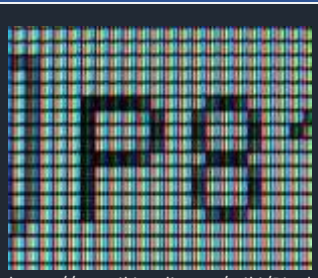
Steganography

จอภาพคอมพิวเตอร์สามารถแสดงสีของแต่ละจุดภาพได้ถึง 16 ล้านสี ถ้าสีของจุดภาพเพี้ยนไปเล็กน้อย ตามนุษย์ทั่วไปก็แยกไม่เห็นความแตกต่าง เราจะอาศัยข้อเท็จจริงนี้ ในการ "ยืม" ที่เก็บข้อมูลของแต่ละจุดภาพมาแฝงข้อมูลลับลงไปในแฟ้มภาพ เมื่อเปิดแฟ้มภาพ เราจะมองไม่ออกว่า มีการปนข้อมูลในภาพ แต่สามารถดึงข้อมูลลับนี้กลับคืนมาได้ การบ้านนี้ให้เขียนฟังก์ชัน **`embed_text_to_image`** เพื่อแฝงข้อความ (ภาษาอังกฤษ) เข้าในแฟ้มภาพ และฟังก์ชัน **`get_embedded_text_from_image`** เพื่อดึงข้อความที่แฝงในแฟ้มภาพคืนมา ดังตัวอย่างข้างล่างนี้



https://en.wikipedia.org/wiki/Pixel

cat.png

cat_x.png

"Meet me at 11pm"

```
txt = "Meet me at 11pm"
img_in  = 'cat.png'
img_out = 'cat_x.png'
embed_text_to_image(txt, img_in, img_out)
```

"Meet me at 11pm"

```
img_file = 'cat_x.png'
txt = get_embedded_text_from_file( img_file )
print(txt)
```

สิ่งที่ต้องรู้ก่อนคือ ข้อมูลที่ประกอบกันเป็นภาพ มีอะไรบ้าง ภาพดิจิตอลแบบ raster ที่ใช้กันทั่วไป (และที่ใช้ในโจทย์นี้) ประกอบด้วยจุดภาพ (pixel) เรียงเป็นแถว หลาย ๆ แถว แต่ละแถวมีหลาย ๆ จุด เช่น ภาพขนาด 640 × 480 จุด หมายถึง ภาพที่มี 480 แถว แต่ละแถวมีจุดภาพ 640 จุด สิ่งที่ต้องรู้ละเอียดอีกนิดคือ จุดภาพแต่ละจุดประกอบด้วยจำนวนเต็ม 3 จำนวนที่มีค่าแทนความเข้มของแม่สี แดง (R) เขียว (G) และน้ำเงิน (B) โดยทั่วไปให้แต่ละแม่สีมีค่าตั้งแต่ 0 ถึง 255 คือให้แต่ละแม่สีมีความเข้มได้ 256 ระดับ (0 แทนการไม่มีองค์ประกอบของแม่สีนั้นเลย) การมีแม่สี 3 สี แต่ละสีมี 256 ระดับ จึงแทนสีได้ทั้งหมด 256^3 = 16,777,216 สี โดยสรุป ภาพขนาด 640 × 480 ต้องเก็บจำนวนเต็มทั้งสิ้น 640 × 480 × 3 ตัว (หมายเหตุ: แต่ละจุดภาพอาจมีองค์ประกอบอื่น เช่น ระดับความโปร่งแสงของจุดภาพ ซึ่งขอไม่ใช้ในโจทย์นี้)

เพื่อให้ง่ายสุด ๆ โปรแกรมที่จะเขียนในโจทย์นี้ เก็บภาพในโปรแกรมด้วย nested list ซ้อนกันสามระดับ คือ ถ้า **`img`** เป็นลิสต์ที่เก็บภาพ จะได้ว่า

