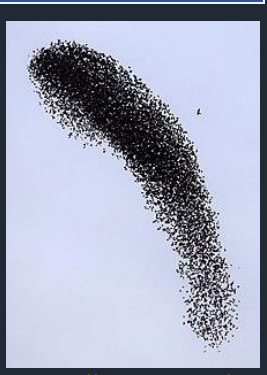
Vicsek Model

Vicsek model เป็นแบบจำลองคณิตศาสตร์ที่ใช้บรรยายพฤติกรรมการเคลื่อนที่ของฝูง เช่น ฝูงนกบิน (อาจเป็นฝูงผึ้ง ฝูงปลา ฝูงค้างคาว ฝูงชน หรืออื่น ๆ ก็ได้ แต่ขอใช้คำว่า นก ไปตลอดทั้งโจทย์) หลักการง่าย ๆ ของแบบจำลองนี้คือ นกแต่ละตัวจะบินในทิศทางที่กำหนดโดยค่าเฉลี่ยของทิศทางของนกตัวที่อยู่ใกล้ ๆ บวกกับทิศทางสะเปะสะปะตามใจตนเองอีกเล็กน้อย https://en.wikipedia.org/wiki/Vicsek_model

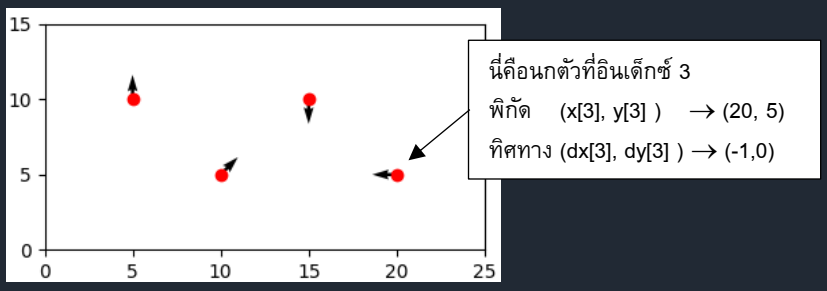

https://en.wikipedia.org/wiki/Flock_(birds)

การบ้านนี้เกี่ยวกับการคำนวณทิศทางดังกล่าว เพื่อให้ง่ายจะสนใจการเคลื่อนที่ของฝูงนกในระนาบสองมิติ ให้ถือว่า ทุกตัวบินด้วยความเร็วเท่ากันหมด ขอเก็บข้อมูลของนกต่าง ๆ ในลิสต์ `x`, `y`, `dx` และ `dy` โดยที่

- `x[i]`, `y[i]` แทนพิกัดคาร์ทีเซียนบนระนาบสองมิติของนกตัวที่ `i`
- `dx[i]`, `dy[i]` แทนทิศทางของนกตัวที่ `i` เคลื่อนที่ไป (`dx[i]`, `dy[i]`) เป็น unit vector คือ $(\texttt{dx[i]})^2 + (\texttt{dy[i]})^2$ เท่ากับ 1 (อาจแทนทิศทางด้วยมุมในช่วง $[0, 2\pi)$ ก็ได้ แต่การบ้านนี้เลือกแทนทิศทางด้วยเวกเตอร์คาร์ทีเซียน)

ตัวอย่าง: รูปข้างล่างนี้แสดงตำแหน่งและทิศทางของนก 4 ตัว (แสดงตำแหน่งด้วยจุดและทิศทางด้วยลูกศร) ของข้อมูล

$$\texttt{x} = [5, 10, 15, 20],\ \texttt{y} = [10, 5, 10, 5],\ \texttt{dx} = [0, \tfrac{1}{\sqrt{2}}, 0, -1],\ \texttt{dy} = [1, \tfrac{1}{\sqrt{2}}, -1, 0]$$



โปรแกรมของการบ้านนี้อยู่ในหน้าสุดท้าย เมื่อเขียน code ทุกอย่างครบถ้วน จะจำลองและแสดงการเคลื่อนที่ของฝูงนก โดยนิสิตต้องเขียนฟังก์ชัน `gen_data`, `move_all` และ `neighbor_average_direction` ตามข้อกำหนดดังต่อไปนี้ (ดู**ตัวอย่างภาพเคลื่อนไหวแสดงผลลัพธ์**ที่ได้หลังเขียนแต่ละฟังก์ชันที่ทำงานถูกต้อง) **จงเขียนสามฟังก์ชันตามข้อกำหนดที่อธิบายไว้ในตารางข้างล่างนี้**

ฟังก์ชันที่ต้องเขียน

`def gen_data(N, W, H):`

- **`N`** เป็นจำนวนเต็มเก็บจำนวนนก
- **`W`, `H`** เป็นจำนวนจริงเก็บความกว้างและความสูงของวินโดว์ที่แสดงกลุ่มนก
- คืน ลิสต์ 4 ตัว `x`, `y`, `dx` และ `dy` ที่เก็บข้อมูลตำแหน่งและทิศทางของนก `N` ตัวที่สุ่มสร้างขึ้นมา โดยตำแหน่งสุ่ม `x[i]` ของนก ต้องอยู่ในช่วง $[0, W)$, ตำแหน่งสุ่ม `y[i]` ของนก ต้องอยู่ในช่วง $[0, H)$ และทิศทางสุ่มของนก `dx[i]`, `dy[i]` ต้องเป็น *unit vector* โดยให้สุ่มมุมในช่วง $[0, 2\pi)$ ก่อนแล้วค่อยเปลี่ยนเป็น Cartesian vector

  หมายเหตุ: คำสั่ง `random.random()` จะคืนจำนวนสุ่มแบบ float ในช่วง $[0, 1)$

- ตัวอย่าง

```
x, y, dx, dy = gen_data(2, 15, 10)
print(x, y)
print(dx, dy)
```

อาจแสดง

```
[7.225278481091861, 1.2857053740918356] [9.442188477686212, 3.7718286414337294]
[-0.7515061360903788, 0.9147065870925196] [-0.6597261003011091, 0.404118620617207 94]
```

(อาจได้ผลอย่างอื่น เนื่องจากข้อมูลได้จากการสุ่ม แต่ต้องเป็นไปตามข้อกำหนดข้างต้น)

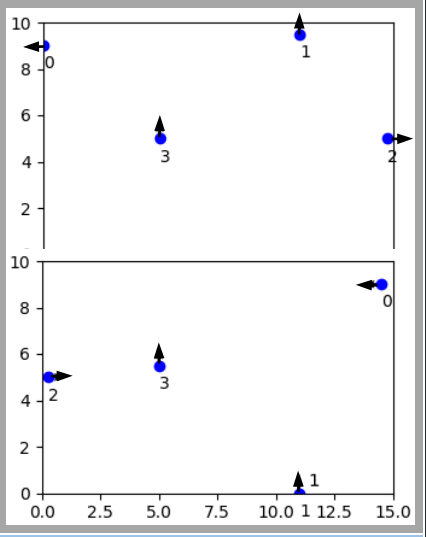
`def move_all(x, y, dx, dy, d, W, H):`

- **x, y** เป็นลิสต์ โดยที่ `x[i]`, `y[i]` คือพิกัด (x_i, y_i) ของนกตัวที่ `i`
- **dx, dy** เป็นลิสต์ โดยที่ `dx[i]`, `dy[i]` คือ unit vector ของทิศทางที่นกตัวที่ `i` กำลังพุ่งไป
- **d** เป็นจำนวนจริงแทนการกระจัด (displacement) ที่นกจะเปลี่ยนตำแหน่งไปในทิศทาง `dx`, `dy`
- **W, H** เป็นจำนวนจริงแทนความกว้างและความสูงของวินโดว์ที่แสดงกลุ่มนก
- ฟังก์ชันนี้ไม่คืนอะไร สิ่งที่ทำคือ ปรับพิกัดตำแหน่งของนกแต่ละตัวให้เปลี่ยนการกระจัดไป `d` ในทิศทางที่กำหนดโดย `dx`, `dy`
  ถ้าตำแหน่งใหม่อยู่นอกช่วงของวินโดว์ที่แสดงกลุ่มนก ให้ปรับตำแหน่ง "วน" กลับมาที่ตำแหน่งตรงข้าม เช่น ถ้า `W` = 10 แล้วพิกัด x หลังปรับมีค่าเป็น 10.2 ก็ให้เปลี่ยนเป็น 0.2 แต่ถ้าหลังปรับแล้วมีค่าเป็น –0.2 ก็ให้เปลี่ยนเป็น 9.8
  สำหรับพิกัด y ก็พิจารณาในทำนองเดียวกัน
- ตัวอย่าง

