Programming Assignment 2

In [ ]:
```
# Import what you need
```

Question 1

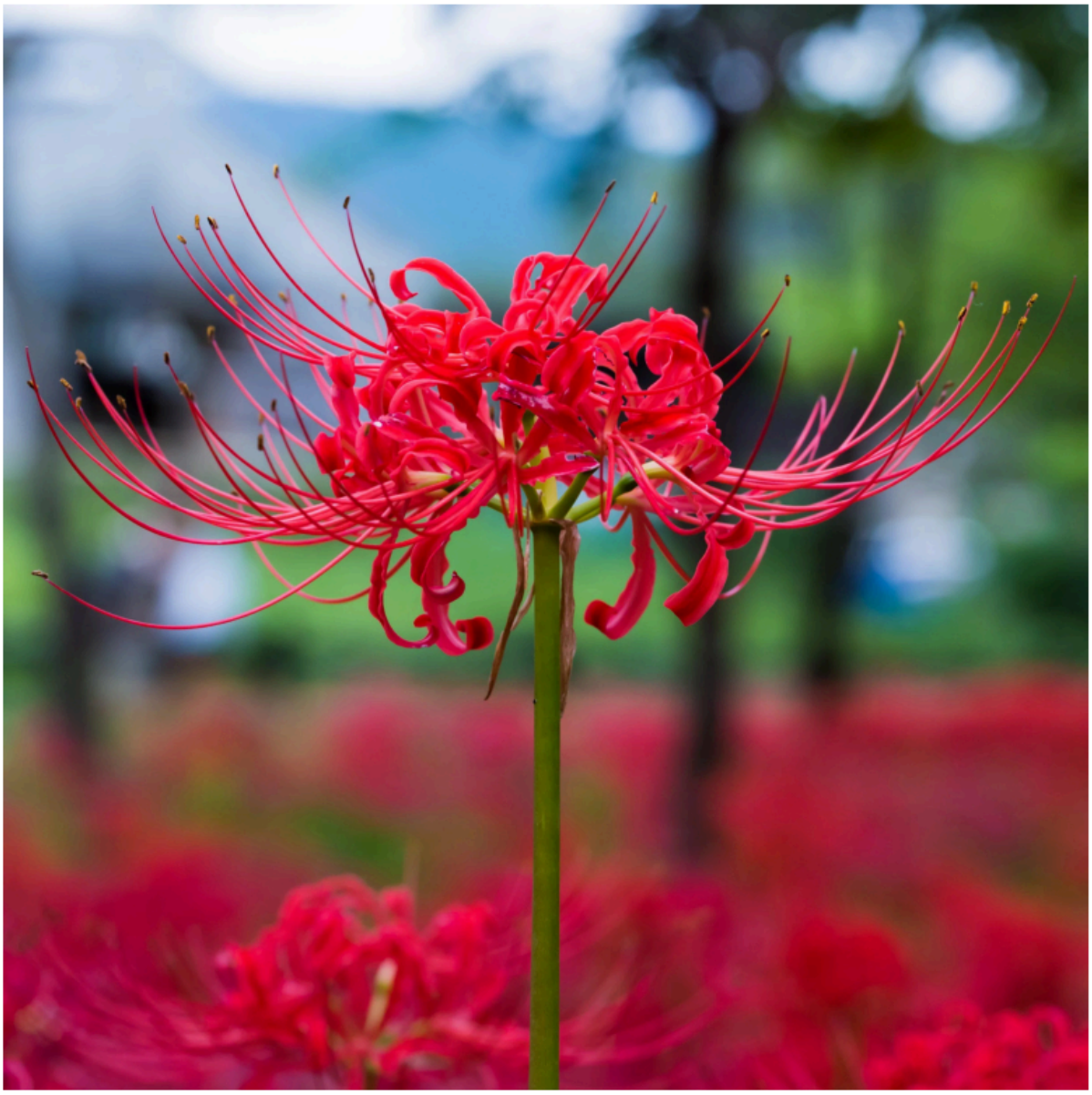
Study the Python notebook on image manipulation. Load an image from your working folder as a tensor of order 3. Perform the following on the images by simply manipulating this tensor. You are not allowed to use any specialized libraries or any specialized functions, only basic Python and Numpy routines.

