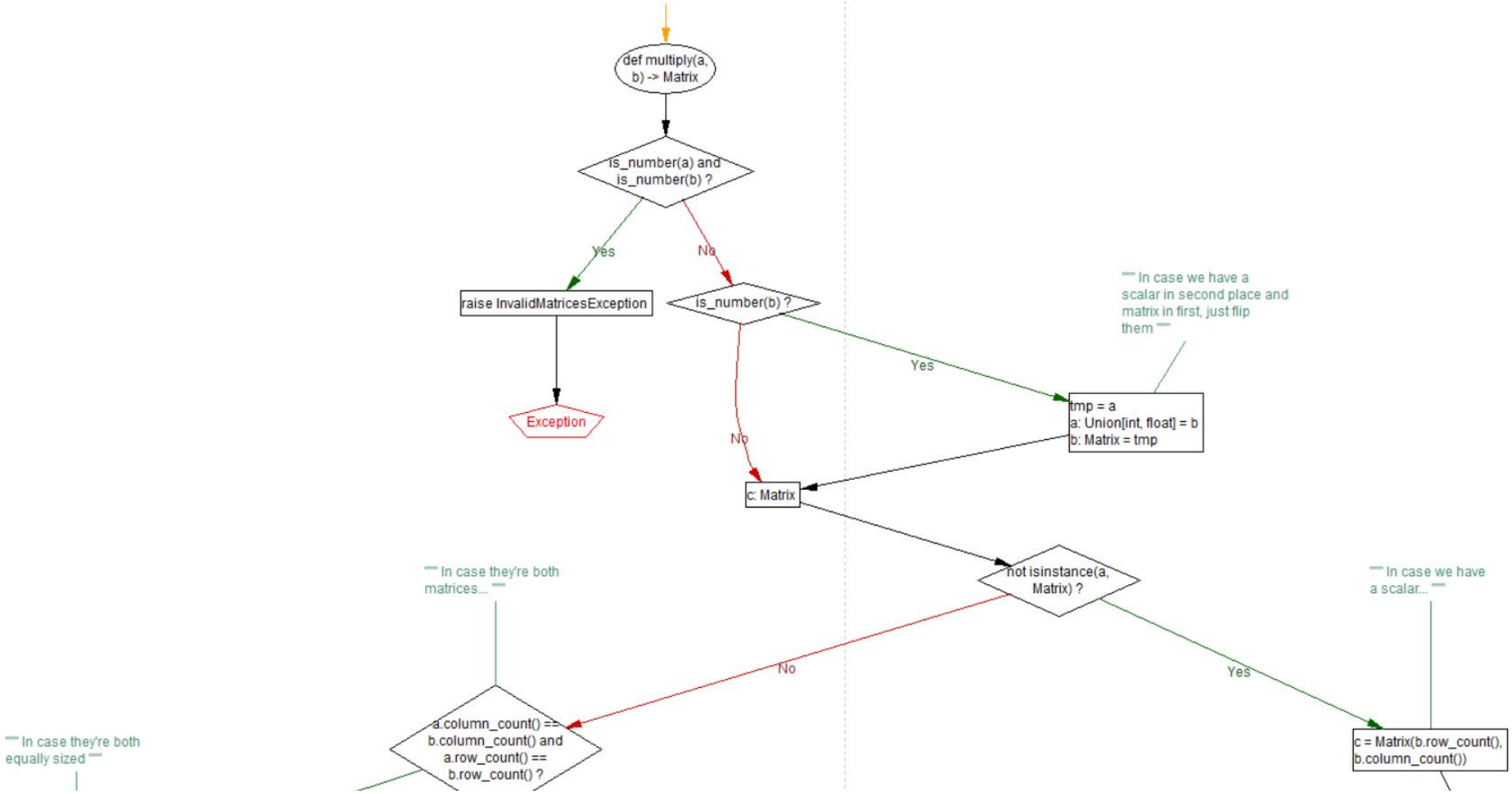

def set_value_at(self, row: int, column: int, value)
not row in self.__data ?
Yes
No
self.__data[row] = {}
self.__data[row][column] = value
End