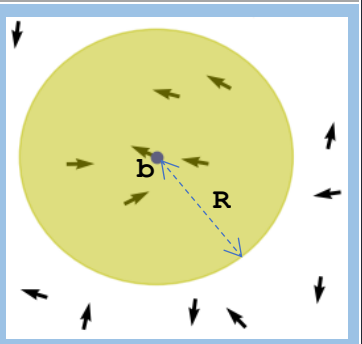
```
x = [0.0, 11.0, 14.75, 5.0]; y = [9.0, 9.5, 5.0, 5]
dx = [-1.0, 0.0, 1.0, 0.0]; dy = [0.0, 1.0, 0.0, 1.]
move_all(x, y, dx, dy, 0.5, 15, 10)
print(x, y)
print(dx, dy)
```

จะแสดง

```
[14.5, 11.0, 0.25, 5.0] [9.0, 0.0, 5.0, 5.5]
[-1.0, 0.0, 1.0, 0.0] [0.0, 1.0, 0.0, 1.0]
```



`def neighbor_average_direction(x, y, dx, dy, k, R):`



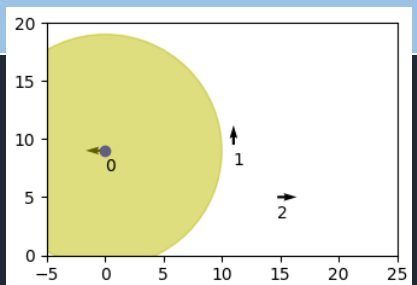
- **x, y** เป็นลิสต์ โดยที่ `x[i]`, `y[i]` คือพิกัด (x_i, y_i) ของนกตัวที่ `i`
- **dx, dy** เป็นลิสต์ โดยที่ `dx[i]`, `dy[i]` คือ unit vector ของทิศทางที่นกตัวที่ `i` กำลังพุ่งไป
- **k** เป็นจำนวนเต็ม ที่เป็นเลขอินเด็กซ์ของ `x`, `y`, `dx`, `dy` ระบุนกตัวที่สนใจ
- **R** เป็นจำนวนจริง
- คืน จำนวนจริง 2 ค่า แทน *unit vector* ซึ่งเป็นค่าเฉลี่ยของทิศทางการบินของนกต่าง ๆ ที่อยู่ห่างจากนกตัวที่ `k` ไม่เกิน `R` (รวมพิจารณานกตัวที่ `k` ด้วย)
- ตัวอย่าง

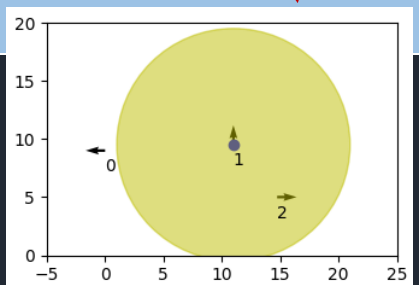
```
x = [0.0, 11.0, 14.75]; y = [9.0, 9.5, 5.0]
dx = [-1.0, 0.0, 1.0]; dy = [0.0, 1.0, 0.0]
mx, my = neighbor_average_direction(x, y, dx, dy, 0, 10)
print(mx, my)
mx, my = neighbor_average_direction(x, y, dx, dy, 1, 10)
print(mx, my)
mx, my = neighbor_average_direction(x, y, dx, dy, 1, 15)
print(mx, my)
```

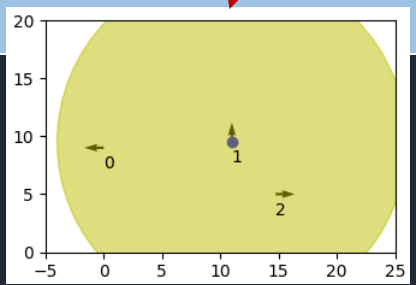
จะแสดง

```
-1.0 1.2246467991473532e-16
0.7071067811865476 0.7071067811865475
6.123233995736766e-17 1.0
```

ค่าที่คำนวณได้ อาจต่างจากที่แสดงในหลักที่มีนัยสำคัญน้อย ๆ (ด้วยเหตุที่ขั้นตอนการคำนวณอาจไม่เหมือนกัน)