> (A) Crop the image to 60% of its size symmetrically from the center. Display the resulting image.

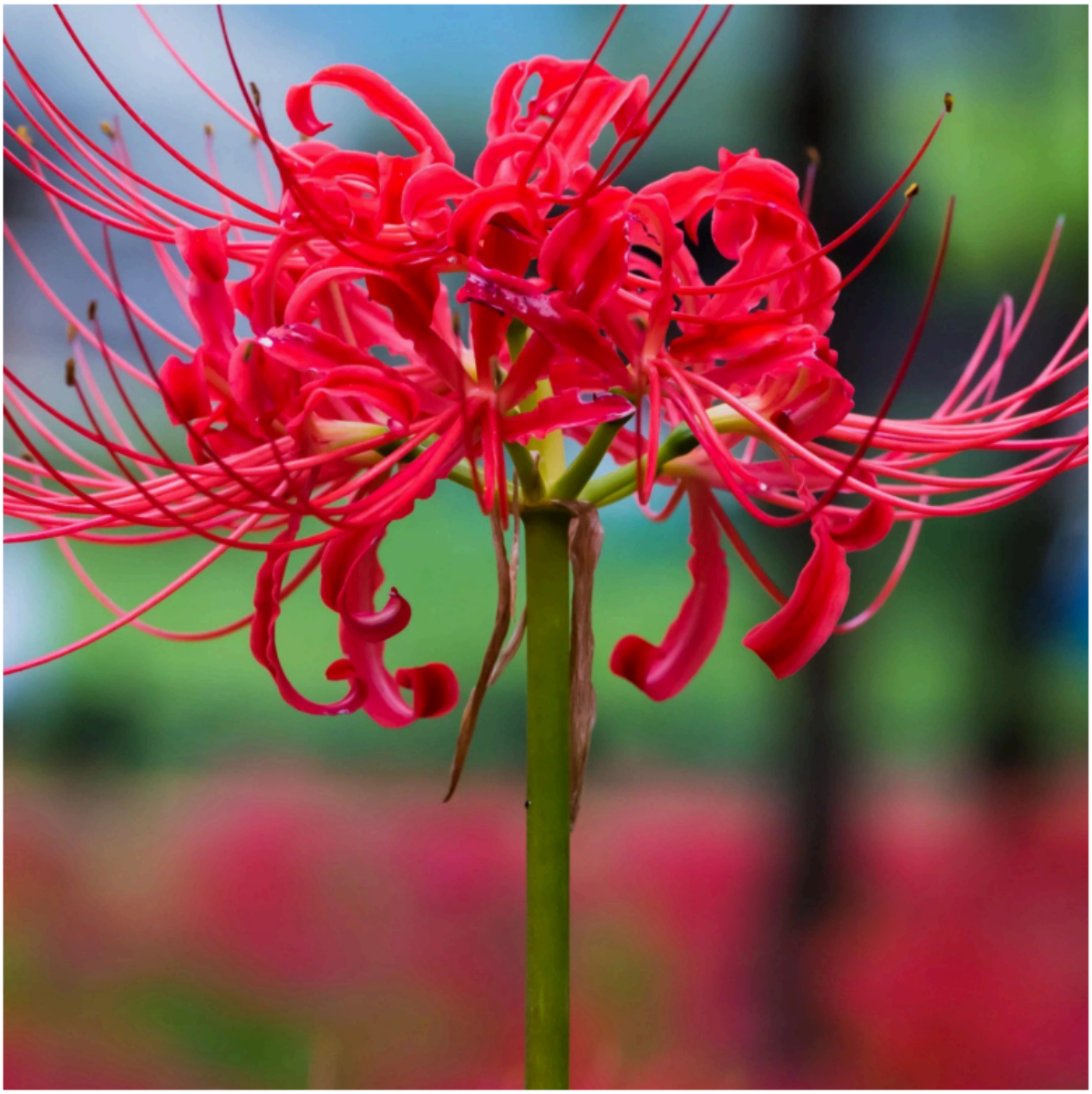
In [35]:
```
# Your code here
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

##THIS FUNC IS TAKEN FROM IMAGE MANIPULATION MODULE
def display_image(arr_image,color_map=None,dim=(10,20)):
    plt.figure(figsize=dim)
    plt.axis('off')
    plt.imshow(arr_image, cmap=color_map)
    plt.show()

pix = mpimg.imread('Lycoris.jpeg')

print("Tensor: ", pix.shape, "\n")
display_image(pix)

height, width, color = pix.shape
centerY = height/2
centerX = width/2

##new image's height
heightN = int(height * 0.6)
widthN = int(width * 0.6)
print(heightN,widthN)

##calculate the gaps on top and bottom
topGap = int((height - heightN) / 2)
sideGap = int((width - widthN) / 2)

## not allowed to use this cropping method;
#pixN = pix[ topGap: topGap+heightN , sideGap: sideGap+widthN, :]

pixL = np.zeros((heightN, widthN, color), dtype=np.uint8)
```

```
for i in range(heightN):
    for j in range(widthN):
        pixN[i,j] = pix[i+topGap,j+sideGap]


print("Tensor: ", pixN.shape, "\n")
display_image(pixN)
```

```
Tensor:  (2890, 2890, 3)
```