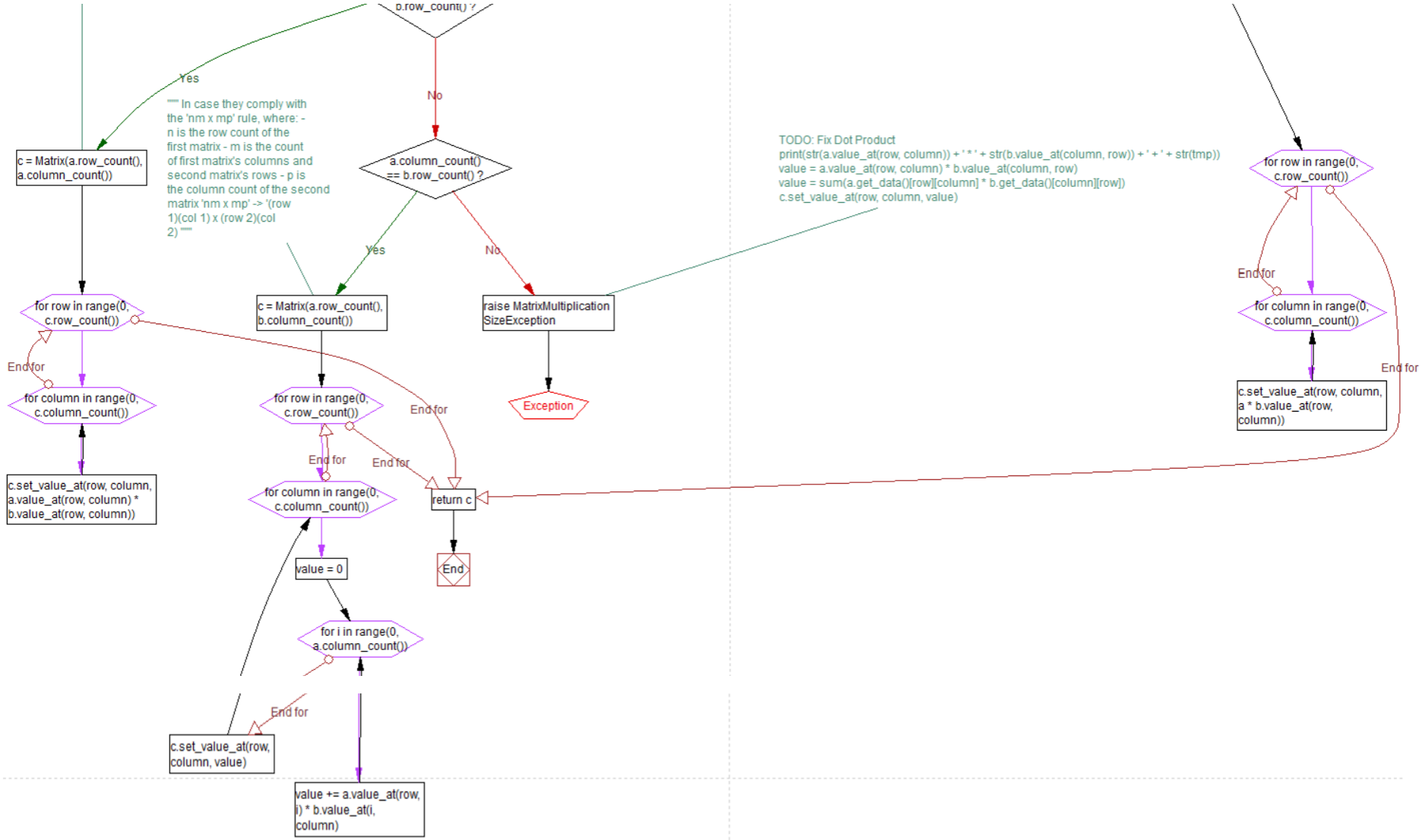
Buy Visustin
Save effort

def get_data(self)
return self.__data
End


def __str__(self)
Buy Visustin
Save effort
result = ''
for row in range(0, self.row_count())
End for
row > 0 ?
return result
Yes
End for
result += '\n'
No
End
for column in range(0, self.column_count())
result += '\t' + str(self.value_at(row, column))


def sum_matrices(a: Matrix, b: Matrix) -> Matrix
Buy Visustin
Save effort
(a.column_count() != b.column_count()) or (a.row_count() != b.row_count()) ?
Yes
No
raise MatrixSummationSizeException
Exception
c = Matrix(a.row_count(), a.column_count())
for row in range(0, a.row_count())
End for
End for
for column in range(0, a.column_count())
return c
c.set_value_at(row, column, a.value_at(row, column) + b.value_at(row, column))
End


def multiply(a, b) -> Matrix
is_number(a) and is_number(b) ?
Yes
No
raise InvalidMatricesException
Exception
is_number(b) ?
Yes
No
"" In case we have a scalar in second place and matrix in first, just flip them ""
tmp = a
a: Union[int, float] = b
b: Matrix = tmp
c: Matrix
not isinstance(a, Matrix) ?
No
Yes
"" In case they're both matrices... ""
a.column_count() == b.column_count() and a.row_count() == b.row_count() ?
"" In case they're both equally sized ""
"" In case we have a scalar... ""
c = Matrix(b.row_count(), b.column_count())


b.row_count() ?
Yes
No
""" In case they comply with the 'nm x mp' rule, where: - n is the row count of the first matrix - m is the count of first matrix's columns and second matrix's rows - p is the column count of the second matrix 'nm x mp' -> '(row 1)(col 1) x (row 2)(col 2) """
c = Matrix(a.row_count(), a.column_count())
a.column_count() == b.row_count() ?
Yes
No
TODO: Fix Dot Product
print(str(a.value_at(row, column)) + ' * ' + str(b.value_at(column, row)) + ' + ' + str(tmp))
value = a.value_at(row, column) * b.value_at(column, row)
value = sum(a.get_data()[row][column] * b.get_data()[column][row])
c.set_value_at(row, column, value)
for row in range(0, c.row_count())
End for
for column in range(0, c.column_count())
c.set_value_at(row, column, a * b.value_at(row, column))
End for
for row in range(0, c.row_count())
End for
for column in range(0, c.column_count())
c.set_value_at(row, column, a.value_at(row, column) * b.value_at(row, column))
c = Matrix(a.row_count(), b.column_count())
raise MatrixMultiplicationSizeException
Exception
for row in range(0, c.row_count())
End for
End for
for column in range(0, c.column_count())
value = 0
for i in range(0, a.column_count())
End for
c.set_value_at(row, column, value)
value += a.value_at(row, i) * b.value_at(i, column)
return c
End