

ฟังก์ชันแบบเวียนเกิด

Recursive Functions

# ความสัมพันธ์เวียนเกิด (Recurrences)

- การเขียนความสัมพันธ์ของจำนวนเต็มในลำดับ

$$0, 1, 2, 3, 4, \dots \quad a_n = a_{n-1} + 1 \quad \text{เมื่อ } n > 0, a_0 = 0$$

$$3, 5, 7, 9, 11, \dots \quad a_n = a_{n-1} + 2 \quad \text{เมื่อ } n > 0, a_0 = 3$$

$$0, 1, 3, 6, 10, 15, \dots \quad a_n = a_{n-1} + n \quad \text{เมื่อ } n > 0, a_0 = 0$$

$$0, 1, 1, 2, 3, 5, 8, \dots \quad f_n = f_{n-1} + f_{n-2} \quad \text{เมื่อ } n > 1, f_0 = 0, f_1 = 1$$

$$a_n = 0, 1, 3, 6, 10, 15, \dots$$

- รู้ว่า  $a_n = (0+1+2+\dots+n)$

```
def a(n):  
    s = 0;  
    for i in range(n+1):  
        s += i  
    return s
```

แบบที่ 1

- รู้ว่า  $a_n = a_{n-1} + n$  เมื่อ  $n > 0, a_0 = 0$

```
def a(n) {  
    if n <= 0:  
        return 0  
    else:  
        return a(n-1) + n;
```

แบบที่ 2

# ฟังก์ชันหาค่า $n!$

```
def fac(n):  
    f = 1  
    for i in range(1, n+1):  
        f = f * i  
  
    return f
```

แบบที่ 1

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 2 \times 1$$

```
def fac(n):  
    if n == 0:  
        return 1  
    f = fac(n-1)  
    return n * f
```

แบบที่ 2

กรณีเล็กสุด

ตามนิยามของแฟกทอเรียล

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n \geq 1 \end{cases}$$

# ข้อดี-ข้อด้อย

การเขียนแบบ recursive มีทั้งข้อดีและข้อด้อย

ข้อดี

- สั้นกระชับรัด
- ในบางกรณี มุมมองแบบ recursive จะทำให้เห็นวิธีแก้ปัญหได้ง่ายขึ้น
- ถ้าจำนวนชั้นของ loop ไม่ "คงที่" การใช้ recursive จะง่ายกว่ามาก

ข้อด้อย

- บางครั้งทำงานช้ากว่าแบบ loop
- ใช้หน่วยความจำมากกว่า

## ตัวอย่าง : `flatten_list`

เขียนฟังก์ชัน `flatten_list` ซึ่งรับ list of lists of lists ...  
เช่น `[[1,2,3],4,[5,[6,7,[8,9],10],[11,12]]]`  
เพื่อคืน list ที่มี สมาชิกทุกตัวของ list เดิม เช่น

input: `[[1,2,3],4,[5,[6,7,[8,9],10],[11,12]]]`

output: `[1,2,3,4,5,6,7,8,9,10,11,12]`

Hint: ถ้าอยากทดสอบว่า x เป็น list หรือไม่ ใช้

- `type(x) is list` หรือ
- `isinstance(x,list)`

# flatten\_list

```
def flatten_list(d):  
    flat = []  
    for e in d:  
        if type(e) is list:  
            flat += flatten_list(e)  
        else:  
            flat.append(e)  
    return flat  
  
x = [1, [[[2,3],4]], [[5,6],7],8]  
print(flatten_list(x))
```

# flatten\_list (อีกแบบ)

```
def flatten_list(x):  
    if len(x) == 0:  
        return x  
    h = x[0:1]  
    if isinstance(x[0], list):  
        h = flatten_list(x[0])  
    return h + flatten_list(x[1:])  
  
x = [1, [[2,3],4], [[5,6],7],8]  
print(flatten_list(x))
```



# ลองเขียนดู : ผลรวมของจำนวนในลิสต์

จงเขียนฟังก์ชัน **total(x)** ที่รับ **x** ซึ่งเป็นลิสต์ที่เก็บจำนวน แล้วคืนผลบวกของจำนวนทุกตัวใน **x**  
ให้เขียนแบบ recursive ไม่มีการใช้วงวน

ข้อแนะนำ :

ให้  $\text{total}(n) = x[0] + \dots + x[n-1]$  ย่อมหมายความว่า  
 $\text{total}(n) = \text{total}(n-1) + x[n-1]$  สำหรับ  $n > 1$

```
def total(x):  
    if len(x) == 0 :  
        _____  
    return _____
```

# ลองเขียนดู : is\_palindrome

จงเขียนฟังก์ชัน **is\_palindrome(x)** ที่รับสตริง **x**  
แล้วคืน True ถ้า x เป็น palindrome  
False ถ้า x ไม่ใช่ palindrome

ต.ย. "21022012", "ทายาท" เป็น palindrome

```
def is_palindrome(x):
```

```
    if len(x) == 0 :
```

```
        _____  
  
    return _____ and _____
```