```python
#!/bin/env python3

# Man må installere python3.6, Pillow og numpy ved hjelp av pip install
# Altså etter man har installert python3.6 så må man skrive disse i et kommandovindu
#
# pip install Pillow
# pip install numpy
#
# Da skal det funke, hvis ikke spør meg
from PIL import Image
import numpy as np
import os
import json
import math
import re
import subprocess

##################################################################################################
# Velg om du vil kjøre en full analyse av et helt bilde eller valgt blokk
FULL_ANALYZE = True

# Endre tallet for å velge metode!
METHODE = 1

# Skriv inn navnet på bildefilen. NB! Filen må ligge i mappen "fourier_bilder"
IMAGE_NAME = "natur.png"

# Antall ledd i cosinusrekkene
AMOUNT_OF_TERMS = 30

##################################################################################################
# HVIS IKKE FULL ANALYSE, bruk settinger under!
# Velg om du vil generere piecewise for maple eller lage et bilde fra psi(t) funksjonen.
GENERATE_PIECEWISE_BOOL = True

# Velg perioden til piecewise-kommandoen. Det vi har brukt før er 64
PERIOD = 64

# Velg indeksene for 8x8 blokken dere vil analysere! 0, 0 er den blokken øverst til venstre.
BLOCK_INDEXES = ( 22, 26 ) # VELG HVILKEN BLOKK AV 8x8 DERE VIL UNDERSØKE. (x, y)
# natur (22,26)
# kunstig (11,11)

# Skriv inn en factor som bildet blir skalert med når det genereres slik at det er enklere å analysere
SCALE_FACTOR = 20


# Her skrive man inn cosinusuttrykket fra maple! Sørg for at det ser riktig ut og at verdien fra
# cosinusuttrykket blir returnert fra funksjonen. DETTE ER BARE HVIS ANALYSE AV BLOKKER.
# Bytt ut "255*cos(.4*t)" med det som kommer ut fra maple!
def psi(t):
    return 87.31250000-3.566504710*cos(.1472621557*t)+141.0276691*cos(0.4908738522e-1*t)+
    .985911939*cos(.2945243113*t)-30.20040468*cos(.1963495409*t)-19.82176616*cos(.2454369261*t)+
    64.69265187*cos(0.9817477044e-1*t)-5.785914227*cos(.4908738522*t)+11.87593921*cos(.3436116965*t)+
    9.888450648*cos(.3926990818*t)+1.725469068*cos(.4417864670*t)-8.730228779*cos(.8344855487*t)-
    8.052902122*cos(.5399612374*t)-3.278710526*cos(.5890486226*t)+4.938007688*cos(.6381360078*t)-
    5.852056005*cos(.7853981635*t)-2.248346107*cos(.8835729339*t)+6.159973590*cos(.9326603192*t)+
    7.475453779*cos(.9817477044*t)+8.610305090*cos(.6872233931*t)+3.258151057*cos(.7363107783*t)+
    1.287146206*cos(1.030835090*t)-4.999438596*cos(1.079922475*t)-5.420052033*cos(1.129009860*t)-
    3.142672782*cos(1.423534171*t)-2.335600308*cos(1.472621557*t)+3.729066048*cos(1.276272016*t)-
    1.094383682*cos(1.178097245*t)+2.971808007*cos(1.227184630*t)+1.643479624*cos(1.325359401*t)-
    1.312866830*cos(1.374446786*t)

##################################################################################################

IMAGE_DIR = os.path.realpath(os.path.join(os.path.dirname(__file__), '../bilder/fourier_bilder/'))
ANALYSERTE_BLOKKER_DIR = os.path.join(IMAGE_DIR, "analyserte_blokker/")

class pf:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'

   def format(text, l_pf):
       return l_pf + text + pf.END

def clamp(n, smallest, largest):
    return max(smallest, min(n, largest))
```

```
    # Returnerer en 8x8 som er et resultat av psi(t)
    def change_to_fourierseriesvalues(eight_by_eight, methode):
        try:
            value_array = ImageArrayToValues.convert(eight_by_eight, methode)
            T = len(value_array)
        except:
            print("Error: Vennligst velg en \"method\" funksjon.")