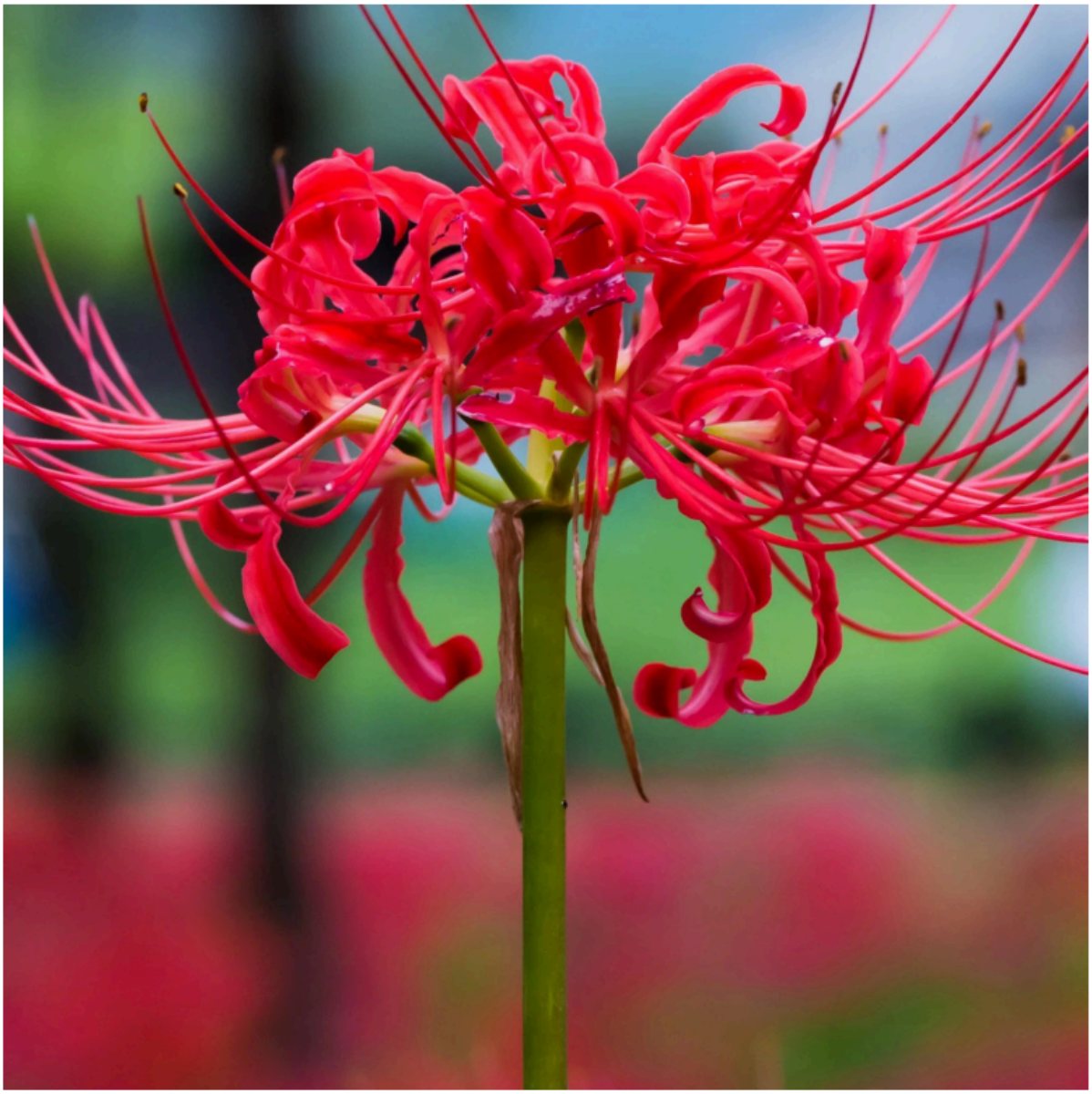
```
1734 1734
Tensor:  (1734, 1734, 3)
```



