Technical University of Moldova

Special Mathematics

Laboratory No.4

Author:
st. Polina Gore
gr. FAF-161

Supervisor:
Victor Țurcanu

April 22, 2017

Understanding logical operations

Here you have *image* 1 that can be obtained by running the code from *listing* 1.

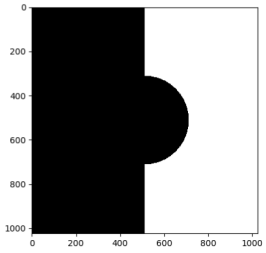


Figure 1: Desired output

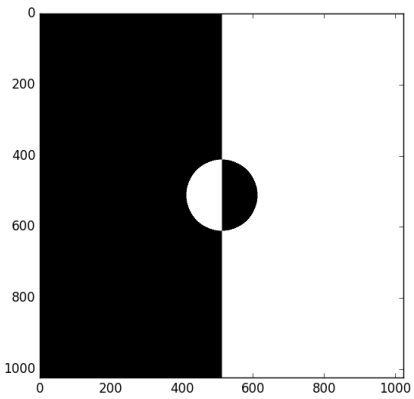
```
import numpy as np
import matplotlib.pyplot as plt

nx = 1024
x = np.linspace(-nx/2, nx/2, nx)
mat_x, mat_y = np.meshgrid(x, x)
R = np.sqrt(mat_x**2 + mat_y**2)
filter = np.logical_and(mat_x > 0, R
    > 100)
plt.imshow(filter, cmap='gray')
plt.show()
```

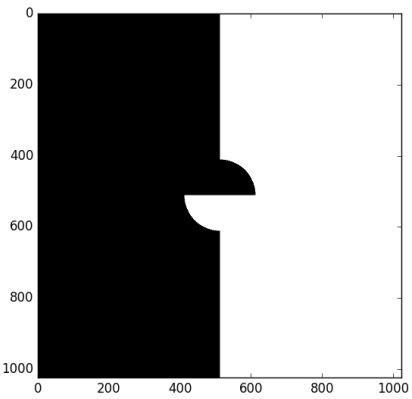
Listing 1: Code

Change it to obtain the following images.

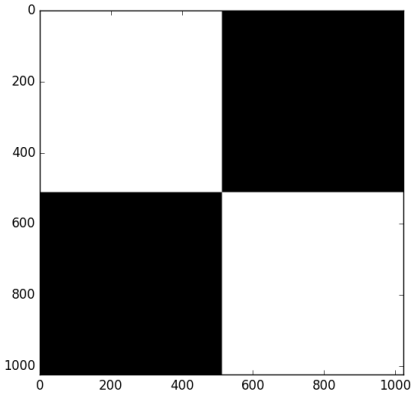
Solution

1) Either $x < 0$ or $R > 100$ *2)* ($R > 100$ or $y > 0$) and ($R < 100$ or $x > 0$)

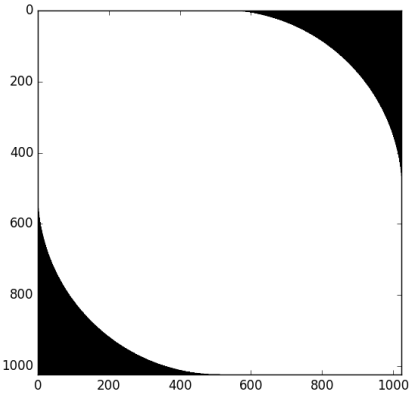




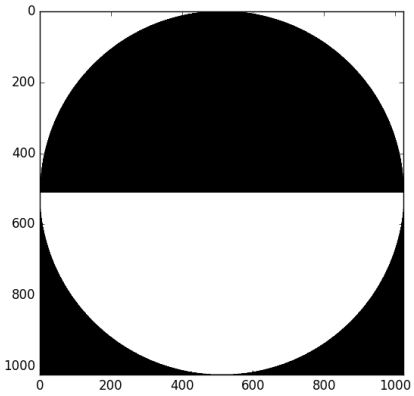
3) Neither $y < 0$ nor $x < 0$



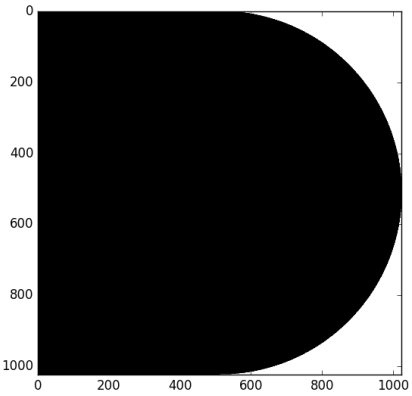
4) $R < 512$ or (either $y < 0$ or $x > 0$)



5) Either $y > 0$ or $R > 512$



6) $x > 0$ and $R > 512$



Leibniz harmonic triangle

Write a program that prints the harmonic triangle for the depth n, where n is an input value.