- **`img[r]`** เก็บข้อมูลของจุดภาพต่าง ๆ ในแถวที่ `r` ของภาพ
- **`img[r][c]`** เก็บข้อมูลของจุดภาพ `1` จุด ณ แถวที่ `r` คอลัมน์ที่ `c` ของภาพ
- **`len( img )`** คือจำนวนแถว และ **`len( img[0] )`** คือจำนวนจุดในแต่ละแถว
- **`len( img[0][0] )`** คือจำนวนแม่สีของแต่ละจุด (ซึ่งมีค่าเท่ากับ 3 เพราะมีแม่สีสามสี) โดยที่
  - **`img[r][c][0]`** เก็บระดับความเข้มของแม่สีแดง
  - **`img[r][c][1]`** เก็บระดับความเข้มของแม่สีเขียว
  - **`img[r][c][2]`** เก็บระดับความเข้มของแม่สีน้ำเงิน

เช่น ภาพขนาด 4 × 3 คือ ภาพที่มี 3 แถว แถวละ 4 จุด (แต่ละจุดมีแม่สี 3 สี) ดังรูปทางซ้าย แทนด้วยลิสต์ซ้อนลิสต์ในรูปทางขวาข้างล่างนี้

20	206	22	32	34	36	85	87	89	103	15	17
21	210	21	31	31	36	81	81	81	110	11	15
22	220	22	32	32	32	82	82	82	120	12	12

```
[
  [ [20,206,22], [32,34,36], [85,87,89], [103,15,17] ],
  [ [21,210,21], [31,31,36], [81,81,81], [110,11,15] ],
  [ [22,220,22], [32,32,32], [82,82,82], [120,12,12] ],
]
```

วิธีแฝงข้อมูลในภาพที่ใช้ในโจทย์นี้ คือ แปลงข้อมูลที่ต้องการแฝงเป็นเลขฐานสองเสียก่อน จากนั้นนำแต่ละบิตไปแฝงตามค่าของแม่สีต่าง ๆ ในภาพ แม่สีละ 1 บิต (ดังนั้นหนึ่งจุดภาพก็แฝงได้ 3 บิต) แฝงไล่ไปเรื่อยจากซ้ายไปขวาทีละแถว ๆ จากบนลงล่าง โดยใช้กฎการแฝงตามตารางทางขวานี้ C คือค่าเดิมของแม่สี, b คือค่าของบิตข้อมูลที่ต้องการแฝง และ C^* คือค่าใหม่เมื่อแฝง b ใน C แล้ว เช่น ต้องการแฝง 0 ในค่า 20 ก็ได้เป็น 20 แต่ถ้าแฝง 0 ในค่า 23 จะได้ 22 เมื่อวิธีแฝงเป็นแบบนี้ การดึงข้อมูลที่แฝงกลับมาจะง่ายมาก คือ ถ้า C^* เป็นจำนวนคู่ แสดงว่า มี 0 แฝงอยู่ แต่ถ้าเป็นจำนวนคี่ ก็มี 1 แฝงอยู่

ตารางกฎการแฝงบิตในแม่สี

C	b	C^*	
จำนวนคู่	0	C	จำนวนคู่
จำนวนคู่	1	$C+1$	จำนวนคี่
จำนวนคี่	0	$C-1$	จำนวนคู่
จำนวนคี่	1	C	จำนวนคี่

ตัวอย่าง ถ้าแฝงข้อมูลฐานสอง 0001110001101 เข้าในภาพที่แทนด้วยลิสต์ทางซ้ายข้างล่างนี้ จะได้ลิสต์ใหม่ทางขวา

```
[     0   0   0     1   1   1     0   0   0     1   1   1
  [ [20,206,22], [32,34,36], [85,87,89], [103,15,17] ],
      0   1
  [ [21,210,21], [31,31,36], [81,81,81], [110,11,15] ],

  [ [22,220,22], [32,32,32], [82,82,82], [120,12,12] ],
]
```

```
[
  [ [20,206,22], [33,35,37], [84,86,88], [103,15,17] ],

  [ [20,211,21], [31,31,36], [81,81,81], [110,11,15] ],

  [ [22,220,22], [32,32,32], [82,82,82], [120,12,12] ],
]
```