> (B) Display the image upon flipping it horizontally. Left becomes right and right becomes left.

In [65]:
```
# Your code here

pixL = np.zeros((heightN, widthN, color), dtype=np.uint8)

for i in range(heightN):
    for j in range(widthN):
        #reverse the second loop and erite from right ot left in to the new
        pixL[i,j] = pix[i+topGap,sideGap+widthN-j]


print("Tensor: ", pixL.shape, "\n")
display_image(pixL)
```

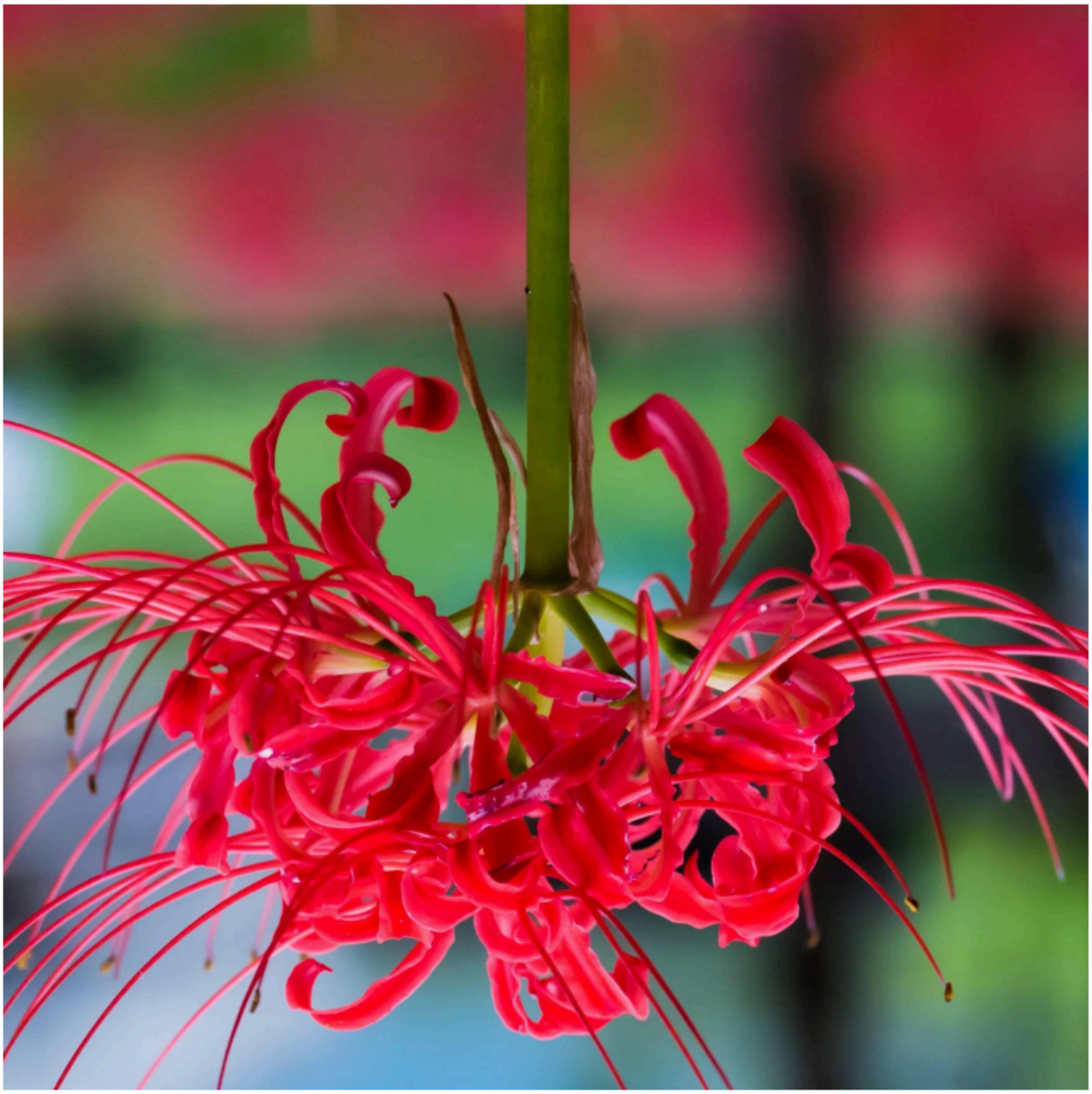
```
Tensor:  (1734, 1734, 3)
```



> (C) Display the image after flipping it vertically. Top becomes bottom and bottom becomes top.

In [69]:
```
# Your code here

pixT = np.zeros((heightN, widthN, color), dtype=np.uint8)

for i in range(heightN):
    for j in range(widthN):
        #reverse the second loop and erite from right ot left in to the new
        pixT[i,j] = pix[topGap+heightN-i,sideGap+j]


print("Tensor: ", pixT.shape, "\n")
display_image(pixT)
```