Solution

The Leibniz harmonic triangle *is a triangular arrangement of unit fractions in which the outermost diagonals consist of the reciprocals of the row numbers and each inner cell is the absolute value of the cell above minus the cell to the left.* (Wikipedia)

This is the rule I used to *'create'* the Leibniz triangle.

Examples:

```
Enter the nr of rows: 7

The Leibniz's harmonic triangle:


                    1
                 1/2   1/2
              1/3   1/6   1/3
           1/4   1/12   1/12   1/4
        1/5   1/20   1/30   1/20   1/5
     1/6   1/30   1/60   1/60   1/30   1/6
1/7   1/42   1/105   1/140   1/105   1/42   1/7



Enter the nr of rows: 12

The Leibniz's harmonic triangle:


                                        1
                                     1/2   1/2
                                  1/3   1/6   1/3
                               1/4   1/12   1/12   1/4
                            1/5   1/20   1/30   1/20   1/5
                         1/6   1/30   1/60   1/60   1/30   1/6
                      1/7   1/42   1/105   1/140   1/105   1/42   1/7
                   1/8   1/56   1/168   1/280   1/280   1/168   1/56   1/8
                1/9   1/72   1/252   1/504   1/630   1/504   1/252   1/72   1/9
          1/10   1/90   1/360   1/840   1/1260   1/1260   1/840   1/360   1/90   1/10
     1/11   1/110   1/495   1/1320   1/2310   1/2772   1/2310   1/1320   1/495   1/110   1/11
1/12   1/132   1/660   1/1980   1/3960   1/5544   1/5544   1/3960   1/1980   1/660   1/132   1/12
```

Truth table solver

You have to write a program that computes the truth table for various expressions.
The set of expressions are limited to:

- *and* operation
- *or* operation
- *not* operation
- supports paranthesis

Solution

First of all, I checked the expression to be sure it's a working one using regex (which needs the *re* module).
If it's not, exit with an error message, otherwise, evaluate it.
To evaluate the expression, I used the *eval* function, but before, I changed every sign (!, *, +) to the corresponding logical terms (not, and, or).
And to make the truth table, I used the *product* function from *itertools* module which is the cross product of true and false repeated n times (n = number of distinct variables).

Examples:

```
Enter a valid logic expression: (a + b * c) + !d
a | b | c | d | (a + b * c) + !d
--------------------------------
0 | 0 | 0 | 0 | 1
0 | 0 | 0 | 1 | 0
0 | 0 | 1 | 0 | 1
0 | 0 | 1 | 1 | 0
0 | 1 | 0 | 0 | 1
0 | 1 | 0 | 1 | 0
0 | 1 | 1 | 0 | 1
0 | 1 | 1 | 1 | 1
1 | 0 | 0 | 0 | 1
1 | 0 | 0 | 1 | 1
1 | 0 | 1 | 0 | 1
1 | 0 | 1 | 1 | 1
1 | 1 | 0 | 0 | 1
1 | 1 | 0 | 1 | 1
1 | 1 | 1 | 0 | 1
1 | 1 | 1 | 1 | 1
```

```
Enter a valid logic expression: (!x + y) * z + (!z*y*k)
k | x | y | z | (!x + y) * z + (!z*y*k)
---------------------------------------
0 | 0 | 0 | 0 | 0
0 | 0 | 0 | 1 | 1
0 | 0 | 1 | 0 | 0
0 | 0 | 1 | 1 | 1
0 | 1 | 0 | 0 | 0
0 | 1 | 0 | 1 | 0
0 | 1 | 1 | 0 | 0
0 | 1 | 1 | 1 | 1
1 | 0 | 0 | 0 | 0
1 | 0 | 0 | 1 | 1
1 | 0 | 1 | 0 | 1
1 | 0 | 1 | 1 | 1
1 | 1 | 0 | 0 | 0
1 | 1 | 0 | 1 | 0
1 | 1 | 1 | 0 | 1
1 | 1 | 1 | 1 | 1
```