โปรแกรมต้นฉบับ

```
import matplotlib.pyplot as plt
from matplotlib import animation, rc
import random
import math

def main():
    import sys
    # check if this code is running in colab
    in_colab = 'google.colab' in sys.modules

    random.seed(1111)
    W, H = 120, 100
    # 200 birds on a WxH window
    x,y,dx,dy = gen_data(200, W, H)

    fig = plt.figure(figsize=(4*W/H, 4))
    anim = animation.FuncAnimation(fig, animate,
                   fargs=(x, y, dx, dy, W, H),
                   frames=(60 if in_colab else None),
                   repeat=False, interval=50)
    if in_colab:
        rc('animation', html='jshtml')
        return anim
    else:
        plt.show()

def animate(n, x, y, dx, dy, W, H):
    NOISE = 0.3         # +/- direction noise radians
    R = 0.10*min(W, H)  # neighbors within R
    V = 0.02*min(W, H)  # velocity -> displacement in each time step
    move_all(x, y, dx, dy, V, W, H)
    ax = [0.0]*len(x)
    ay = [0.0]*len(x)
    for k in range(len(x)):
        ax[k],ay[k] = neighbor_average_direction(x, y, dx, dy, k, R)
        t = math.atan2(ay[k],ax[k]) + (NOISE - 2*NOISE*random.random())
        ax[k] = math.cos(t)
        ay[k] = math.sin(t)
    dx[:] = ax   # update the orginal direction vector
    dy[:] = ay
    plt.clf()    # clear the figure
    plt.quiver(x, y, dx, dy) # quiver plot: a field of arrows
    plt.xlim((0, W))
    plt.ylim((0, H))
#----------------------------------------
```

download code นี้ได้

```
# HW5: Vicsek Model
# พิมพ์เลขประจำตัว ชื่อและนามสกุลของนิสิต

def gen_data(N, W, H):
    # ควรเขียนฟังก์ชันนี้ให้เสร็จเป็นฟังก์ชันแรก


    return ?, ?, ?, ?

def move_all(x, y, dx, dy, d, W, H):
    # ควรเขียนฟังก์ชันนี้ให้เสร็จเป็นฟังก์ชันที่สอง
    # ถ้ายังไม่เขียน code ของฟังก์ชันนี้
    # ให้ใส่คำสั่ง return ไว้ที่บรรทัดแรกของฟังก์ชันนี้
    return   # ลบบรรทัดนี้ออก ก่อนจะเริ่มเขียนคำสั่งในฟังก์ชันนี้

def neighbor_average_direction(x, y, dx, dy, k, R):
    # ถ้ายังไม่เขียน code ของฟังก์ชันนี้
    # ให้ใส่คำสั่ง return dx[k],dx[k] ไว้ที่บรรทัดแรกของฟังก์ชันนี้
    return dx[k], dy[k]   # ลบบรรทัดนี้ออก ก่อนจะเริ่มเขียนคำสั่งในฟังก์ชันนี้
```

```
#----------------------------------------
main()
```

จะไม่ตรวจ ในกรณีที่

- ส่งชุดคำสั่งที่เขียนผิดหลักไวยากรณ์ของภาษา
- `import package` ใด ๆ เพิ่มเติม
- เขียนคำสั่งใด ๆ เพิ่มเติมนอกฟังก์ชัน
- สร้างตัวแปรใด ๆ เพิ่มเติมนอกฟังก์ชัน
- แก้ไขบรรทัดที่ def ฟังก์ชัน

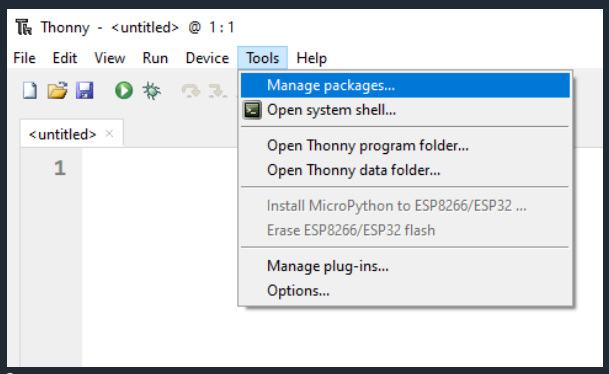
ส่งเฉพาะ code ส่วนสีเหลืองนี้
ในแฟ้ม HW5_*เลขประจำตัวนิสิต*.py