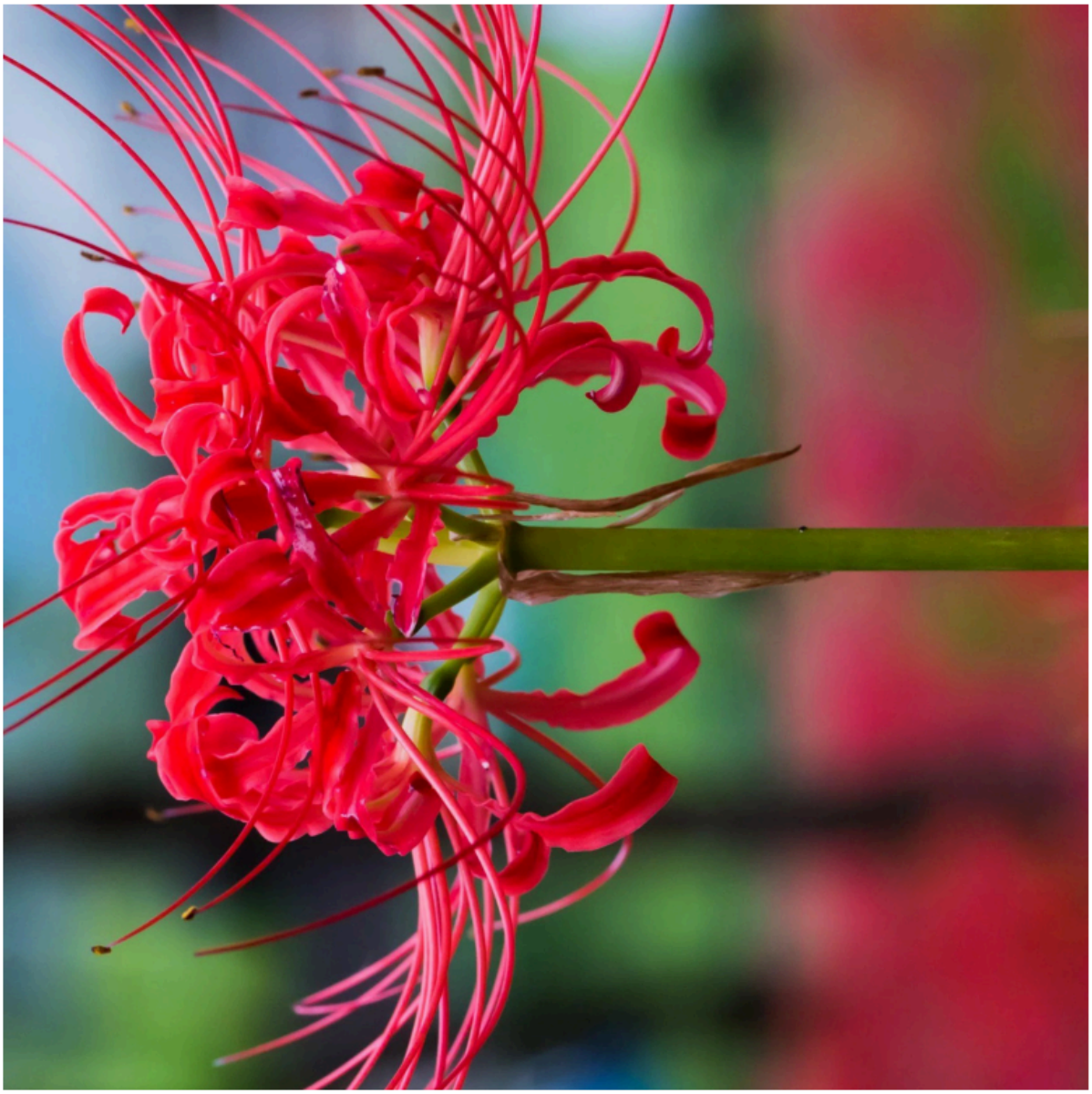
```
Tensor:  (1734, 1734, 3)
```



> (D) Rotate the image by 90° counter clock-wise. Display the resulting image.

In [73]:
```
# Your code here

pix90 = np.zeros((widthN, heightN, color), dtype=np.uint8)

for i in range(heightN):
    for j in range(widthN):
        #reverse the second loop and erite from right ot left in to the new
        pix90[j,i] = pix[topGap+i,sideGap+j]


print("Tensor: ", pix90.shape, "\n")
display_image(pix90)
```

```
Tensor:  (1734, 1734, 3)
```



Question 2

Use the code provided in the Python notebook on linear systems and answer the following.

> (A) **Modify** the code in the function for Gaussian elimination so that it also allows row exchanges in general. Solve the following system by using your code.
>
> $$\begin{bmatrix} 0 & 1 & 2 \\ 5 & -2 & 6 \\ 3 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 9 \\ 2 \end{bmatrix}$$