คำถามที่ตามมาคือ

- จะรู้ได้อย่างไรว่า ภาพที่สนใจมีข้อความซ่อนอยู่หรือเปล่า และ
- ข้อความที่ซ่อนมีอักขระกี่ตัว

เพื่อแก้ปัญหาทั้งสองข้างบนนี้ ข้อมูลทั้งหมดที่ถูกแฝงในภาพ ประกอบด้วย

ลำดับบิตพิเศษ	ลำดับบิตที่แทนจำนวนตัวอักขระที่ถูกแฝง	ลำดับบิตที่แทนอักขระต่าง ๆ ที่ถูกแฝงในภาพ	ลำดับบิตพิเศษ

- **ลำดับบิตพิเศษ** เป็นลำดับบิตจำนวน 16 บิตที่มีค่าอะไรบางอย่าง ที่เรากำหนดไว้ล่วงหน้า ภาพใดที่มีข้อมูลแฝงเริ่มด้วยค่านี้ และจบด้วยค่านี้ ก็ถือว่าจะมีข้อมูลแฝง สำหรับโจทย์นี้ ให้ใช้ลำดับบิต 0100111101001011 เสมอ (ไม่มีเหตุผลใด ๆ ที่เลือกค่านี้ ค่าอะไรก็ได้ซึ่งฟังก์ชันแฝงข้อความกับฟังก์ชันถอดข้อความ ตกลงกันไว้ก่อนว่า จะใช้เหมือนกัน)
- **ลำดับบิตที่แทนจำนวนอักขระที่ถูกแฝง** เป็นลำดับบิตที่เมื่อมองเป็นจำนวนฐานสองก็คือจำนวนเต็มที่แทนจำนวนอักขระที่ถูกแฝงในภาพ กำหนดให้ลำดับบิตนี้มีขนาด 16 บิต สามารถแทนค่าได้ตั้งแต่ 0 ถึง $2^{16} - 1$ แปลว่า แฝงได้ไม่เกิน $2^{16} - 1 = 65{,}535$ ตัว (โดยภาพที่ใช้ก็ต้องมีจุดภาพเพียงพอที่จะแฝงข้อความขนาดนั้นด้วย) เช่น ถ้าต้องการแฝง ABBA ซึ่งมี 4 ตัว ลำดับบิตนี้ก็เป็น 0000000000000100 เพราะ 100 ในระบบเลขฐานสองคือ 4 ในระบบเลขฐานสิบ
- **ลำดับบิตที่แทนอักขระต่าง ๆ ที่ถูกแฝงในภาพ** อักขระภาษาอังกฤษ 1 ตัวใช้เลขฐานสองขนาด 8 บิต เช่น รหัสของ A คือ 01000001, ของ B คือ 01000010 ดังนั้น ถ้าต้องการแฝง ABBA ลำดับบิตก็คือ 01000001010000100100001001000001 ตารางข้างล่างนี้แสดงลำดับบิตของอักขระแต่ละตัว ซึ่งสามารถใช้ฟังก์ชัน `char_to_bits` ที่ได้เขียนให้แล้ว (และจะอธิบายต่อไป) เช่น `char_to_bits("A")` ได้ผลเป็นสตริง "01000001"

ตารางลำดับบิตของตัวอักขระต่าง ๆ

00100000	! 00100001	" 00100010	# 00100011	$ 00100100	% 00100101	& 00100110	' 00100111
(00101000	) 00101001	* 00101010	+ 00101011	, 00101100	- 00101101	. 00101110	/ 00101111
0 00110000	1 00110001	2 00110010	3 00110011	4 00110100	5 00110101	6 00110110	7 00110111
8 00111000	9 00111001	: 00111010	; 00111011	< 00111100	= 00111101	> 00111110	? 00111111
@ 01000000	A 01000001	B 01000010	C 01000011	D 01000100	E 01000101	F 01000110	G 01000111
H 01001000	I 01001001	J 01001010	K 01001011	L 01001100	M 01001101	N 01001110	O 01001111
P 01010000	Q 01010001	R 01010010	S 01010011	T 01010100	U 01010101	V 01010110	W 01010111
X 01011000	Y 01011001	Z 01011010	[01011011	\ 01011100	] 01011101	^ 01011110	_ 01011111
` 01100000	a 01100001	b 01100010	c 01100011	d 01100100	e 01100101	f 01100110	g 01100111
h 01101000	i 01101001	j 01101010	k 01101011	l 01101100	m 01101101	n 01101110	o 01101111
p 01110000	q 01110001	r 01110010	s 01110011	t 01110100	u 01110101	v 01110110	w 01110111
x 01111000	y 01111001	z 01111010	{ 01111011	| 01111100	} 01111101	~ 01111110	• 01111111

ตัวอย่าง: หากต้องการแฝง **ABBA** ลำดับบิตทั้งหมดที่ต้องนำไปแฝง คือ

0100111101001011 0000000000000100 01000001 01000010 01000010 01000001 0100111101001011

แทนเลข 4 คือ `len('ABBA')` A B B A

ลำดับบิตพิเศษ	ลำดับบิตที่แทนจำนวนอักขระ	ลำดับบิตของอักขระต่าง ๆ	ลำดับบิตพิเศษ
16 บิต	16 บิต	8×4 บิต	16 บิต

ดังนั้น ถ้าต้องการแฝงอักขระ N ตัว ต้องใช้บิตทั้งสิ้น $16 + 16 + 8N + 16$ บิต ภาพที่ใช้จึงต้องมีจำนวนจุดภาพไม่น้อยกว่า $\left\lceil 16 + \frac{8N}{3} \right\rceil$ จุด

โปรแกรมต้นฉบับ

```
# Prog-10: Steganography
# 6?3?????21 Name ?
```

ใส่เลขประจำตัว ชื่อ นามสกุล

```
import math
import copy
import numpy
from PIL import Image

# ----------------------------------------
def load_image(filename):
    # return a nested list of an image file
    ...

def save_image(img, filename):
    # save img to a file
    ...

def show_image(filename):
    # show image in filename
    ...

def clone_image(img):
    # return the exact copy of img
    ...

def char_to_bits(ch):
    # return the 8-bit string of ch
    ...

def bits_to_char( bits ):
    # return the char of 8-bit-string bits
    ...

def int_to_bits(n):
    # return the 16-bit string of int n
    ...

def bits_to_int( bits ):
    # return the int of bit-string bits
    ...

def main():
    # main program
    ...

# ------------------
```

เขียนฟังก์ชันอื่นที่ต้องการได้ในบริเวณนี้

```
# ----------------------------------------------
def embed_text_to_image(text, file_in, file_out):

# ----------------------------------------------
def get_embedded_text_from_image(file_in):

# ----------------------------------------------
SPECIAL_BITS = '0100111101001011'
main()
```

โปรแกรมที่ส่ง ห้ามเปลี่ยนอะไรใด ๆ ในส่วนที่มีพื้นหลังเป็นสีแดง และที่เป็นตัวสีแดงโดยเด็ดขาด เปลี่ยนได้เฉพาะส่วนที่มีพื้นหลังเป็นสีเขียวเท่านั้น

