In [105…
```
# cumproduct:
np.cumproduct(b)
```

Out[105…
```
array([    10,    400,  26000,  52000, 624000])
```

In [106…
```
# repeat():
b = np.array([10,40,65,2,12])
b
```

Out[106…
```
array([10, 40, 65,  2, 12])
```

In [108…
```
# repeats each element for the given no.of times.
np.repeat(b,3)
```

Out[108…
```
array([10, 10, 10, 40, 40, 40, 65, 65, 65,  2,  2,  2, 12, 12, 12])
```

In [110…
```
# tile():repeats the whole seq for given no.of times.
np.tile(b,3)
```

Out[110…
```
array([10, 40, 65,  2, 12, 10, 40, 65,  2, 12, 10, 40, 65,  2, 12])
```

In [113…
```
# where():returns the index of specific element.
x = np.array([1,2,34,3,5,4,5,4,3,7,3,9,3])
np.where(x==3)
```

Out[113…
```
(array([ 3,  8, 10, 12], dtype=int64),)
```

In [1]:
```
import numpy as np
```

In [2]:
```
a = np.array([[1,2],[3,4]])
b = np.array([[4,5],[3,2]])
print(a)
print(b)
```

```
[[1 2]
 [3 4]]
[[4 5]
 [3 2]]
```

In [3]:
```
# Dot product:-
np.dot(a,b)
```

Out[3]:
```
array([[10,  9],
       [24, 23]])
```

In [4]:
```
# Cross Product:-
a = np.array([[1,2],[3,4]])
b = np.array([[4,5],[3,2]])
print(a)
print(b)
```

```
[[1 2]
 [3 4]]
[[4 5]
 [3 2]]
```

In [5]:
```
np.cross(a,b)
```

Out[5]:
```
array([-3, -6])
```

In [7]:
```
# transpose:- converting rows to columns and vice-versa.
a
```

Out[7]:
```
array([[1, 2],
       [3, 4]])
```

In [8]:
```
np.transpose(a)
```

Out[8]:
```
array([[1, 3],
       [2, 4]])
```

In [9]:
```
a.T
```

Out[9]:
```
array([[1, 3],
       [2, 4]])
```

In [11]:
```
# Indexing on Arrays:-
x = np.array([12,13,14,15,16,17])
x
```

Out[11]:
```
array([12, 13, 14, 15, 16, 17])
```

In [12]:
```
x[2]
```

Out[12]:
```
14
```

In [13]:
```
# Slicing:
x = np.array([12,13,14,15,16,17])
x[2:]
```

Out[13]:
```
array([14, 15, 16, 17])
```

In [14]:
```
x[::-1]
```

Out[14]:
```
array([17, 16, 15, 14, 13, 12])
```

In [15]:
```
x1 = np.array([[3,4,5],[12,16,17],[20,25,30]])
x1
```

Out[15]:
```
array([[ 3,  4,  5],
       [12, 16, 17],
       [20, 25, 30]])
```

In [18]:
```
# Indexing: varname[row_index,column_index]
x1[0][2]
```

Out[18]:
```
5
```

In [17]:
```
x1[0,2]
```

Out[17]:
```
5
```

In [20]:
```
# by default returns all the values of given row_index.
x1[1]
```

Out[20]:
```
array([12, 16, 17])
```

In [21]:
```
# Slicing:-
# varname[row_slice , column_slice]
x1
```

Out[21]:
```
array([[ 3,  4,  5],
       [12, 16, 17],
       [20, 25, 30]])
```

In [22]:
```
x1[1:3,0:2]
```

Out[22]:
```
array([[12, 16],
       [20, 25]])
```

In [23]:
```
# 2D array:
b = np.random.randint(1,20,(5,5))
b
```

Out[23]:
```
array([[18, 16,  9, 10, 13],
       [13, 17, 16,  4,  1],
       [ 6, 19, 10,  5,  7],
       [ 8, 19,  9, 10, 19],
       [ 6, 15,  3,  9, 12]])
```

In [24]:
```
# to print only odd rows and columns:-
b[1::2,1::2]
```

Out[24]:
```
array([[17,  4],
       [19, 10]])
```

In [26]:
```
# all rows and even columns:-
b[::,::2]
```

Out[26]:
```
array([[18,  9, 13],
       [13, 16,  1],
       [ 6, 10,  7],
       [ 8,  9, 19],
       [ 6,  3, 12]])
```

In [27]:
```
# Concatenation of arrays:- joining of 2 matrix.
a = np.random.randint(1,20,(2,2))
b = np.random.randint(1,20,(2,4))
c = np.random.randint(1,20,(4,4))
```

In [28]:
```
print(a)
print(b)
print(c)
```

```
[[ 4  7]
 [10 12]]
[[ 3 13  1 10]
 [ 1 12  3 16]]
[[17 17 19 17]
 [10 14 15 18]
 [11 11 17  8]
 [ 8 11 11 17]]
```

In [29]:
```
# To concatenate along the row the column size should be same.
# row - axis 0 (by default)
np.concatenate((b,c),axis=0)
```

Out[29]:
```
array([[ 3, 13,  1, 10],
       [ 1, 12,  3, 16],
       [17, 17, 19, 17],
       [10, 14, 15, 18],
       [11, 11, 17,  8],
       [ 8, 11, 11, 17]])
```

In [30]:
```
np.concatenate((b,c))
```

Out[30]:
```
array([[ 3, 13,  1, 10],
       [ 1, 12,  3, 16],
       [17, 17, 19, 17],
       [10, 14, 15, 18],
       [11, 11, 17,  8],
       [ 8, 11, 11, 17]])
```

In [31]:
```
np.concatenate((a,b))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[31], line 1
----> 1 np.concatenate((a,b))

ValueError: all the input array dimensions except for the concatenation axis must ma
tch exactly, but along dimension 1, the array at index 0 has size 2 and the array at
index 1 has size 4
```

In [33]:
```
# to concatenate along columns the row size should be same.
# column - axis -1
np.concatenate((a,b),axis=1)
```