In [75]:

```
# Gauss-Jordan Elimination (without row-swapping)
def findRREF(A):
    '''
    Input: a general rectangular matrix
    Output: the row reduced echelon form of the matrix A
    by using Gauss-Jordan elimination
    '''
    augA = np.copy(A)
    m,n=augA.shape
    print("Augmented matrix: \n",augA)
    for k in range(m):
        # Check that the matrix is not rank deficient
        if np.abs(augA[k,k]) < 10**(-15):
            for i in range(k+1, m):

                ## use > 10**(-15 to check if the [i,k] is 0, if so swap the
                    if np.abs(augA[i, k]) > 10**(-15):
                        augA[[k, i]] = augA[[i, k]]
                        print('Swapped rows',i,'and',k)
                        break

                ####
            else:
                exit("The given matrix is singular or requires row swapping"
        # Convert the pivot element to 1
        augA[k,:] = augA[k,:] /augA[k,k]

        for i in range(m):
            if i==k:
                continue
            z = -augA[i,k]
            # Change the entire rows (k+1)st onward
            augA[i,:] = augA[i,:] + z*augA[k,:]
        print("Pass {}:\n".format(k+1))
        print(augA)
    return augA

np.set_printoptions(precision=4)
A = np.array([[0,1,2,3],
              [5,-2,6,9],
              [3,1,-2,2]],dtype=float)
augA_modified = findRREF(A)
# The last column is the solution
soln = augA_modified[:,3:]
print("\n The Solution is \n",soln)
```

```
Augmented matrix:
 [[ 0.  1.  2.  3.]
 [ 5. -2.  6.  9.]
 [ 3.  1. -2.  2.]]
Swapped rows 1 and 0
Pass 1:

[[ 1.  -0.4  1.2  1.8]
 [ 0.   1.   2.   3. ]
 [ 0.   2.2 -5.6 -3.4]]
Pass 2:

[[  1.   0.   2.   3.]
 [  0.   1.   2.   3.]
 [  0.   0. -10. -10.]]
Pass 3:

[[ 1.  0.  0.  1.]
 [ 0.  1.  0.  1.]
 [-0. -0.  1.  1.]]

 The Solution is
 [[1.]
 [1.]
 [1.]]
```

> (B) **Use or modify** the provided code to write a function for finding the inverse of any square matrix. Use this to find the inverse of the following. Show the output.

In [98]:

```python
# Your code here
def invByRREF(A):
    m,n=A.shape
    if m!=n:
        exit("To find inverse, please provide a square matrix only.")
    # Appropriate augmented matrix. One line
    augA = np.copy(A)
    modified_augA = findRREF(augA)
    # Take appropriate slice of modified_augA for inverse
    print('RRREF form:\n',modified_augA)
    invA = np.zeros((m,n))
    for i in range(m):
        for j in range(n):
            invA[j,i] = modified_augA[i,j]

    return invA

A = np.array([[1,1,4],
              [3,2,4],
              [1,1,6]],dtype=float)
inv = invByRREF(A)
print('Inverted matrix: \n',inv)
```

```
Augmented matrix:
 [[1. 1. 4.]
 [3. 2. 4.]
 [1. 1. 6.]]
Pass 1:

[[ 1.  1.  4.]
 [ 0. -1. -8.]
 [ 0.  0.  2.]]
Pass 2:

[[ 1.  0. -4.]
 [-0.  1.  8.]
 [ 0.  0.  2.]]
Pass 3:

[[ 1.  0.  0.]
 [-0.  1.  0.]
 [ 0.  0.  1.]]
RRREF form:
 [[ 1.  0.  0.]
 [-0.  1.  0.]
 [ 0.  0.  1.]]
Inverted matrix:
 [[ 1. -0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

> (C) **Modify** the code to write a function for finding the determinant of a general square matrix. Show the output for a matrix of random integer entries (6×6). Use the built-in function np.linalg.det() to find the determinant of the same matrix for comparison.

In [142…

```python
# Your code here
def find_det(A):
    m,n = A.shape
    mA = np.copy(A)
    det = 0
    if m!=n:
        exit("To find determinant, please provide a square matrix only.")

    ##base case for recursion to stop
    if m == 2:
        return (mA[0,0]*
                mA[1,1]-
                mA[0,1]*
                mA[1,0])
    else:
        for i in range(m):
            ## (-1)**i is for changing sign
            det = det + (-1)**i * mA[0,i] * find_det(subMatrix(i,mA))
    return det
```

```
#next sub matrix ignoring the kth column, i didnt want to use np.delete
def subMatrix(k,A):
    m,n = A.shape
    mA = np.copy(A)
    subM = np.zeros((m-1,n-1))

    for i in range(1, m):
        cInx= 0; #column index for subM
        for j in range(n):
            if(k==j):
                continue
            subM[i-1,cInx] = mA[i,j]
            cInx +=1
    return subM

A = np.random.rand(6,6)

print(find_det(A))
print (np.linalg.det(A))
```

```
-0.05599039080043465
-0.05599039080043467
```

In [132…

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [ ]: