

Nested Structures

ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย

๒๕๖๒

More Data & Flow Controls

เรียนไปแล้ว

`int, float,
str, bool
list, dict`

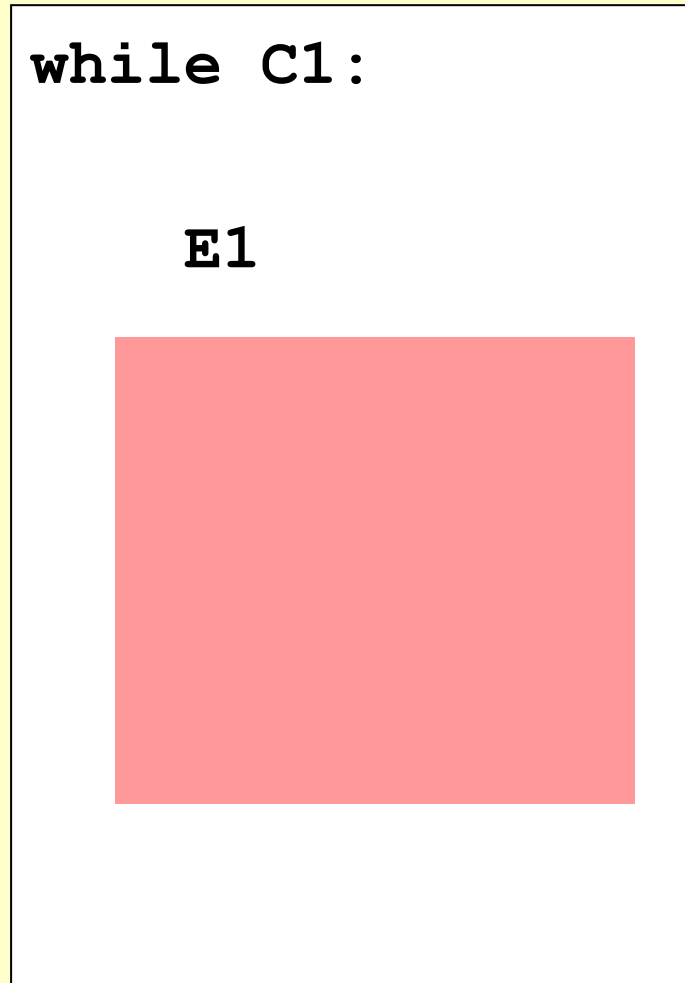
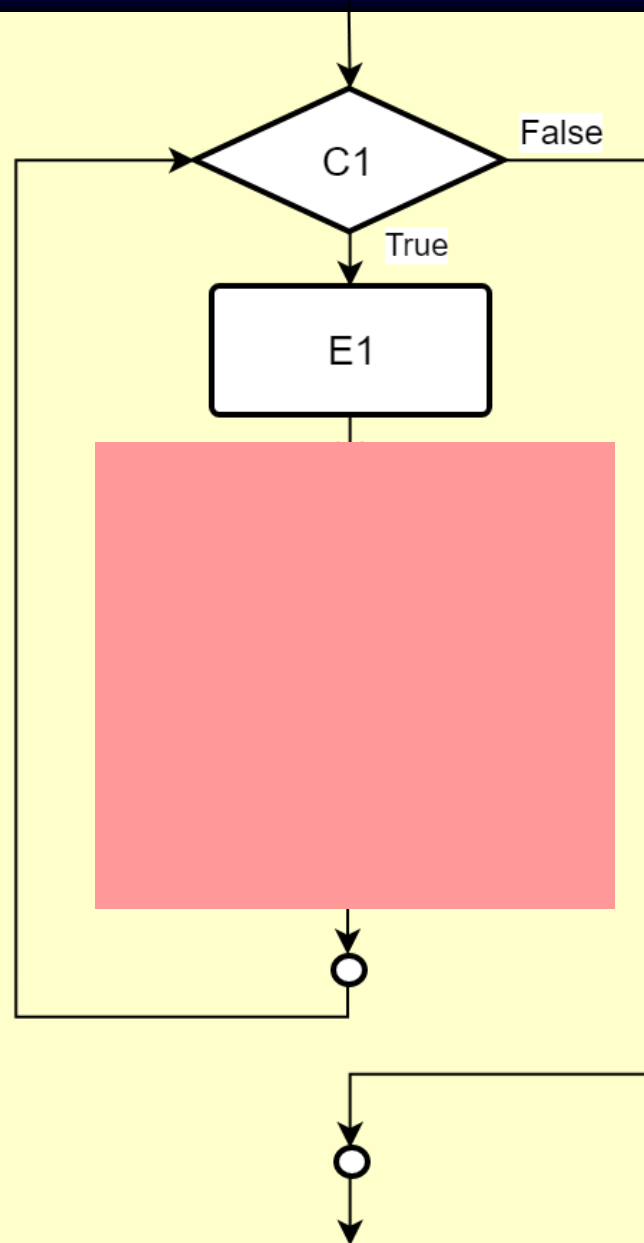
`if-elif-else
while
for
break
function`

จะเรียนต่อไป

`tuple, set,
list, dict,
numpy array,
class/object`

`nested loop
comprehension
recursion`

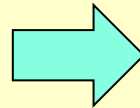
Nested Loops: while ซ้อน while



ตัวอย่าง: หา หรม. ของจำนวนเต็มหลาย ๆ คู่

```
x = input().split()
while x[0] != 'q':
    a,b = int(x[0]),int(x[1])
    while b != 0:
        a,b = b, a % b
    print(a)
    x = input().split()
```

5	8
143	65
q	



1
13

a	b
143	65
65	13
13	0

แบบฝึกหัด: Factorization

```
def factor(N):  
    f = []  
    k = 2  
    while k <= N:  
        ถ้า k หาร N ลงตัว  
        ก็ วนหาร N ด้วย k  
        จน k ไม่เป็น factor ของ N  
        เพิ่ม k และจำนวนครั้งที่หาร ใส่ใน f  
        k += 1  
    return f
```

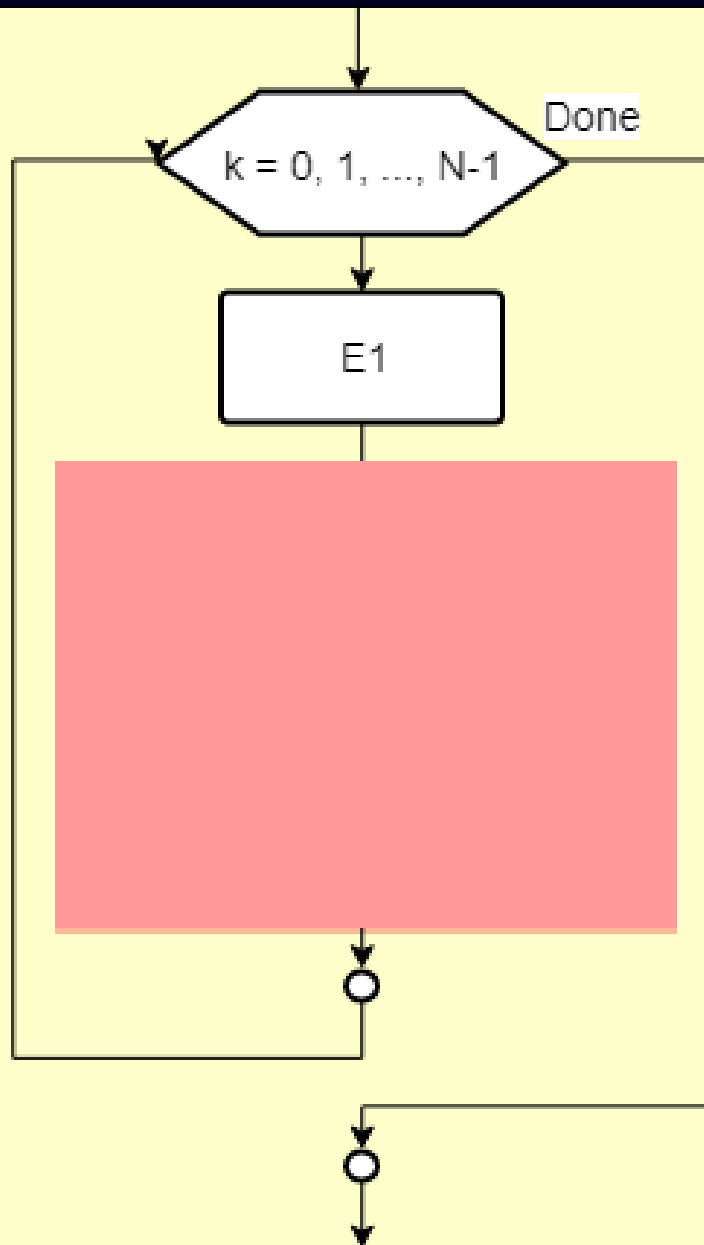
เอ๊ะ ทำไมเราไม่ใช้
for k in range(2, N+1):

N = 200, k = 2, N = 200 → 100 → 50 → 25
N = 25, k = 3
N = 25, k = 4
N = 25, k = 5, N = 25 → 5 → 1

f = [[2, 3], [5, 2]]

$$200 = 2^3 \cdot 5^2$$

Nested Loops: for ชั้น for



```
for k in range(N) :
```

```
    E1
```



ตัวอย่าง

```
for i in range(3):  
    for j in range(4):  
        print(i, j)
```

i	j

0	0
	1
	2
	3
1	0
	1
	2
	3
2	0
	1
	2
	3

```
for i in range(3):  
    for j in range(i, 4):  
        print(i, j)
```

i	j

0	0
	1
	2
	3
1	1
	2
	3
2	2
	3

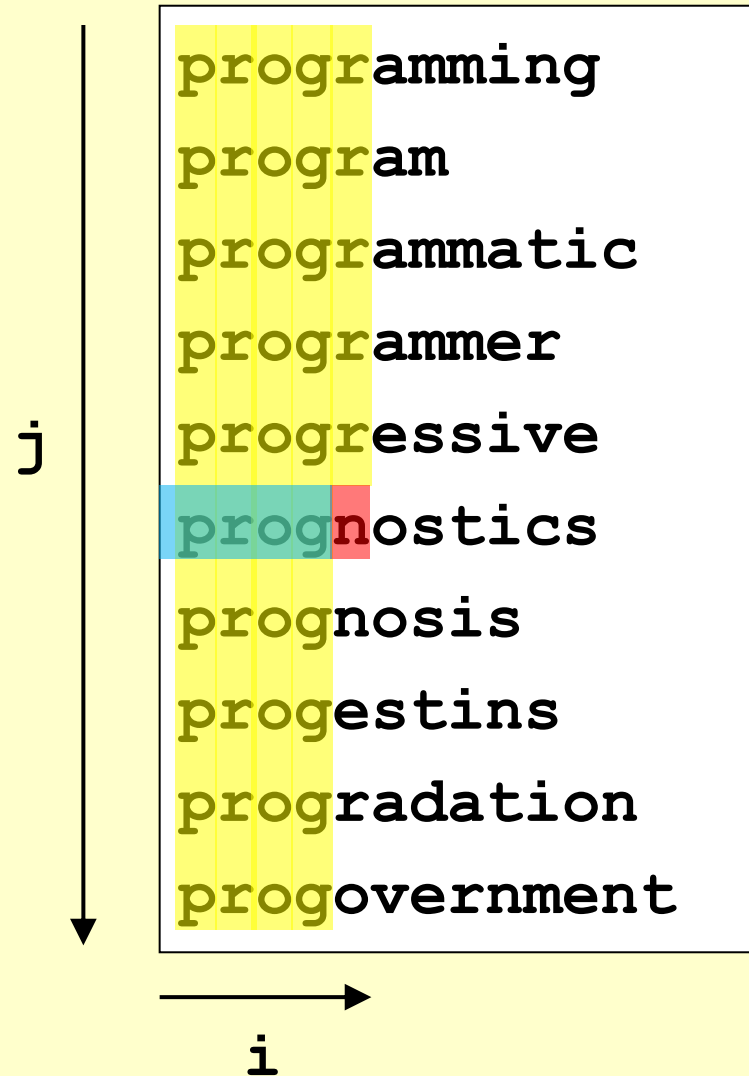
```
for i in range(3):  
    for j in range(i+1, 4):  
        print(i, j)
```

i	j

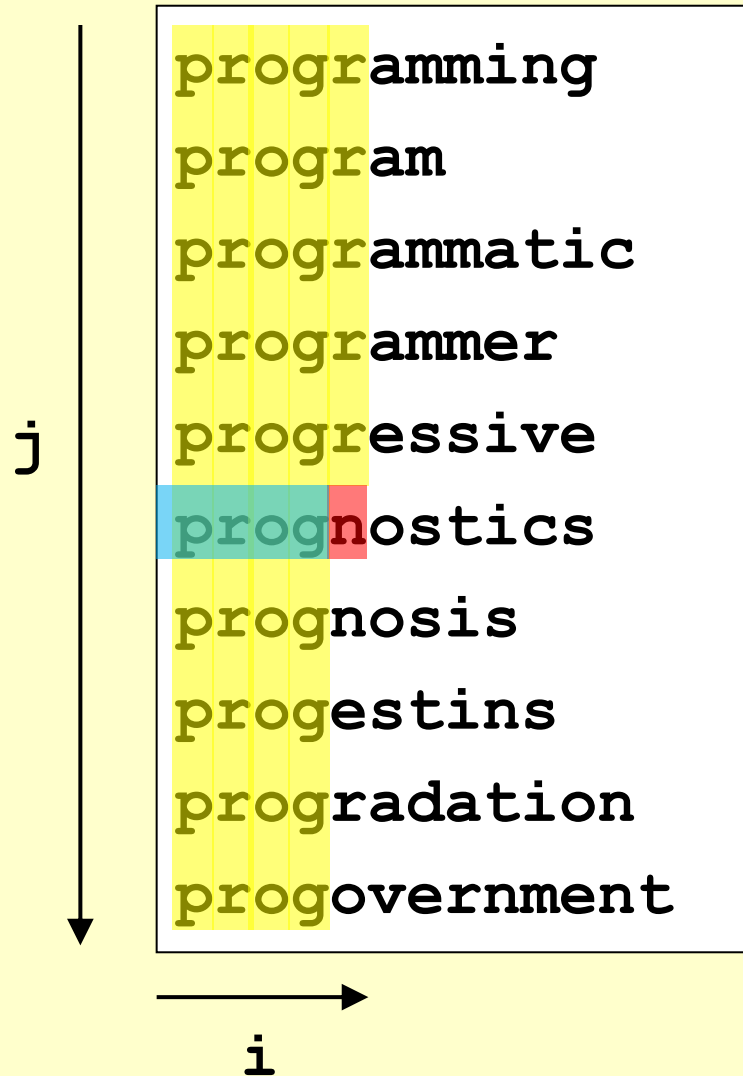
0	1
	2
	3
1	2
	3
2	3

แจกแจง index ของ
ทุกคู่ข้อมูลในลิสต์

ตัวอย่าง: ฟังก์ชันหา prefix ยาวสุดของรายการสตริง



ตัวอย่าง: ฟังก์ชันหา prefix ยาวสุดของรายการสตริง



```
def longest_prefix(words):  
    for i in range(len(words[0])):  
        c = words[0][i]  
        for j in range(1, len(words)):  
            if i >= len(words[j]) or \  
               c != words[j][i]:  
                return words[j][:i]  
    return words[0]
```

ตัวอย่าง: ฟังก์ชันตรวจสอบข้อมูลซ้ำกันในลิสต์

```
def has_duplicate( x ):  
    for i in range(len(x)-1):  
        for j in range(i+1, len(x)):  
            if x[i] == x[j]:  
                return True  
    return False
```

i	j

0	1
	2
	3
1	2
	3
2	3

x = [11, 34, 22, 34]

ลุยตรวจทุกคู่

หมายเหตุ: วิธีนี้ตรวจสอบความซ้ำซ้อนที่ค่อนข้างช้ามาก

ตัวอย่าง: Pairwise Coprime

- A set of integers can be called **coprime** if its elements share no common positive factor except 1.

21	15	35
3×7	3×5	5×7

- A stronger condition on a set of integers is **pairwise coprime**, which means that a and b are coprime for every pair (a, b) of different integers in the set.

21	10	121
3×7	2×5	11×11

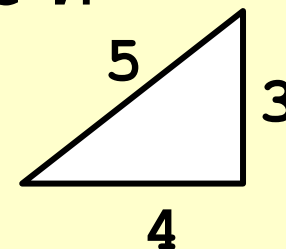
- The set $\{2, 3, 4\}$ is coprime, but it is not pairwise coprime since 2 and 4 are not relatively prime.

ตัวอย่าง: Pairwise Coprime

```
def gcd(a,b):  
    while b != 0:  
        a,b = b, a%b  
    return a  
  
def is_pairwise_coprime( d ):  
    for i in range(len(d)-1):  
        for j in range(i+1, len(d)):  
            if gcd(d[i],d[j]) != 1:  
                return False  
    return True
```

แบบฝึกหัด: Primitive Pythagorean Triple

- Pythagorean triple: จำนวนเต็ม a , b และ c ที่ $a^2 + b^2 = c^2$ เช่น $(3, 4, 5)$
- ถ้า (a,b,c) เป็น Pythagorean triple (ka, kb, kc) ก็เป็นด้วย โดยที่ $k = 1, 2, 3, 4, \dots$
- เราต้องการ Primitive Pythagorean triple คือ Pythagorean triple (a,b,c) ที่ a, b และ c เป็น coprime (คือมี ห.ร.ม. เป็น 1)
- จงเขียนโปรแกรมหา Pythagorean triple ทุกค่าที่ $a \leq b \leq c \leq M$ โดยที่ M คือ input เช่น ให้ $M = 20$ จะได้



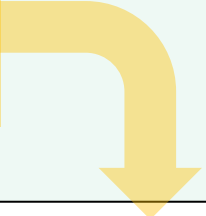
$[3, 4, 5], [5, 12, 13], [8, 15, 17]$

แบบฝึกหัด: Primitive Pythagorean Triple

```
def gcd(a,b):  
    while b != 0:  
        a,b = b, a%b  
    return a  
  
def is_coprime(a, b, c):  
    ???  
  
def primitive_Pythagorean_triple( M ):  
    triple = []  
    for ??? in range( ??? ):  
        for ??? in range( ??? ):  
            for ??? in range( ??? ):  
                ???  
  
    return triple
```

ย้ายวงวนชั้นในไปเขียนเป็นฟังก์ชัน

```
def is_pairwise_coprime(d):  
    for i in range(len(d)-1):  
        for j in range(i+1, len(d)):  
            a,b = d[i],d[j]  
            while b != 0:  
                a,b = b, a%b  
            if a != 1:  
                return False  
    return True
```



```
def gcd(a,b):  
    while b != 0:  
        a,b = b, a%b  
    return a
```

เข้าใจง่ายขึ้น

```
#-----  
def is_pairwise_coprime( d ):  
    for i in range(len(d)-1):  
        for j in range(i+1, len(d)):  
            if gcd(d[i],d[j]) != 1:  
                return False  
    return True
```

ย้ายวงวนชั้นในไปเขียนเป็นฟังก์ชัน

```
n = int(input())
count = 0
for k in range(n):
    t = input()
    c = 0
    for ch in t:
        if "0" <= ch <= "9":
            c += 1
    count += c
print(count)
```

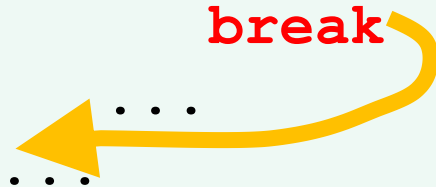
นับตัวเลขจากข้อมูล
หลายบรรทัด

```
def count_digits(s):
    c = 0
    for ch in s:
        if "0" <= ch <= "9":
            c += 1
    return c

#-----
n = int(input())
count = 0
for k in range(n):
    t = input()
    count += count_digits(t)
print(count)
```

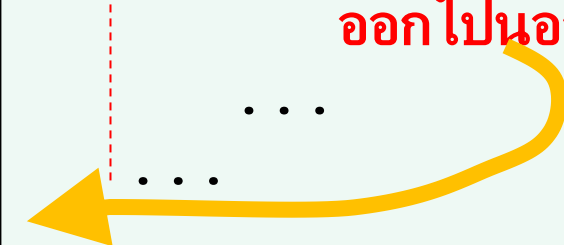

break ออกไปหลาย ๆ ชั้นไม่ได้

```
...  
for ...  
    ...  
    for ...  
        ...  
        if condition:  
            ...  
            break  
        ...  
    ...  
...
```



break จะกระโดด
ออกมาจากวงวนที่
break อยู่

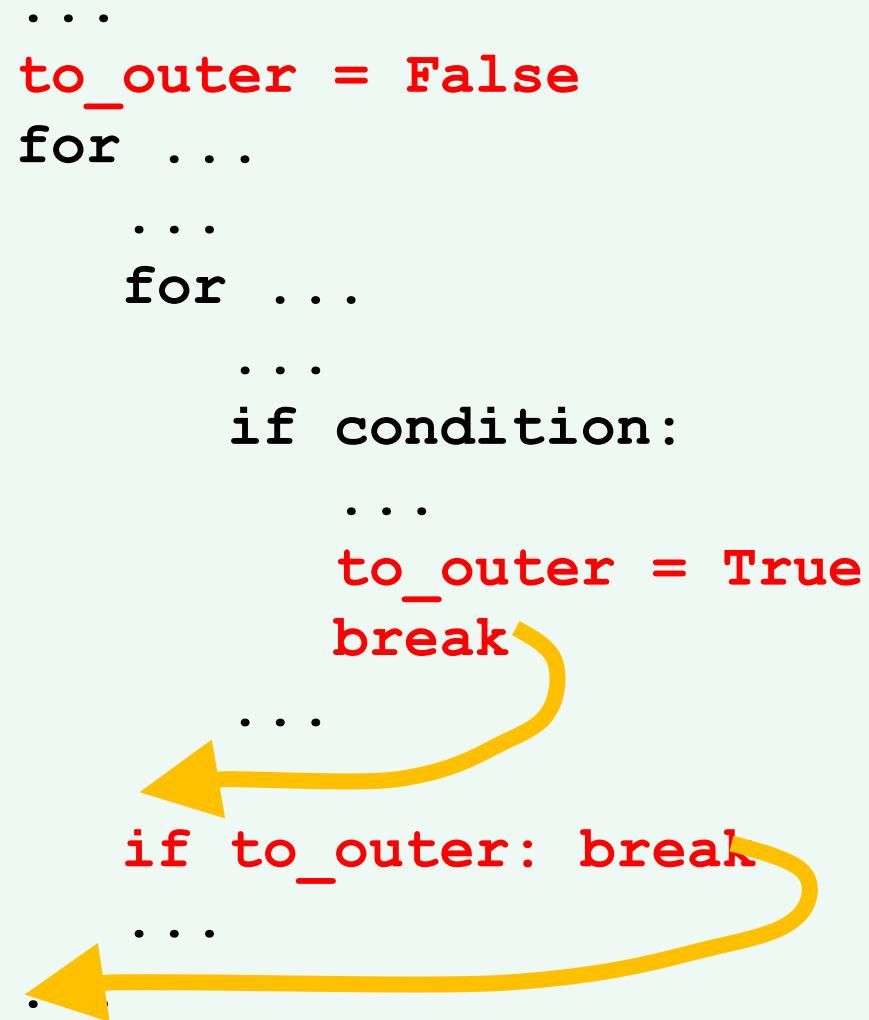
```
...  
for ...  
    ...  
    for ...  
        ...  
        if condition:  
            ...  
            ออกไปนอกสุด  
        ...  
    ...  
...
```



ถ้าต้องการให้ break
กระโดดออกมาวงนอก ๆ
จะอย่างไร

break ออกไปหลาย ๆ ชั้นด้วย ตัวแปรเสริม

```
...  
to_outer = False  
for ...  
    ...  
    for ...  
        ...  
        if condition:  
            ...  
            to_outer = True  
            break  
        ...  
    if to_outer: break  
    ...  
.
```



break ออกไปหลาย ๆ ขั้นตอนด้วย ตัวแปรเสริม

```
prefix = words[0]
for i in range(len(words[0])):
    c = words[0][i]
    for j in range(1, len(words)):
        if i >= len(words[j]) or \
            c != words[j][i]:
            prefix = words[j][:i]
```

อยากออกไปนอกสุด



หา longest prefix
ของคำใน words

```
prefix = words[0]
found = False
for i in range(len(words[0])):
    c = words[0][i]
    for j in range(1, len(words)):
        if i >= len(words[j]) or \
            c != words[j][i]:
            prefix = words[j][:i]
            found = True
```

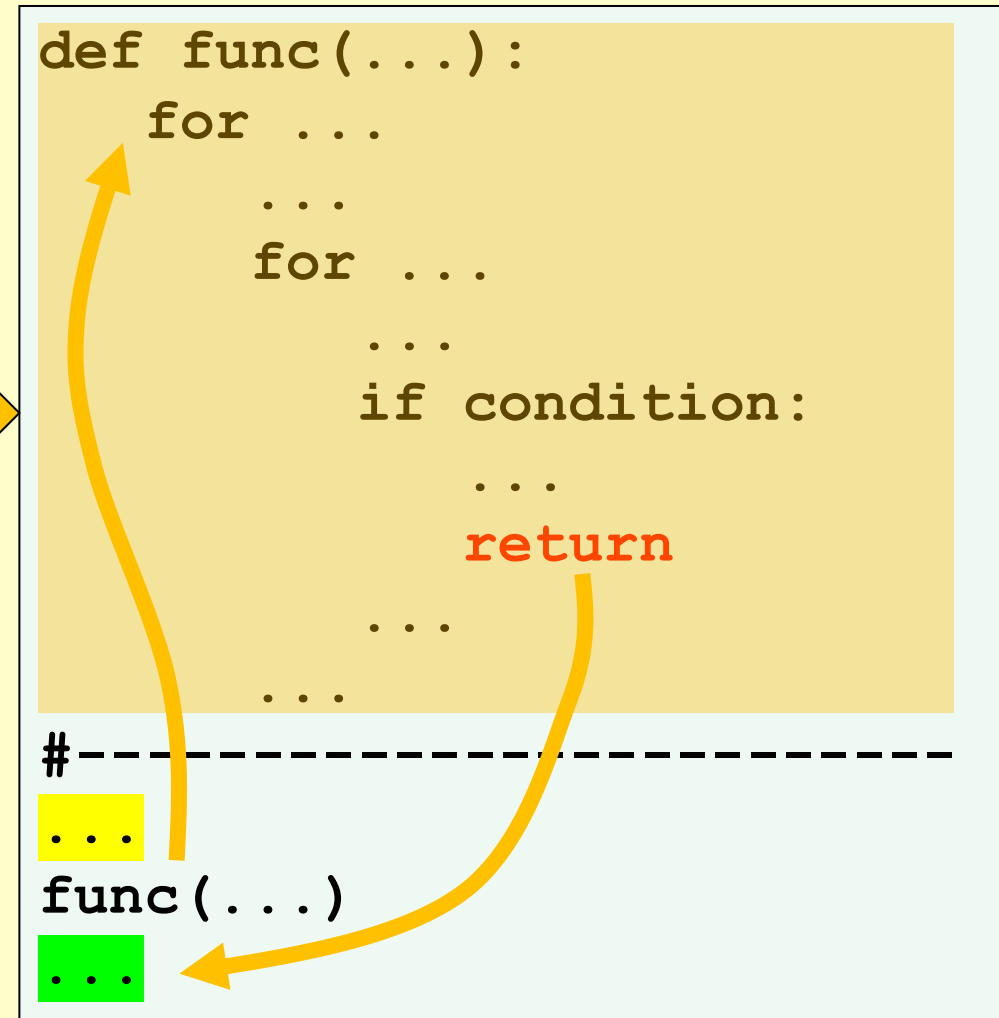
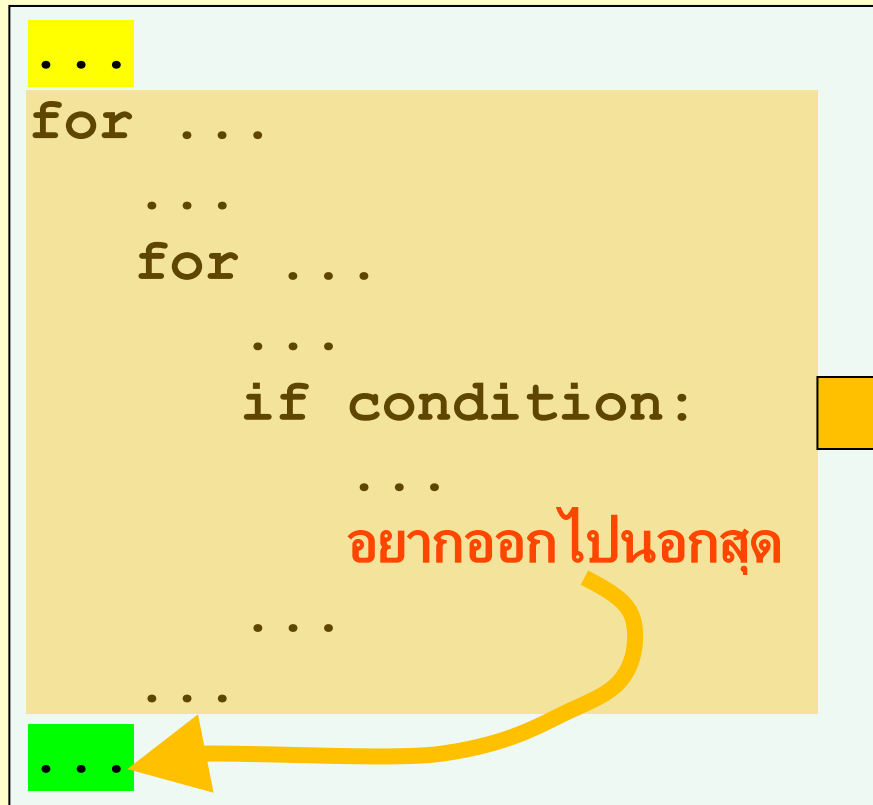
break



if found: break



break ออกไปหลาย ๆ ชั้นด้วย การแยกออกเป็นฟังก์ชัน



break ออกไปหลาย ๆ ขั้นตอน ด้วยการแยกออกเป็นฟังก์ชัน

```
prefix = words[0]
for i in range(len(words[0])):
    c = words[0][i]
    for j in range(1, len(words)):
        if i >= len(words[j]) or \
            c != words[j][i]:
            prefix = words[j][:i]
```

อยากออกไปนอกสุด



```
def longest_prefix(words):
    for i in range(len(words[0])):
        c = words[0][i]
        for j in range(1, len(words)):
            if i >= len(words[j]) or \
                c != words[j][i]:
                return words[j][:i]
    return words[0]
```

```
prefix = longest_prefix(words)
```

Nested Lists: ลิสต์ซ้อนในลิสต์

- เก็บข้อมูลที่ประกอบด้วยข้อมูลย่อยที่เป็นลิสต์
 - `["Ranee", 1989, ["Plerng Boon", "Bubphe Sanniwat", "Krong Kam"]]`
- เก็บข้อมูลหลาย ๆ ตัว ที่แต่ละตัวมีข้อมูลย่อย ๆ
 - `[[6131001021, 3.8], [6130020221, 3.7]]`
- เก็บข้อมูลชั่วคราวเพื่อนำไปประมวลผล (เช่น sort ตามความยาว)
 - `["your", "kiss", "is", "on", "my", "list"]`
`[[4, "your"], [4, "kiss"], [2, "is"], [2, "on"], [2, "my"], [4, "list"]]`
- เก็บเมทริกซ์
 - `[[1, 2, 3, 0], [2, 3, 0, 1], [4, 1, 2, 2]]` $\begin{bmatrix} 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 4 & 1 & 2 & 2 \end{bmatrix}$

สร้าง nested list

Ranee

1989

Plerng Boon, Bubphe Sanniwat, Krong Kam

```
name = input()
byear = int(input())
series = input().split(", ")
actress = [name, byear, series]
```

```
[ "Ranee", 1989,
  [ "Plerng Boon", "Bubphe Sanniwat", "Krong Kam" ] ]
```

สร้าง nested list

3
6131001021 3.8
6130020221 3.7
6130150721 2.7

```
n = int(input())  
students = []  
for i in range(n):  
    student_ID, gpax = input().split()  
    gpax = float(gpax)  
    students.append( [student_ID, gpax] )
```

[["6131001021", 3.8], ["6130020221", 3.7], ["6130150721", 2.7]]

สร้าง nested list

```
["your", "kiss", "is", "on", "my", "list"]
```

```
[ [4, "your"], [4, "kiss"], [2, "is"],  
  [2, "on"], [2, "my"], [4, "list"] ]
```

```
def sort_by_length( words ):  
    x = []  
    for w in words:  
        x.append( [len(w), w] )  
    x.sort()  
    for k in range(len(x)):  
        words[k] = x[k][1]
```

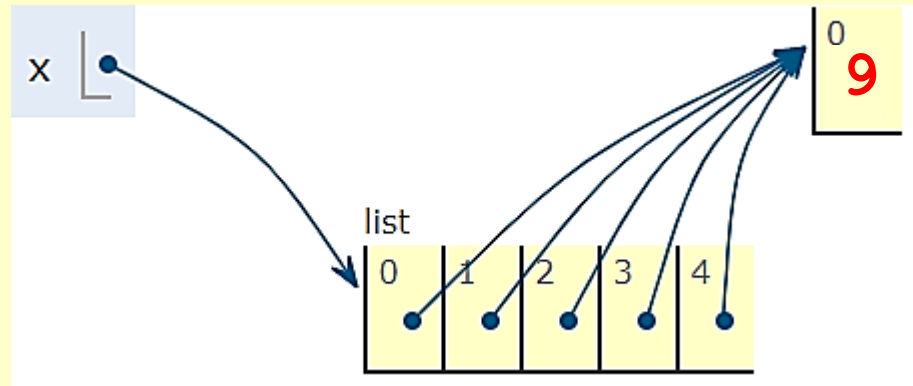
```
[ [2, "is"], [2, "my"], [2, "on"],  
  [4, "kiss"], [4, "list"], [4, "your"] ]
```

```
["is", "my", "on", "kiss", "list", "your"]
```

ข้อควรระวัง

`x = [0]*5` ได้ `[0, 0, 0, 0, 0]`

`x = [[0]]*5` ได้ `[[0], [0], [0], [0], [0]]`
`x[0][0] = 9` ได้ `[[9], [9], [9], [9], [9]]`



```
x = []  
for i in range(5):  
    x.append([0])
```

แบบนี้แต่ละช่องเป็นคอนละลิสต์

แบบฝึกหัด: First Fit / Best Fit

จงเขียนโปรแกรมแบ่งรายการของจำนวนเต็มไม่เกิน 100 ออกเป็นรายการย่อย ๆ แต่ละรายการมีผลรวมไม่เกิน 100 ให้ได้จำนวนรายการน้อย ๆ

[20,90,10,80]

First Fit

[[20]]

[[20], [90]]

[[20,10], [90]]

[[20,10], [90], [80]]

เจอลิสต์แรกที่ใช้ได้ก็ใส่เลย

Best Fit

[[20]]

[[20], [90]]

[[20], [90,10]]

[[20,80], [90,10]]

เลือกลิสต์อันที่ใส่แล้ว
มีผลรวมใกล้ 100 ที่สุด

Nested List as Matrix

3				[[1.0,	2.0,	3.0,	0.0],	
1	2	3	0		[2.0,	3.0,	0.0,	1.0],	
2	3	0	1		[4.0,	1.0,	2.0,	2.0]]
4	1	2	2						

```
def read_matrix():  
    m = []  
    nrows = int(input())  
    for k in range(nrows):  
        x = input().split()  
        r = []  
        for e in x:  
            r.append( float(e) )  
        m.append(r)  
    return m
```

print_matrix(M)

```
def print_matrix( M ) :  
    if len(M) == 1 :  
        print(M)  
    else :  
        print("[ " + str(M[0]))  
        for i in range(1, len(M)-1) :  
            print(" " + str(M[i]))  
        print(" " + str(M[-1]) + "]")
```

```
M = [[1,2,3,4], [2,2,1,3], [2,6,7,7]]  
print_matrix(M)
```

```
[[1, 2, 3, 4]  
 [2, 2, 1, 3]  
 [2, 6, 7, 7]]
```

add_matrix(A, B)

```
def add_matrix(A, B):  
    C = []  
    nrows = len(A)  
    ncols = len(A[0])  
    for i in range(nrows):  
        C.append( [0.0]*ncols )  
        for j in range(ncols):  
            C[i][j] = A[i][j] + B[i][j]  
    return C
```

```
A = read_matrix()  
B = read_matrix()  
C = add_matrix(A, B)  
print_matrix(C)
```

แบบฝึกหัด: $\text{mult}(A, B)$

```
def mult(A, B):  
    C = []
```

```
    return C
```

$$C_{i,j} = \sum_{k=0}^{q-1} A_{i,k} B_{k,j}$$

A มีขนาด $p \times q$, B มีขนาด $q \times r$, C มีขนาด $p \times r$

List Comprehension

วิธีการเขียนคำสั่งสร้างลิสต์ที่สั้น และมีประสิทธิภาพ

```
# ให้ x เป็นลิสต์ของ int  
t = []  
for e in x:  
    t.append(2*e)
```

```
# แปลงทุกค่าใน x เป็นอีกอย่าง  
t = [ 2*e for e in x ]
```

```
t = []  
for e in x:  
    if e >= 0:  
        t.append(e)
```

```
# เลือกบางค่าในลิสต์ x  
t = [ e for e in x if e >= 0 ]
```

```
t = []  
for e in x:  
    if e >= 0:  
        t.append(2*e)
```

```
# เลือกบางค่าในลิสต์ x มาแปลง  
t = [2*e for e in x if e >= 0 ]
```


ตัวอย่าง: อ่านรายการของจำนวนบนบรรทัดเดียวกัน

```
x = input().split()
d = []
for e in x:
    d.append( int(e) )
...
```

```
d = []
for e in input().split():
    d.append( int(e) )
...
```

ใช้ list comprehension

```
d = [int(e) for e in input().split()]
...
```

```
d = [float(e) for e in input().split()]
...
```

ตัวอย่าง: เรียงลำดับสตริงตามความยาว

```
def sorted_by_length(s):  
    t = []  
    for e in s:  
        t.append( [len(e), e] )  
    t.sort()  
    r = []  
    for n,e in t:  
        r.append( e )  
    return r
```

ใช้ list comprehension

```
def sorted_by_length(s):  
    t = [[len(e),e] for e in s]  
    t.sort()  
    return [e for n,e in t]
```

List Comprehension เร็วกว่า

```
import timeit

def for_loop(n):
    t = []
    for i in range(n):
        t.append(n)
    return t

def comprehension(n):
    return [n for i in range(n)]

def time(func):
    print(timeit.timeit(func+" (1000000)",
        globals=globals(), number=100))

time("for_loop")          # 9.2609192
time("comprehension")    # 4.7720880999999995
```