Out[33]:
```
array([[ 4,  7,  3, 13,  1, 10],
       [10, 12,  1, 12,  3, 16]])
```

In [36]:
```
# Stacking: joining of array along the vertical axis or horizontal axis.
# Horizontal Stack: along columns
np.hstack((a,b))
```

Out[36]:
```
array([[ 4,  7,  3, 13,  1, 10],
       [10, 12,  1, 12,  3, 16]])
```

In [37]:
```
# Vertical Stack: along rows
np.vstack((b,c))
```

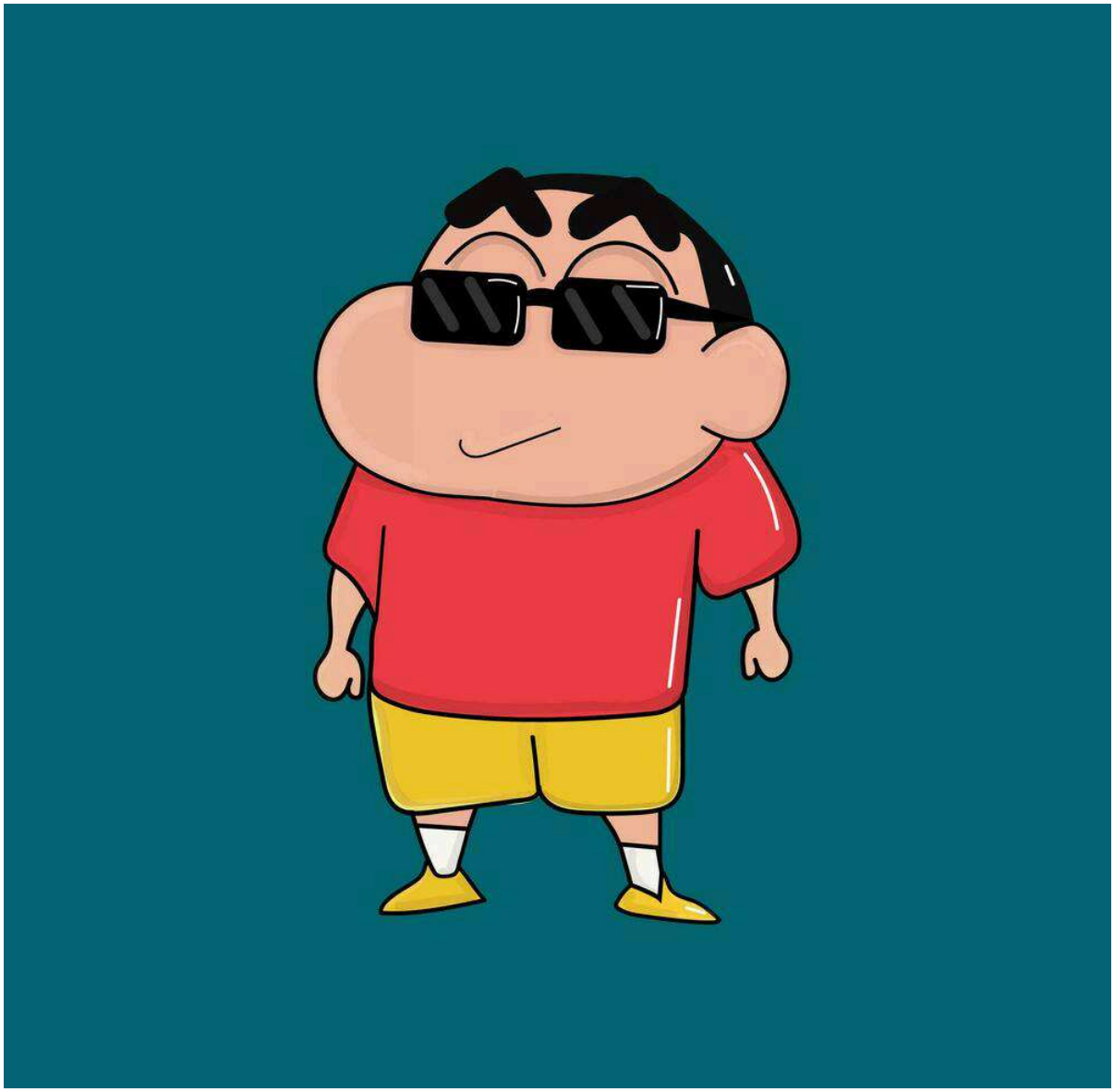
Out[37]:
```
array([[ 3, 13,  1, 10],
       [ 1, 12,  3, 16],
       [17, 17, 19, 17],
       [10, 14, 15, 18],
       [11, 11, 17,  8],
       [ 8, 11, 11, 17]])
```

PIL :- Python Imaging Library

In [38]:
```
# importing the library:
from PIL import Image
```

In [40]:
```
# To open the image file:-
img = Image.open(r"C:\Users\CTTC\Downloads\shinchan-cool-pose-with-sunglasses-free-
img
```

Out[40]:



In [43]:
```
# format: returns the extension of the image
img.format
```

Out[43]: 'JPEG'

In [44]:
```
type(img)
```

Out[44]: PIL.JpegImagePlugin.JpegImageFile

In [47]:
```
# Color Channel:
img.mode
```

Out[47]: 'RGB'

In [48]:
```
# size of the image:
img.size
```

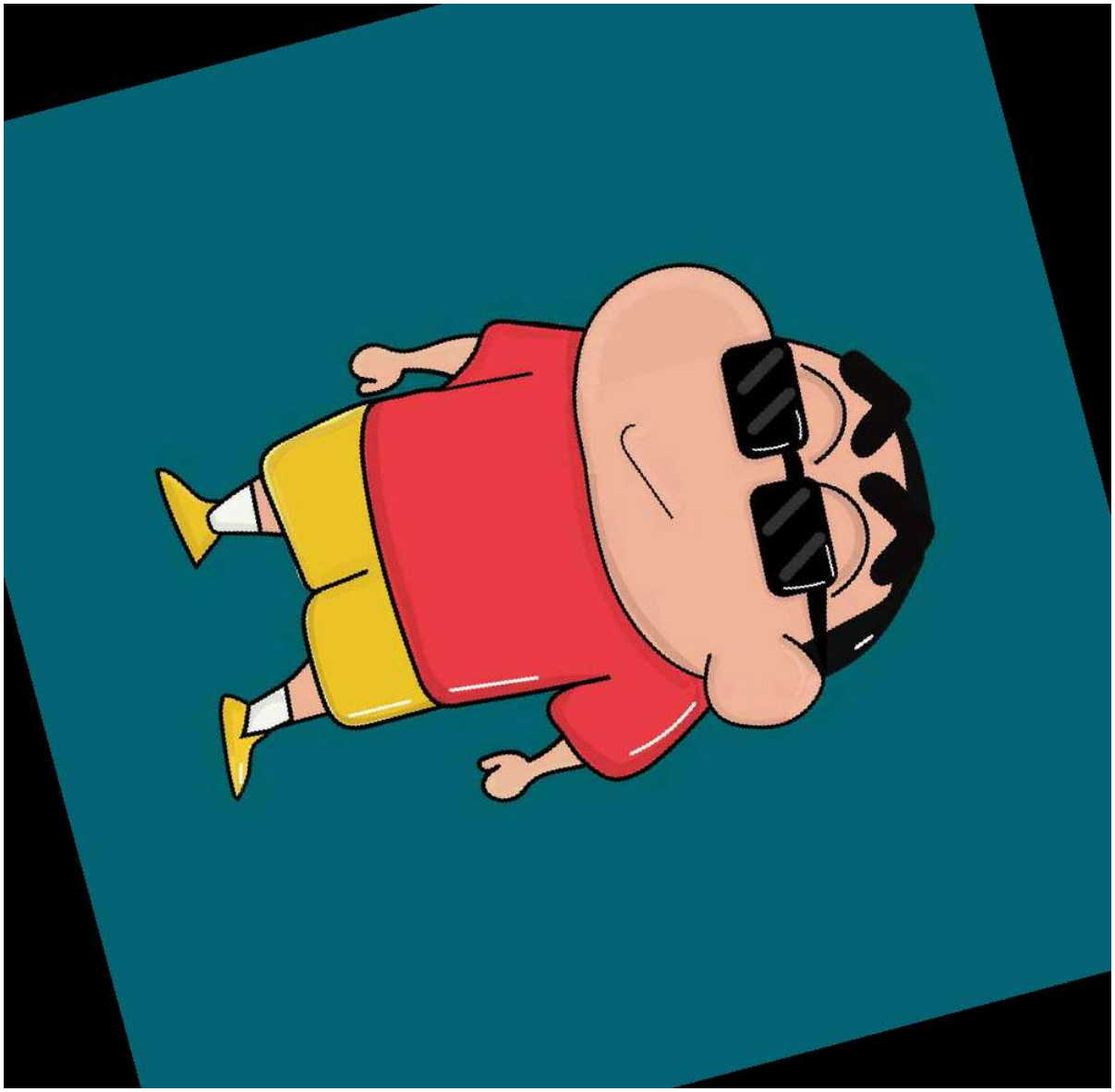
Out[48]: (1000, 980)

In [49]:
```
# to rotate the image:-
img.rotate(180)
```

Out[49]:

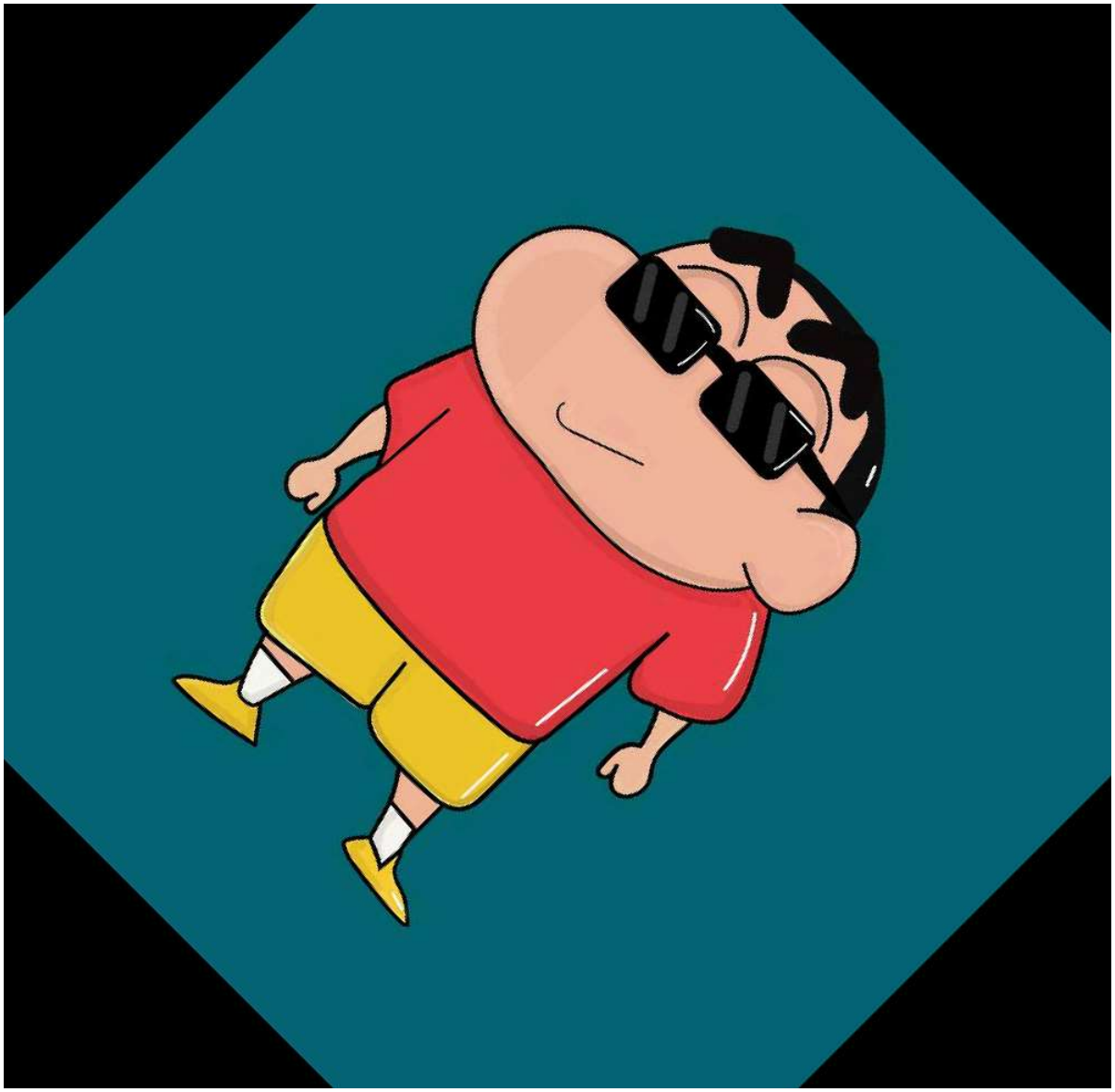
In [51]:
```
img.rotate(3525)
```

Out[51]:



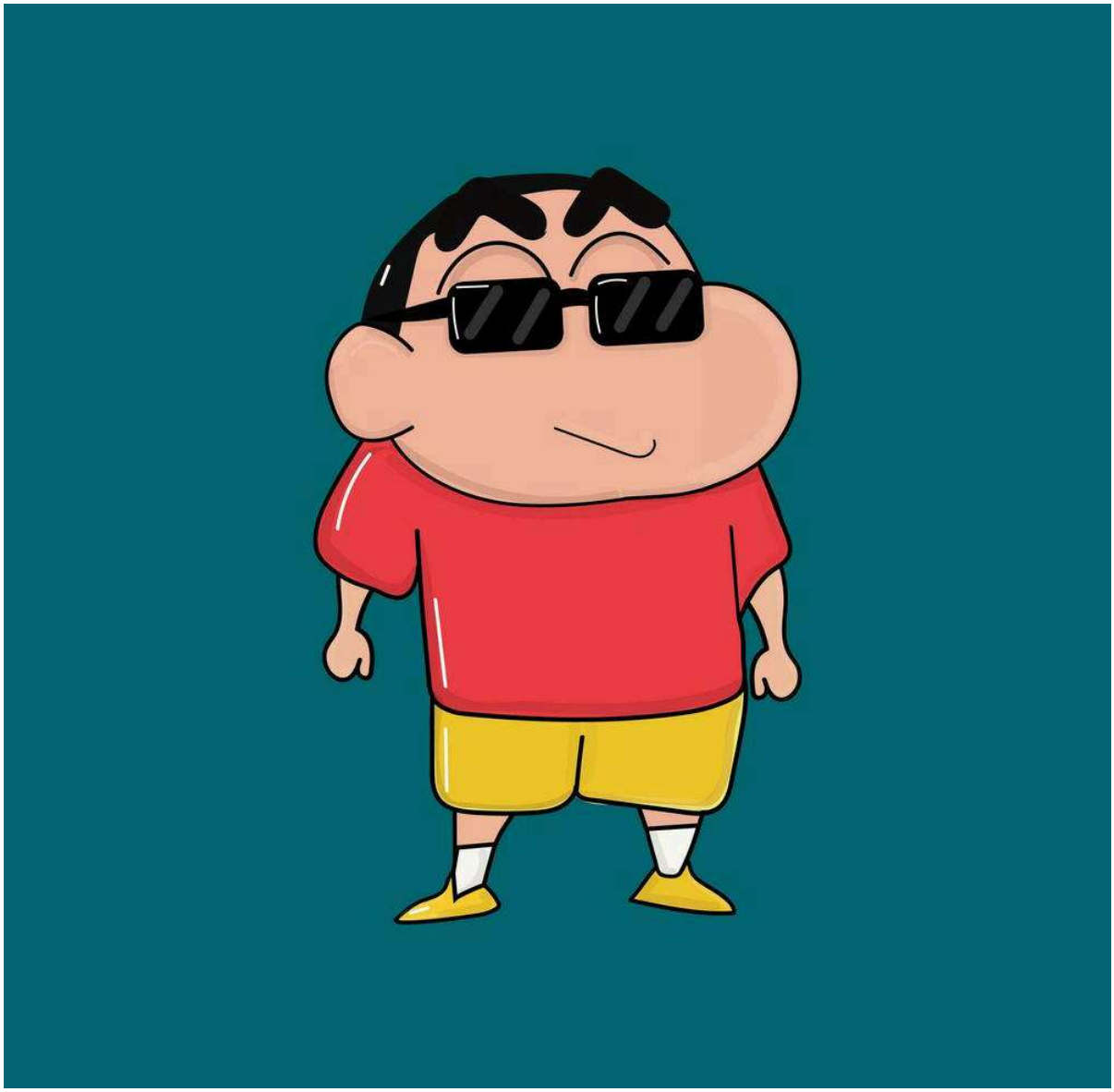
In [52]:
```
img.rotate(-45)
```

Out[52]:



In [54]:
```
# mirror the image:-
# Horizontal Flip/Mirroring:
img.transpose(Image.FLIP_LEFT_RIGHT)
```

Out[54]:

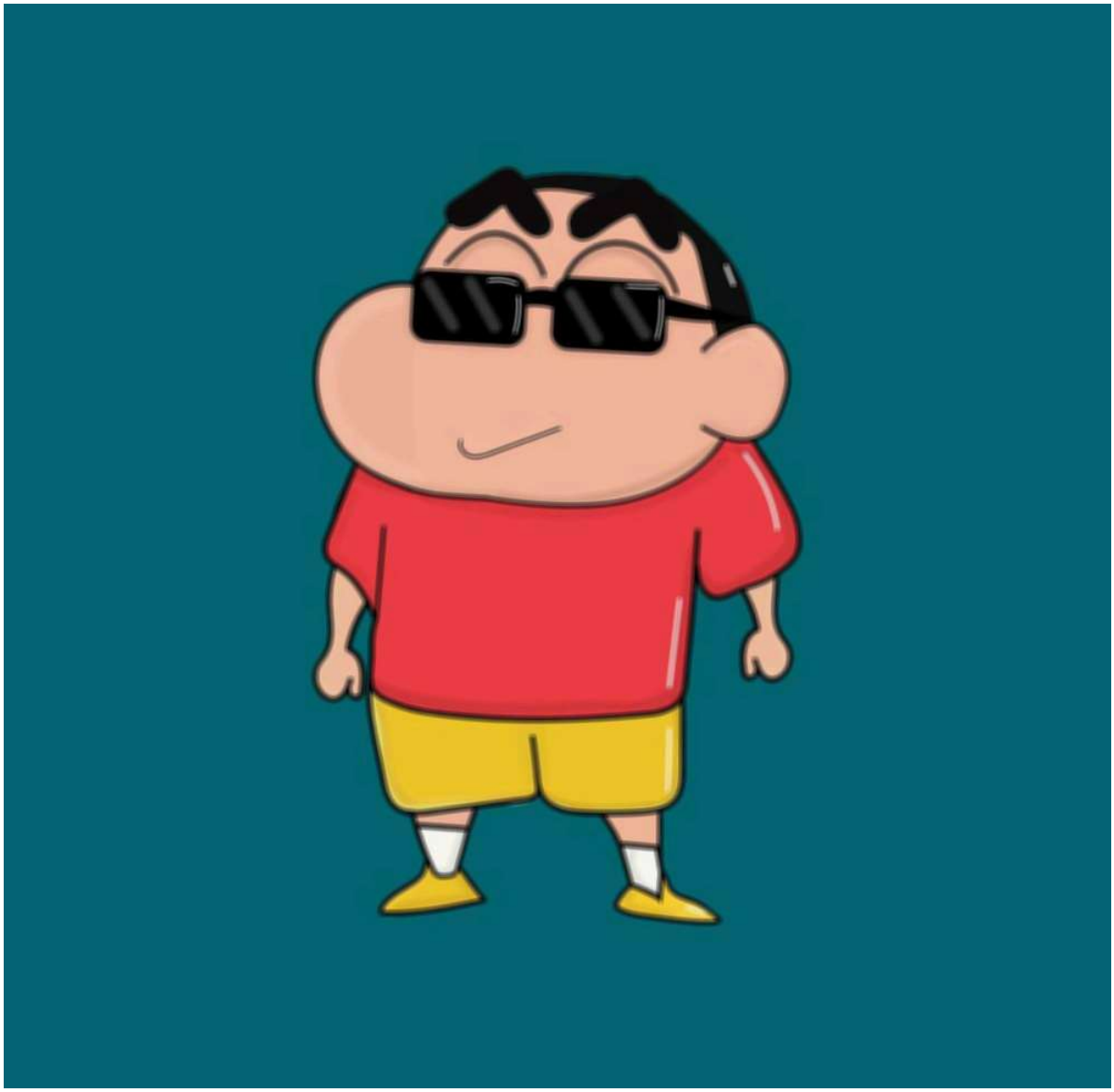


In [55]:
```
# Vertical Flip/Mirroring:
img.transpose(Image.FLIP_TOP_BOTTOM)
```

Out[55]:

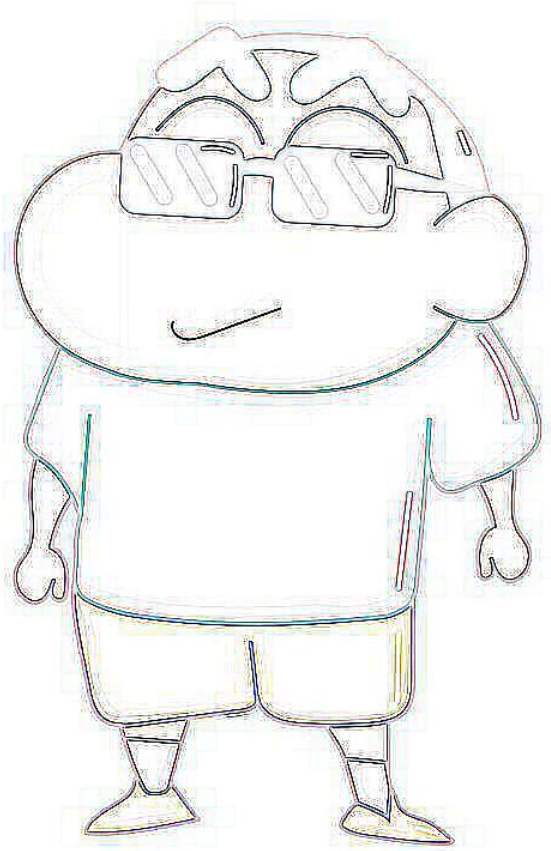
In [56]:
```
# Applying filters to Image:-
from PIL import ImageFilter
img.filter(ImageFilter.BLUR)
```

Out[56]:



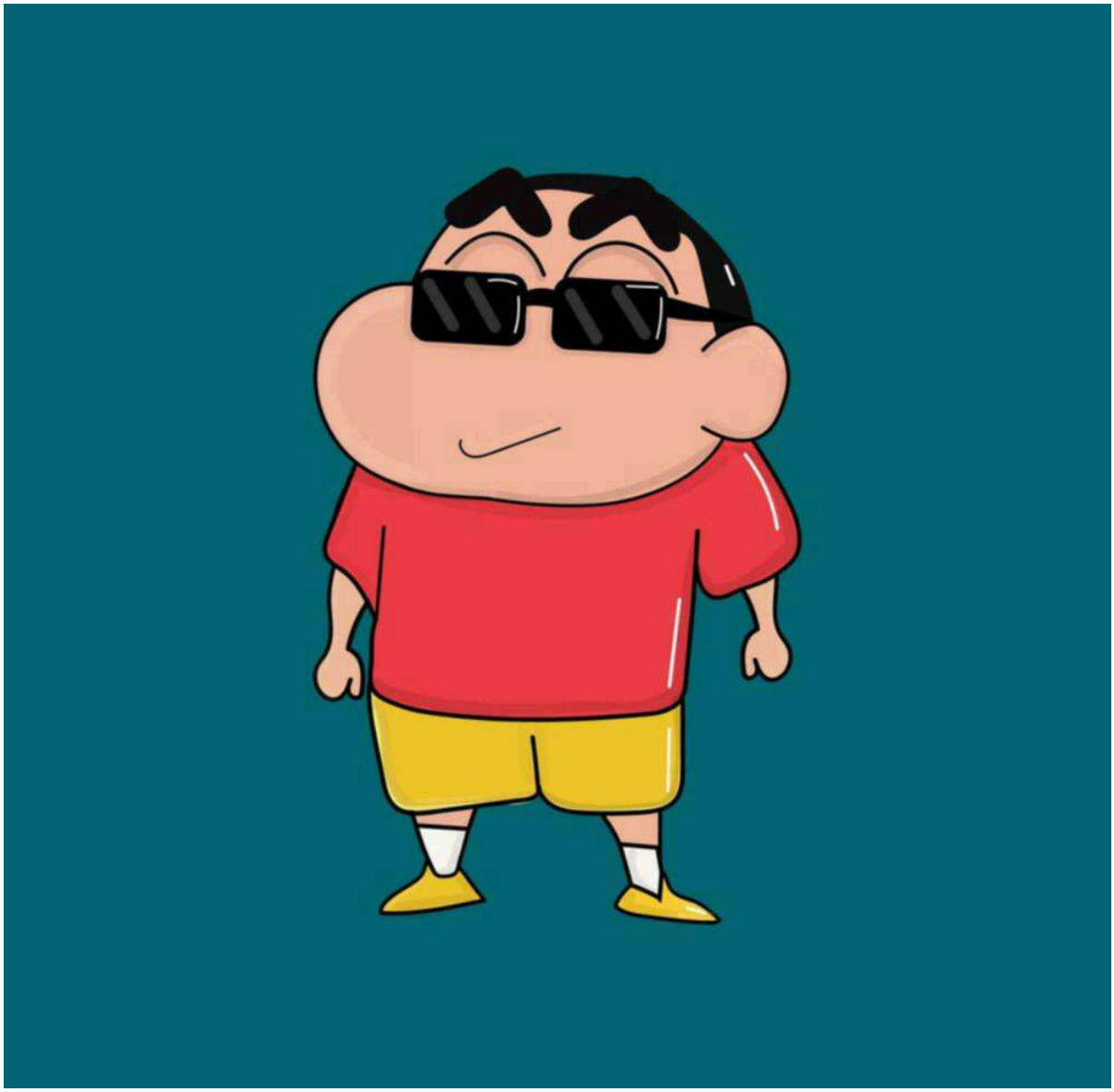
In [57]:
```
img.filter(ImageFilter.CONTOUR)
```

Out[57]:



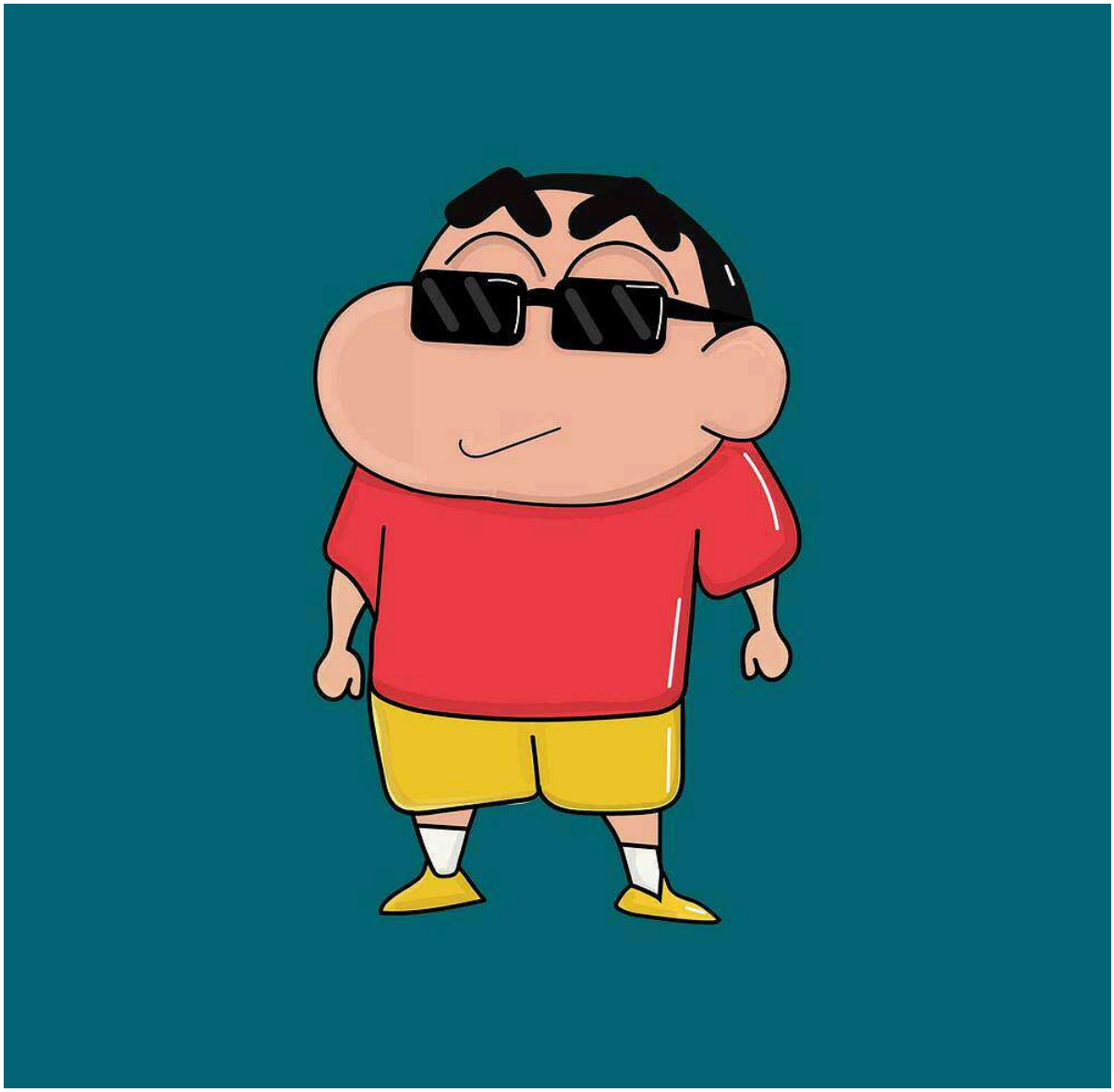
In [58]:
```
img.filter(ImageFilter.SMOOTH)
```

Out[58]:



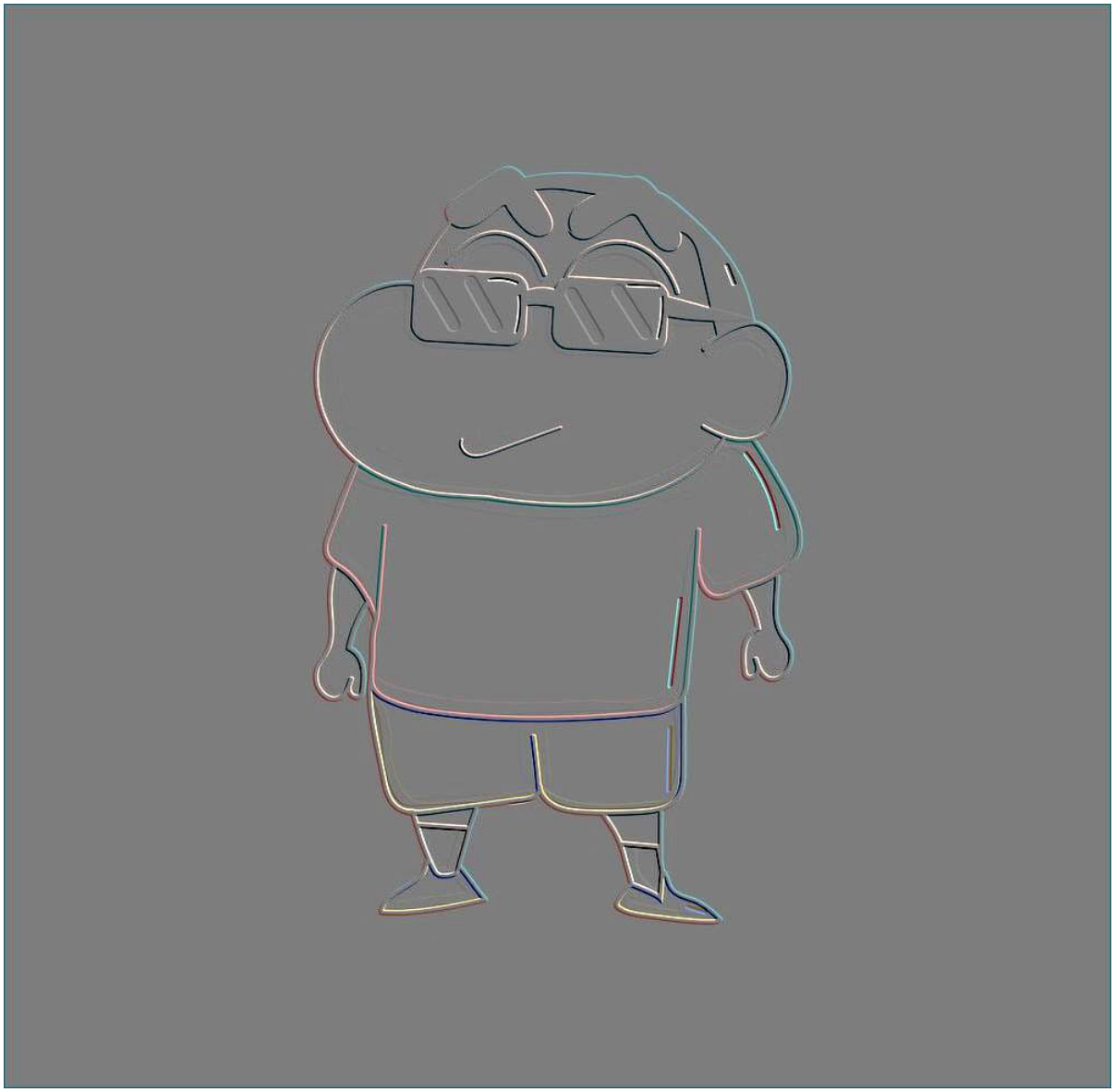
In [59]:
```
img.filter(ImageFilter.DETAIL)
```

Out[59]:



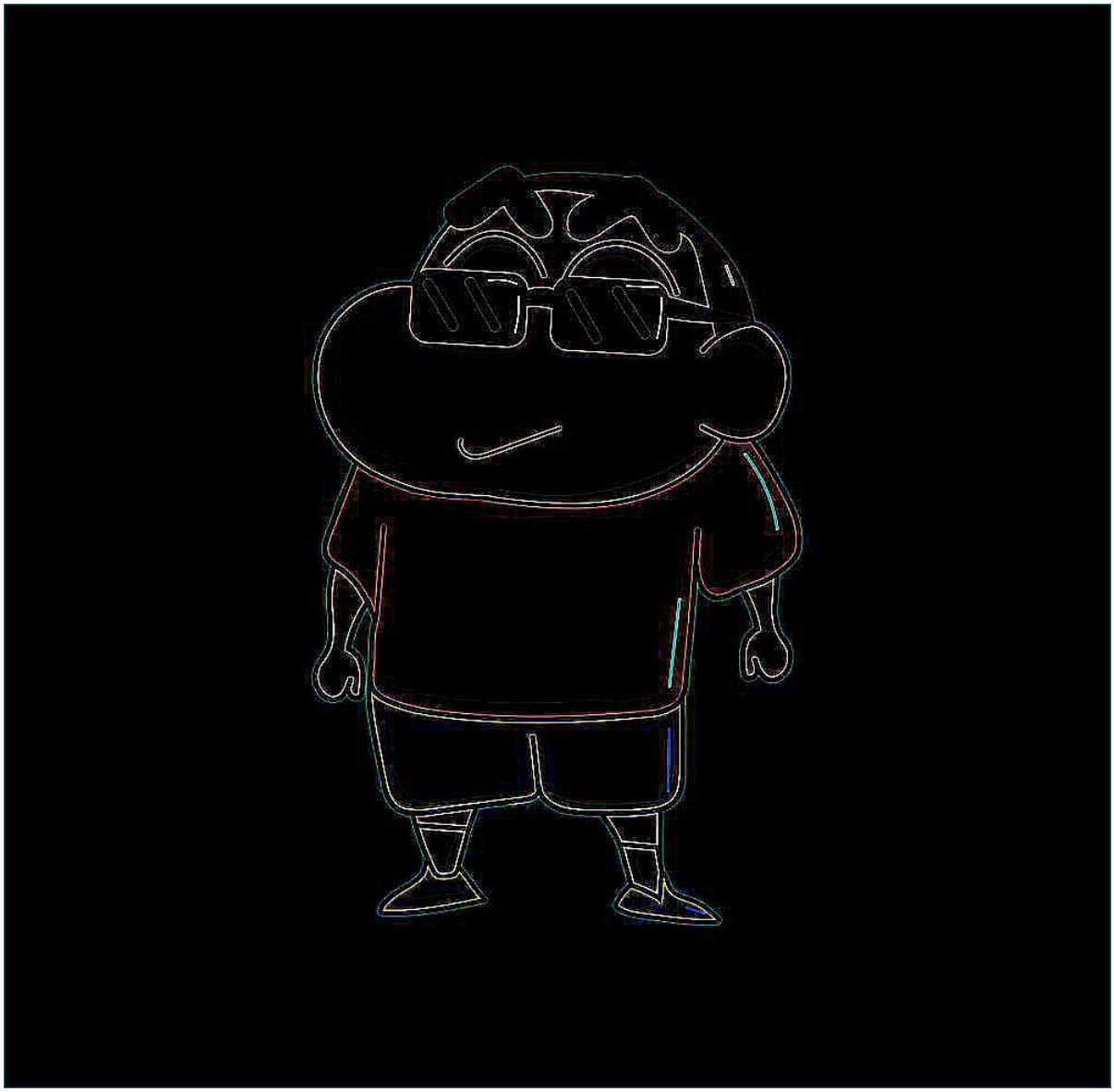
In [60]:
```
img.filter(ImageFilter.EMBOSS)
```

Out[60]:



In [61]:
```
img.filter(ImageFilter.FIND_EDGES)
```

Out[61]:



In [62]:
```
# To resize the image:-
img1 = img.resize((500,250))
img1
```

Out[62]:

In [63]:
```
img1.save(r"C:\Users\CTTC\Downloads\modified_img.jpg")
```

PANDAS:- Panel Data Analysis

It is a python library that helps to deal with analyzing and manipulating the datas.
We can also read datasets using pandas.
Pandas is 2 types:-

- Series (single column)
- DataFrame (multiple columns)

In [64]:
```
# importing the library:
import pandas as pd
```

In [65]:
```
# Creating a Series:-
s = pd.Series(['a','b','c','d'])
s
```

Out[65]:
```
0    a
1    b
2    c
3    d
dtype: object
```

In [66]:
```
# assigning index values to the series.
s1 = pd.Series(['a','b','c'],index=[10,20,30])
s1
```

Out[66]:
```
10    a
20    b
30    c
dtype: object
```

In [67]:
```
# Indexing in Series:-
s1[20]
```

Out[67]:
```
'b'
```

In [69]:
```
# Slicing:
s1[::2]
```

Out[69]:
```
10    a
30    c
dtype: object
```

In [70]:
```
# Creating a Series from tuple:
s2 = ('apple','bat','cow','dog')
```

```
type(s2)
```

Out[70]:
```
tuple
```

In [71]:
```
pd.Series(s2)
```

Out[71]:
```
0    apple
1      bat
2      cow
3      dog
dtype: object
```

In [73]:
```
# Creating a Series from set
s = {100,200,300,400,500}
pd.Series(s)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[73], line 3
      1 # Creating a Series from set
      2 s = {100,200,300,400,500}
----> 3 pd.Series(s)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\series.py:512, in Series.__init__(self, data, index, dtype, name, copy, fastpath)
    510         data = data.copy()
    511 else:
--> 512     data = sanitize_array(data, index, dtype, copy)
    514     manager = get_option("mode.data_manager")
    515     if manager == "block":

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\construction.py:641, in sanitize_array(data, index, dtype, copy, allow_2d)
    632     return sanitize_array(
    633         data,
    634         index=index,
   (...)
    637         allow_2d=allow_2d,
    638     )
    640 else:
--> 641     _sanitize_non_ordered(data)
    642     # materialize e.g. generators, convert e.g. tuples, abc.ValueView
    643     data = list(data)

File C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\construction.py:692, in _sanitize_non_ordered(data)
    688 """
    689 Raise only for unordered sets, e.g., not for dict_keys
    690 """
    691 if isinstance(data, (set, frozenset)):
--> 692     raise TypeError(f"'{type(data).__name__}' type is unordered")

TypeError: 'set' type is unordered
```

In [74]:
```
# Creating a Series using dictionary:
d = pd.Series({1:'apple',2:'kiwi',3:'orange'})
```

In [75]:
```
d
```

Out[75]:
```
1     apple
2      kiwi
3    orange
dtype: object
```

In [ ]:
```

```