            return eight_by_eight

        for i in range(T):
            value_array[i] = clamp(int(psi(i)), 0, 255)

        return ValuesToImageArray.convert(value_array, methode)

    # Returnerer en 8x8 som er et resultat av psi_string
    def change_to_fourierseriesvalues_psi_string(eight_by_eight, methode, psi_string):
        try:
            value_array = ImageArrayToValues.convert(eight_by_eight, methode)
            T = len(value_array)
        except:
            print("Error: Vennligst velg en \"method\" funksjon.")
            return eight_by_eight

        for t in range(T):
            value_array[t] = clamp(int(eval(psi_string)), 0, 255)

        return ValuesToImageArray.convert(value_array, methode)

    # Bredde og høyde på arrayen må være delelig på 8
    def array_into_eight_by_eight(np_array):
        width = np_array.shape[0]
        height = np_array.shape[1]
        if(width % 8 > 0 or height % 8 > 0):
            return None

        # Les som en bok
        array_eight_by_eights = [ [] for i in range(int(height/8)) ]
        array_eight_by_eight_col_offset = 0

        for y_matrix in range(int(height/8)):
            n1 = np.arange(8*y_matrix, 8*y_matrix+8)
            for x_matrix in range(int(width/8)):
                n2 = np.arange(8*x_matrix, 8*x_matrix+8)
                array_eight_by_eights[array_eight_by_eight_col_offset].append(np_array[n2[None,:],n1[:,None]])

            array_eight_by_eight_col_offset += 1

        return array_eight_by_eights

    # Gjør om en 2d array med 8x8 blokker til en stor 2d array med pikselverdier
    def assembly_array_of_eight_by_eights(array_eight_by_eights):
        rows_eight_by_eight = []
        for row_eight_by_eight in array_eight_by_eights:
            if len(row_eight_by_eight) > 0:
                rows_eight_by_eight.append(np.concatenate(row_eight_by_eight, axis=1))
        return np.concatenate(rows_eight_by_eight, axis=0)

    # Gjør om en 2d array med pikselverdier til en png-bildefil
    def image_array_to_image(np_array, filepath):
        new_image = Image.fromarray(np_array,'L')
        with open(filepath, 'wb') as image_file:
            new_image.save(image_file, 'PNG')

    def image_array_to_image_scale(np_array, filepath, scalefactor):
        first_row = True
        scaled_array = None
        for row in np_array:
            combined_scaled_row = None
            first_value = True
            for value in row:
                scaled_value_array = np.full((scalefactor, scalefactor), value, dtype=np.uint8)
                combined_scaled_row =
                    np.concatenate([combined_scaled_row, scaled_value_array], axis=1)
                    if not first_value else scaled_value_array
                first_value = False

            scaled_array = np.concatenate([scaled_array, combined_scaled_row], axis=0) if not first_row else combined_scaled_row
            first_row = False

        new_image = Image.fromarray(scaled_array,'L')
        with open(filepath, 'wb') as image_file:
            new_image.save(image_file, 'PNG')

    # Gjør om en bildefil til en 2d array med pikselverdier (bare sort/hvitt)
    def image_to_image_array(loaded_image):
        converted_image_file = loaded_image.convert('L')
        return np.reshape(np.array(converted_image_file.getdata(), np.dtype(np.uint8)), (-1, converted_image_file.width))

    # Skriver en array til en json fil
    def write_array_to_datafile(filepath, np_array):
        with open(filepath, "w") as json_file:
            json_str = json.dumps(np_array.tolist())
```

```
            json_str = re.sub(r'([0-9]+\],)','\g<1>\n', json_str)
            json_str = re.sub(r'([^\-]\b[0-9]{1}\b)','   \g<1>',json_str)
            json_str = re.sub(r'([\-]\b[0-9]{1}\b)|([^\-]\b[0-9]{2}\b)','  \g<1>\g<2>',json_str)
            json_str = re.sub(r'([\-]\b[0-9]{2}\b)|([^\-]\b[0-9]{3}\b)',' \g<1>\g<2>',json_str)
            json_file.write(json_str)

    # Leser en json fil med en 2d array
    def read_array_from_datafile(filepath):
        res_json = ""
        with open(filepath, 'r') as json_file:
            res_json = json.load(json_file)
        return res_json

    # Ser om en mappe eksisterer og lager den hvis ikke den eksisterer. Returnerer dirpath
    def check_directory(dirpath):
        if not os.path.exists(dirpath):
            os.makedirs(dirpath)
        return dirpath

    # Gjør om 1d array med verdier til en piecewise kommando
    def array_to_piecewise(array, periode):
        res = "f(t):=piecewise("
        roffset = 0
        array_len = len(array)
        increment = periode/array_len
        for value in array:
            res += str(roffset) + "<=t<=" + str(roffset+increment) + "," + str(value) + ","
            roffset += increment
        res = res.strip().strip(',')
        return res + ")"

    # Gjør om 1d array med verdier til en piecewise kommando
    def array_to_piecewise_cli(array, periode):
        res = "f(t):=piecewise("
        roffset = 0
        array_len = len(array)
        increment = periode/array_len
        for value in array:
            res += str(roffset) + "<=t and t<=" + str(roffset+increment) + "," + str(value) + ","
            roffset += increment
        res = res.strip().strip(',')
        return res + ")"


    class ImageArrayToValues():
        def __init__(self):
            print("Class not supposed to be instantiated!")