ฟังก์ชันที่มีให้

ฟังก์ชัน	ตัวอย่างการใช้งาน
load_image(filename) รับ filename เป็นสตริงเก็บชื่อแฟ้ม (โจทย์นี้รองรับเฉพาะแบบ PNG เท่านั้น) คืน ลิสต์ที่เก็บภาพที่อ่านจากแฟ้ม เป็นลิสต์ซ้อนกันสามชั้น ตามที่อธิบายก่อนนี้ ฟังก์ชันนี้อ่านข้อมูลจากแฟ้มภาพ filename มาเก็บในลิสต์ในรูปแบบที่อธิบายก่อนนี้	`fi = 'c:\\temp\\cat.png'` `img = load_image( fi )`

save_image(img, filename)
รับ img เป็นลิสต์ที่เก็บภาพที่อ่านจากแฟ้ม เป็นลิสต์ซ้อนกันสามชั้น ตามที่อธิบายก่อนนี้
filename เป็นสตริงเก็บชื่อแฟ้ม (โจทย์นี้รองรับเฉพาะแบบ PNG เท่านั้น)
ไม่คืนอะไร
 	
ฟังก์ชันนี้นำข้อมูลภาพในลิสต์ img ไปบันทึกในแฟ้มภาพ filename
	`img = [ [[255,0,0], [255,0,0]],`
	`[[255,0,0], [255,0,0]],`
	`[[255,0,0], [255,0,0]] ]`
	
	`save_image(img, 'c:\\temp\\x.png')`
	
	# img เก็บภาพขนาด **3** แถว แถวละ **2** จุด แต่ละจุดเป็นสีแดงสด

show_image(filename) รับ filename เป็นสตริงเก็บชื่อแฟ้ม (โจทย์นี้รองรับเฉพาะแบบ PNG เท่านั้น) ไม่คืนอะไร ฟังก์ชันนี้อ่านแฟ้มภาพ filename มาแสดงทางจอภาพ	`fi = 'c:\\temp\\cat.png'` `show_image( fi )`

clone_image(img)
รับ img เป็นลิสต์ที่เก็บภาพ
คืน ลิสต์ใหม่ที่มีข้อมูลและโครงสร้างเหมือน img ทุกประการ
 	
ฟังก์ชันนี้มีไว้สร้างสำเนาของลิสต์ img
	`fi = 'c:\\temp\\cat.png'`
	`img0 = load_image( fi )`
	`img1 = clone_image( img0 )`
	`img1[0][0] = [255,255,0] # to yellow`

char_to_bits(ch)
รับ ch เป็นสตริงเก็บอักขระ 1 ตัว
คืน สตริงที่เก็บลำดับของ '1' กับ '0' ขนาด 8 หลัก ที่แทนรหัสของอักขระใน ch
 	
ฟังก์ชันนี้มีไว้แปลงอักขระ 1 ตัวไปเป็นสตริงเลขฐานสองขนาด 8 หลัก
	`ch = 'A'`
	`bits = char_to_bits( ch )`
	`print( bits == '01000001' )`
	# print ข้างบนนี้ต้องได้ True

bits_to_char(bits) รับ bits เป็นสตริงที่เก็บลำดับของ '1' กับ '0' ขนาด 8 หลัก คืน สตริงที่มีอักขระหนึ่งตัว โดยอักขระตัวนี้มีรหัสเลขฐานสองตามที่เก็บใน bits ฟังก์ชันนี้มีไว้แปลงสตริงเลขฐานสองขนาด 8 หลัก ไปเป็นอักขระ 1 ตัว	**bits = '01000001'** **ch = bits_to_char(bits)** **print(ch == 'A')** # print ข้างบนนี้ต้องได้ True
int_to_bits(n) รับ n เป็นจำนวนเต็ม คืน สตริงเก็บลำดับของ '1' กับ '0' ขนาด 16 หลักที่แทนเลขฐานสองของจำนวนเต็มใน n ฟังก์ชันนี้มีไว้แปลงจำนวนเต็ม ไปเป็นสตริงเลขฐานสองขนาด 16 หลัก	**n = 30** **bits = int_to_bits(n)** **print(bits == '0000000000011110')** # print ข้างบนนี้ต้องได้ True
bits_to_int(bits) รับ bits สตริงที่เก็บลำดับของ '1' กับ '0' ขนาด 16 หลัก คืน จำนวนเต็มที่มีเลขฐานสองตามค่าที่เก็บใน bits ฟังก์ชันนี้มีไว้แปลงสตริงเลขฐานสองขนาด 16 หลัก ไปเป็นจำนวนเต็ม	**bits = '0000000000011110'** **n = bits_to_int(bits)** **print(n == 30)** # print ข้างบนนี้ต้องได้ True

ฟังก์ชันที่ต้องเขียน

def embed_text_to_image(text, file_in, file_out)

- **text** เป็นสตริงเก็บข้อความ (ภาษาอังกฤษและสัญลักษณ์ต่าง ๆ ที่แสดงในตารางหน้าที่ 2)
- **file_in** เป็นสตริงเก็บชื่อแฟ้มภาพต้นฉบับ (แฟ้มนี้มีอยู่แล้วแน่ ๆ)
- **file_out** เป็นสตริงเก็บชื่อแฟ้มภาพผลลัพธ์ที่ได้จากการนำ **text** มาแฝงในภาพที่อ่านมาจากแฟ้มภาพ **file_in**
  ขอเน้นว่า ไม่มีการเปลี่ยนข้อมูลใด ๆ ในแฟ้ม **file_in**
- คืน **True** ถ้าทำงานได้สำเร็จ
  False ถ้าสร้างแฟ้มภาพผลลัพธ์ไม่ได้ เพราะภาพจากแฟ้ม **file_in** มีเนื้อที่ไม่พอให้แฝงข้อมูลที่ต้องการ
- ผลที่ได้: แฟ้มภาพ **file_out** ที่ได้จากการแฝงข้อความ **text** ในภาพจากแฟ้มภาพ **file_in** ในรูปแบบที่ได้นำเสนอ

def get_embedded_text_from_image(file_in)

- **file_in** เป็นสตริงเก็บชื่อแฟ้มภาพที่ต้องการดึงข้อความที่แฝงไว้
- ถ้าสิ่งที่อ่านได้จากแฟ้มมีรูปแบบที่ถูกต้อง จะคืนข้อความที่แฝงไว้ในแฟ้มภาพ **file_in**
- ถ้ามีรูปแบบไม่ถูกต้อง จะคืนสตริงว่าง ๆ ("" ซึ่งคือสตริงที่ความยาวเป็น 0)

ลำดับบิตพิเศษ	ลำดับบิตที่แทนจำนวนอักขระที่ถูกแฝง	ลำดับบิตที่แทนอักขระต่าง ๆ ที่ถูกแฝงในภาพ	ลำดับบิตพิเศษ

ข้อแนะนำ

- ลองเขียนฟังก์ชัน **get_embedded_text_from_image** ก่อน แล้วทดลองดึงข้อความแฝงจากแฟ้มต่อไปนี้
  - **5x5_A.png** แฟ้มนี้มีตัว **A** แฝงอยู่
  - **5x5_AB.png** แฟ้มนี้มี **AB** แฝงอยู่
  - **5x5_ABC.png** แฟ้มนี้มี **ABC** แฝงอยู่
  - **5x5_none0.png, 5x5_none1.png, 5x5_none2.png** ทั้งสามแฟ้มไม่มีข้อความแฝง
  - **tux_hidden_text.png** แฟ้มนี้มีข้อความยาว ๆ แฝงอยู่ (ขอไม่บอก ถ้าเห็นก็จะรู้ว่าถูกหรือไม่)
  - แฟ้มที่ขึ้นต้นด้วย **5x5** มีขนาด 5×5 pixels มีขนาดเล็ก แสดงข้อมูลในลิสต์ออกมาดูได้ง่าย เหมาะกับการหาที่ผิดของโปรแกรม
- เขียนฟังก์ชัน **embed_text_to_image** ที่สามารถใช้ฟังก์ชัน **get_embedded_text_from_image** ร่วมทดสอบความถูกต้องของ **embed_text_to_image**