หมายเหตุ: เมื่อนิสิตเรียนถึงหัวข้อ NumPy (สัปดาห์ท้าย ๆ) ก็จะสามารถปรับการทำงานของโปรแกรมนี้ให้สั้นลง ทำงานได้เร็วขึ้น และรองรับฝูงนกที่มีปริมาณมากได้

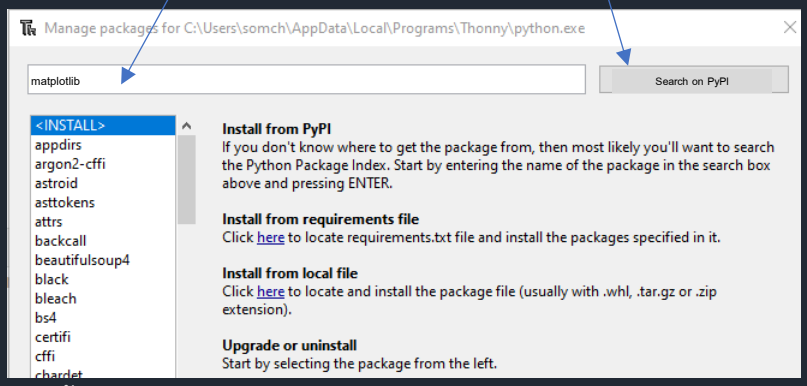
วิธีติดตั้ง matplotlib ใน Thonny

โปรแกรมในการบ้านนี้ ต้องการคลังคำสั่งที่ชื่อว่า `matplotlib` ซึ่งไม่มีให้มากับ `Thonny` ต้องถูกติดตั้งเพิ่มเติม ทำได้ดังนี้

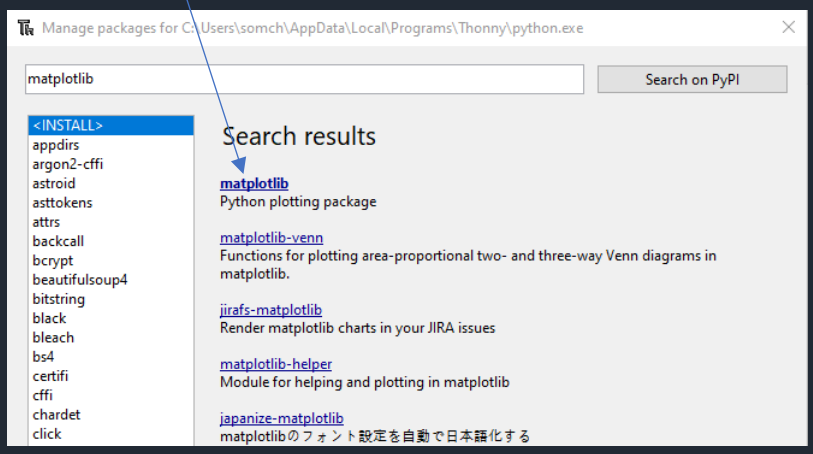
- ใน `Thonny` เลือกเมนู `Tools -> Manage packages`



- ใส่คำว่า **`matplotlib`** และกดปุ่ม **`Search on PyPI`**



- จากนั้นคลิกเลือก



- แล้วก็กดปุ่ม `Install` รอจนเสร็จ แล้วก็กดปุ่ม `Close`

การใช้โปรแกรมใน Colab

โปรแกรมต้นฉบับสามารถทำงานใน `Colab` ได้ด้วย (ซึ่งมี `matplotlib` ให้แล้ว) แต่ผลลัพธ์ที่ได้จะถูกจำกัดจำนวน `frames` ของภาพเคลื่อนไหวไว้ตามที่กำหนดในโปรแกรม ในโปรแกรมกำหนดไว้ `60` ภาพ ไม่แนะนำให้ตั้งจำนวนมาก เพราะจะช้า แต่สามารถลองเปลี่ยนได้ในส่วนของคำสั่ง **`frames=( 60 if in_colab else None),`** ในฟังก์ชัน `main`

ถึงแม้จะใช้ `Colab` กับโปรแกรมนี้ได้ แต่ขอแนะนำให้เขียนใน `Thonny` จะคล่องตัวกว่า