            return None

        # Gjør om en array til en 1d array med pikselverdier. 3 ulike metoder (se "Prosjektoppgaven.mw")
        @staticmethod
        def __image_array_to_values_metode1(np_array):
            res = []
            for row in np_array:
                for value in row:
                    res.append(value)
            return res

        @staticmethod
        def __image_array_to_values_metode2(np_array):
            res = []
            reverse_row = False
            for row in np_array:
                altered_row = reversed(row) if reverse_row else row
                reverse_row = not reverse_row
                for value in altered_row:
                    res.append(value)
            return res

        @staticmethod
        def __image_array_to_values_metode3(np_array):
            res = []
            increment_offset = 1
            x = 0
            y = 0

            res.append(np_array[y][x])

            for index in range(int(np_array.shape[0]/2)):
                for i in range(increment_offset, 0, -1):
                    if(y > 0):
                        y -= 1
                    x += 1
                    res.append(np_array[y][x])
                    #print("X-loop: " + str(y) + str(x))

                increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

                for i in range(increment_offset, 0, -1):
                    if(x > 0):
                        x -= 1
                    y += 1
```

```
                    res.append(np_array[y][x])
                    #print("Y-loop: " + str(y) + str(x))

                increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

            for index in range(int(np_array.shape[0]/2)):
                moved_over = False
                for i in range(increment_offset, 0, -1):
                    if(y > 0 and moved_over):
                        y -= 1
                    moved_over = True
                    x += 1
                    res.append(np_array[y][x])
                    #print("X-loop: " + str(y) + str(x))

                increment_offset -= 1

                moved_over = False
                for i in range(increment_offset, 0, -1):
                    if(x > 0 and moved_over):
                        x -= 1
                    moved_over = True
                    y += 1
                    res.append(np_array[y][x])
                    #print("Y-loop: " + str(y) + str(x))

                increment_offset -= 1

            return res

        image_array_to_values_methods = {
            '1':__image_array_to_values_metode1.__func__,
            '2':__image_array_to_values_metode2.__func__,
            '3':__image_array_to_values_metode3.__func__
        }

        @classmethod
        def convert(cls, np_array, method):
            return cls.image_array_to_values_methods[str(method)](np_array)

    class ValuesToImageArray():

        def __init__(self):
            print("Class not supposed to be instantiated!")
            return None

        # Gjør om en 1d array til en 2d array med pikselverdier. 3 ulike metoder.
        @staticmethod
        def __values_to_image_array_metode1(value_array):
            np_array = [ [] for i in range(8) ]
            for i in range(8):
                for j in range(8):
                    np_array[i].append(value_array[j + (i * 8)])

            return np.array(np_array, dtype=np.uint8)

        @staticmethod
        def __values_to_image_array_metode2(value_array):
            np_array = [ [] for i in range(8) ]
            reverse_row = False
            for i in range(8):
                for j in range(8):
                    row_offset = j if not reverse_row else 7-j
                    np_array[i].append(value_array[(i*8) + row_offset])
                reverse_row = not reverse_row
            return np.array(np_array, dtype=np.uint8)

        @staticmethod
        def __values_to_image_array_metode3(value_array):
            np_array = np.indices((8,8))[0]
            increment_offset = 1
            x = 0
            y = 0

            reversed_value_array = list(reversed(value_array))

            np_array[y][x] = reversed_value_array.pop()

            for index in range(int(np_array.shape[0]/2)):
                for i in range(increment_offset, 0, -1):
                    if(y > 0):
                        y -= 1
                    x += 1
                    np_array[y][x] = reversed_value_array.pop()
                    #print("X-loop: " + str(y) + str(x))

                increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

                for i in range(increment_offset, 0, -1):
                    if(x > 0):
                        x -= 1
                    y += 1
```

```
                    np_array[y][x] = reversed_value_array.pop()
                    #print("Y-loop: " + str(y) + str(x))

                increment_offset += (1 if increment_offset < np_array.shape[0]-1 else 0)

            for index in range(int(np_array.shape[0]/2)):
                moved_over = False
                for i in range(increment_offset, 0, -1):
                    if(y > 0 and moved_over):
                        y -= 1
                    moved_over = True
                    x += 1
                    np_array[y][x] = reversed_value_array.pop()
                    #print("X-loop: " + str(y) + str(x))

                increment_offset -= 1

                moved_over = False
                for i in range(increment_offset, 0, -1):
                    if(x > 0 and moved_over):
                        x -= 1
                    moved_over = True
                    y += 1
                    np_array[y][x] = reversed_value_array.pop()
                    #print("Y-loop: " + str(y) + str(x))

                increment_offset -= 1

            return np.array(np_array, dtype=np.uint8)

        values_to_image_array_methods = {
            '1':__values_to_image_array_metode1.__func__,
            '2':__values_to_image_array_metode2.__func__,
            '3':__values_to_image_array_metode3.__func__
        }

        @classmethod
        def convert(cls, valuearray, method):
            return cls.values_to_image_array_methods[str(method)](valuearray)

    # En funksjon som brukes testing av et lite bilde
    def test_main():
        image_name = IMAGE_NAME.split('.')[0]

        np_array = None
        with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
            np_array = image_to_image_array(big_image)

        array_eight_by_eights = array_into_eight_by_eight(np_array)
        print(assembly_array_of_eight_by_eights(array_eight_by_eights))

    def generate_piecewise():
        image_name = IMAGE_NAME.split('.')[0]

        np_array = None
        with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
            np_array = image_to_image_array(big_image)

        array_eight_by_eights = array_into_eight_by_eight(np_array)

        selected_eight_by_eight = array_eight_by_eights[BLOCK_INDEXES[0]][BLOCK_INDEXES[1]]
        print("Kjoerer metode " + str(METHODE) + " paa bilde " + pf.format(IMAGE_NAME, pf.BOLD) +
        " blokk " + pf.format(str(BLOCK_INDEXES),pf.BOLD) + " med " + pf.format(str(PERIOD), pf.BOLD) + " som periode")
        print(array_to_piecewise(ImageArrayToValues.convert(selected_eight_by_eight, METHODE), PERIOD))

    def generate_image_from_psi():
        image_name = IMAGE_NAME.split('.')[0]

        print("Laster bilde: " + pf.format(IMAGE_NAME, pf.BOLD))
        np_array = None
        with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
            np_array = image_to_image_array(big_image)

        print("Skriver bildedata av hele bildet til json-filer.")
        write_array_to_datafile(os.path.join(IMAGE_DIR, image_name + ".json"), np_array)

        print("Genererer 8x8 blokker av det lastede bildet.")
        array_eight_by_eights = array_into_eight_by_eight(np_array)

        methode_dir = os.path.join(ANALYSERTE_BLOKKER_DIR, image_name, "metode" + str(METHODE))
        image_org_dir = check_directory(os.path.join(methode_dir, "bilder", "org/"))
        image_fourier_dir = check_directory(os.path.join(methode_dir, "bilder", "fourier/"))
        data_org_dir = check_directory(os.path.join(methode_dir, "data", "org/"))
        data_fourier_dir = check_directory(os.path.join(methode_dir, "data", "fourier/"))
        data_delta_dir = check_directory(os.path.join(methode_dir, "data", "delta/"))

        print("Velger 8x8-blokken som skal analyseres: " + pf.format(str(BLOCK_INDEXES), pf.BOLD))
        print("Lager bilde av 8x8-blokken som skal analyseres før den endres.")
        org_eight_by_eight = array_eight_by_eights[BLOCK_INDEXES[0]][BLOCK_INDEXES[1]]
        image_array_to_image_scale(org_eight_by_eight, os.path.join(image_org_dir, str(BLOCK_INDEXES) + ".png"), SCALE_FACTOR)

        print("Skriver data av 8x8-blokken som skal analyseres før den endres.")
```

```
        write_array_to_datafile(os.path.join(data_org_dir, str(BLOCK_INDEXES) + ".json"), org_eight_by_eight)

        print(pf.format("Lager en ny 8x8 fra psi(t) funksjonen.", pf.BOLD))
        fourier_eight_by_eight = change_to_fourierseriesvalues(org_eight_by_eight, METHODE).astype(np.dtype(np.uint8))

        print("Lager bilde av 8x8-blokken som er generert fra psi(t).")
        image_array_to_image(fourier_eight_by_eight, os.path.join(image_fourier_dir, str(BLOCK_INDEXES) + ".png"))

        print("Skriver data av 8x8-blokken som er generert fra psi(t).")
        write_array_to_datafile(os.path.join(data_fourier_dir, str(BLOCK_INDEXES) + ".json"), fourier_eight_by_eight)

        print("Skriver data av 8x8-blokken som er forskjellen mellom pikselverdienen fra orginal og fourier")
        write_array_to_datafile(os.path.join(data_delta_dir, str(BLOCK_INDEXES) + ".json"),
        org_eight_by_eight.astype(np.dtype(np.int8)) - fourier_eight_by_eight.astype(np.dtype(np.int8)))

        print(pf.format("Fullført! Sjekk \"fourier_bilder\" mappen for oppdateringer!", pf.BOLD))

    def transform_full_image():
        image_name = IMAGE_NAME.split('.')[0]

        np_array = None
        with Image.open(os.path.join(IMAGE_DIR, IMAGE_NAME)) as big_image:
            np_array = image_to_image_array(big_image)

        temp_dir = check_directory(os.path.join(IMAGE_DIR, "tmp/"))

        array_eight_by_eights = array_into_eight_by_eight(np_array)
        max_progress = (len(array_eight_by_eights) - 1) * (len(array_eight_by_eights[0]) - 1)

        string_of_piecewise = ""

        piecewise_to_psi_script_path = os.path.join(temp_dir, "piecewise_to_psi.mpl")
        psi_output_path = os.path.join(temp_dir, "psi_output.txt")
        with open(piecewise_to_psi_script_path, 'r') as script_file:
            piecewise_script_lines = script_file.readlines()

        for y_index, row in enumerate(array_eight_by_eights):
            for x_index, eight_by_eight in enumerate(row):
                string_of_piecewise = array_to_piecewise_cli(ImageArrayToValues.convert(eight_by_eight, METHODE), PERIOD)

                with open(piecewise_to_psi_script_path, "w") as piecewise_to_psi_file:
                    piecewise_script_lines[0] = string_of_piecewise + ": T:=" + str(2*PERIOD) + ": N:=" + str(AMOUNT_OF_TERMS) + ":\r
                    piecewise_to_psi_file.writelines(piecewise_script_lines)

                #with open(psi_output_path, 'w') as psi_output_file:
                maple_results = subprocess.Popen(['maple', '-q', piecewise_to_psi_script_path], stdout=subprocess.PIPE, shell=False)

                psi_string = maple_results.stdout.readline().decode('utf-8')

                #with open(psi_output_path, 'r') as psi_output_file:
                #    psi_string = psi_output_file.readlines()[0]

                psi_string = psi_string.replace("cg = ", "")

                array_eight_by_eights[y_index][x_index] = change_to_fourierseriesvalues_psi_string(eight_by_eight, METHODE, psi_strir


                current_progress = (y_index*len(array_eight_by_eights) + x_index)
                print(str(current_progress) + " av " + str(max_progress))


        generated_image_dir = check_directory(os.path.join(IMAGE_DIR, image_name, "metode" + str(METHODE)))
        image_array_to_image(assembly_array_of_eight_by_eights(array_eight_by_eights), os.path.join(generated_image_dir, "fourier.png

    def run():
        if(FULL_ANALYZE):
            transform_full_image():
        else:
            if(GENERATE_PIECEWISE_BOOL):
                generate_piecewise()
            else:
                generate_image_from_psi()

    def main():
        run()

    if __name__ == '__main__':
        main()
```