ตัวอย่าง เมื่ออ่านแฟ้ม **5x5_AB.png** มาเก็บเป็นลิสต์ จะได้ลิสต์ซ้อนกันสามชั้น ข้างล่างนี้

```
[
 [[254, 255, 254], [196, 195, 191], [105, 101,  86], [165, 162, 154], [255, 254, 255]],
 [[177, 176, 170], [120, 118, 108], [200, 194, 184], [134, 128, 116], [178, 178, 172]],
 [[115, 112, 100], [203, 200, 192], [200, 198, 192], [213, 206, 199], [110, 106,  96]],
 [[146, 147, 136], [190, 189, 182], [234, 231, 225], [181, 177, 170], [153, 148, 138]],
 [[255, 254, 255], [183, 181, 168], [159, 155, 143], [183, 181, 169], [255, 255, 255]]
]
```

ถ้าคิดว่ามีการแฝงข้อความในรูป ก็ดูแต่ละค่าของแม่สี เพื่อดึงบิตที่ถูกแฝง แม่สีละหนึ่งบิต ออกมาตามตารางกฎการแฝงในหน้าที่ 1 ดังแสดงข้างล่างนี้เป็นเลข 0 กับ 1 ที่ด้านบนของค่าแม่สี เริ่มที่ 16 บิตแรก (พื้นเขียว) คือ **0100111101001011** ซึ่งตรงกับลำดับบิตพิเศษ จากนั้นดู 16 บิตถัดไปที่เป็นลำดับบิตแทนจำนวนอักขระ (พื้นบานเย็น) คือ **0000000000000010** มีค่าเท่ากับ 2 แสดงว่า ข้อความแฝงมีอักขระ 2 ตัว ก็อ่าน 8×2 = 16 บิตถัดไป (พื้นฟ้า อักขระละ 8 บิต) คือ **0100000101000010** ซึ่งคือรหัสของ **AB** และสุดท้ายดูอีก 16 บิตว่าเป็นลำดับพิเศษหรือไม่ ถ้าใช่ ทุกอย่างก็ถูกต้อง ซึ่งในตัวอย่างนี้ก็เป็นลำดับพิเศษตามที่คาดไว้

```
[
     0   1   0     0   1   1     1   1   0     1   0   0     1   0   1
 [[254, 255, 254], [196, 195, 191], [105, 101,  86], [165, 162, 154], [255, 254, 255]],

     1   0   0     0   0   0     0   0   0     0   0   0     0   0   0
 [[177, 176, 170], [120, 118, 108], [200, 194, 184], [134, 128, 116], [178, 178, 172]],

     1   0   0     1   0   0     0   0   0     1   0   1     0   0   0
 [[115, 112, 100], [203, 200, 192], [200, 198, 192], [213, 206, 199], [110, 106,  96]],

     0   1   0     0   1   0     0   1   1     1   1   0     1   0   0
 [[146, 147, 136], [190, 189, 182], [234, 231, 225], [181, 177, 170], [153, 148, 138]],

     1   0   1     1
 [[255, 254, 255], [183, 181, 168], [159, 155, 143], [183, 181, 169], [255, 255, 255]]
]
```

อย่าลืมใช้ฟังก์ชันที่เขียนไว้แล้วให้เป็นประโยชน์

- **load_image** อ่านแฟ้มภาพ
- **save_image** บันทึกแฟ้มภาพ
- **char_to_bits** แปลงตัวอักขระหนึ่งตัวเป็นสตริงลำดับบิตที่แทนรหัสของอักขระ
- **bits_to_char** แปลงสตริงลำดับบิตที่แทนรหัสของอักขระเป็นสตริงตัวอักขระ
- **int_to_bits** แปลงจำนวนเต็มเป็นสตริงลำดับบิต
- **bits_to_int** แปลงสตริงลำดับบิตเป็นจำนวนเต็ม
- และฟังก์ชันอื่น ๆ ที่เขียนไว้แล้ว