## 1-2 ไวยากรณ์ที่ต้องใช้บ่อย

```
9/2=4.5   9//2=4   9%2=1   2**3=8
int(3.9)=3  round(6.5)=6  abs(-9.0)=9.0
eval("32+2")=34   str(36)="36"
print("x:{0} y:{1}".format(x, y))
print(name, end=" ")

s = input("...")          # ได้ str เสมอ
x = eval(input("..."))    # แปลงเลข

name = "Mark"  # M0 a1 r2 k3 / -4 -3 -2 -1
name[1:2]'a'  name[0:3]'Mar'  name[-1]'k'
name[:4]'Mark'  name[1:]'ark'
len() upper() lower() count() capitalize()
title() rstrip() split()  a+b ต่อ  "x"*3 ซ้ำ

a.append(x)  a.extend(b)  a.remove(x)
a.clear()
sum(a) min(a) max(a) len(a)  a[1:3]
a + b ต่อ list     a * 2 ทำซ้ำ
list('home') ['h','o','m','e']
```

## 2 โครงสร้างควบคุม

Python ไม่ใช้ `;` และ `{ }` แต่ใช้**ย่อหน้า TAB**

```
score = eval(input("Enter score: "))
if score >= 80:
    grade = 'A'
elif score >= 70:
    grade = 'B'
else:
    grade = 'F'

count = 1
while count <= 5:
    print(count)
    count += 1

while True:
    if stuff == "q":  break      # ออกจากลูป
    continue                     # ข้ามไปรอบถัดไป

range(3,7) 3 4 5 6   range(5) 0 1 2 3 4
range(1,1) ว่าง!   range(3,10,2) 3 5 7 9
range(6,0,-1) 6 5 4 3 2 1

for i in range(5):
for x in a:              for ch in word:
for i in range(len(a)):
y = [x-10 for x in grades if x > 50]
```

`s.isdigit()` · `s.isupper()` · `isinstance(x, int)`

## 3 ฟังก์ชัน · Scope · CSV

```
def triple(x):
    val = x*x*x
    return val

def find_max_min(a):          # คืน 2 ค่า
    return max(a), min(a)
max_x, min_x = find_max_min(a)

x = 0                         # global
def add_x():
    global x            # ขอก่อนแก้ค่า
    x += 2

import random
n = random.randint(3, 9)

import csv
def readCSV(filename):
    infile = open(filename, 'r')
    mylist = []
    csv_obj = csv.reader(infile)
    for row in csv_obj:
        mylist.append(row)
    infile.close()
    return mylist

# อ้างด้วยชื่อคอลัมน์
reader = csv.DictReader(infile)
for row in reader:
    name.append(row["Name"])
```

ขอ global แล้ว**ห้ามมี local ชื่อเดียวกัน** · **บรรยาย 4** ใช้ `readRecordToList` คือ `readCSV` ที่เพิ่ม `heading = next(infile)` ข้ามหัวตาราง

## 4 swap

```
def swap(a, i, j):
    tmp = a[i]
    a[i] = a[j]
    a[j] = tmp
    return a
```

## 4 Bubble Sort ~ N*N

```
def bubbleSort(a, N):
    for i in range(N):
        for j in range(N - 1, 0, -1):
            if a[j] < a[j - 1]:
                a = swap(a, j, j - 1)
    return a

data = [8, 7, 2, 1]
data_sorted = bubbleSort(data, len(data))
print(data_sorted)
[1, 2, 7, 8]
```

ลูปใน `j` วิ่ง**ถอยหลัง** N−1 → 1 จึงดัน**ค่าน้อยขึ้นหน้า**

## 4 Selection Sort ~ N*N

```
def selectionSort(a, N):
    for i in range(0, N - 1):
        min = i
        for j in range(i + 1, N):
            if a[j] < a[min]:
                min = j
        swap(a, min, i)
    return a

key = [8, 7, 2, 1]
key_sorted = selectionSort(key, len(key))
[1, 2, 7, 8]
```

หา**ตำแหน่ง**ค่าน้อยสุดเก็บใน `min` สลับครั้งเดียวตอนจบรอบ

## 4 Insertion Sort ดีสุด N

```
def insertionSort(a, n):
    for i in range(1, n):
        key = a[i]
        j = i - 1
        while j >= 0 and a[j] > key:
            a[j + 1] = a[j]
            j -= 1
        # Insert key
        a[j + 1] = key

key = [8, 7, 2, 1]
insertionSort(key, len(key))
print(key)
[1, 2, 7, 8]
```

เหมือนเรียงไพ่ในมือ · **ไม่มี return** แก้ list เดิมโดยตรง

## 4 เปรียบเทียบความเร็ว

**Bubble ~ N*N** ง่ายแต่ช้า · **Selection ~ N*N** ประมาณ Bubble · **Insertion N*N** แต่ถ้าเรียงมาดีแล้วเหลือ **N**
วิเคราะห์จาก**จำนวนรอบของคำสั่งที่ถูกเรียกมากที่สุด** คือลูปซ้อนสองชั้น

## 4 เรียง record ด้วยกุญแจที่เป็น column

```
def insertionSortRec(r, n, key_col):
    for i in range(1, n):
        key_rec = r[i]
        j = i - 1
        while j >= 0 and \
          r[j][key_col] > key_rec[key_col]:
            r[j + 1] = r[j]
            j -= 1
        # Insert key
        r[j + 1] = key_rec
```

เทียบที่คอลัมน์ `[key_col]` แต่ย้าย**ทั้งแถว**

## 4 ตัวอย่างข้อมูล record

```
data = [
  ['Mark', "123-65-6789", 75, 78, 82], \
  ['Tim',  "123-65-5723", 67, 45, 74], \
  ['Amy',  "123-65-4542", 78, 65, 90], \
  ['Berk', "123-65-8278", 81, 74, 68] ]
key_column = 4
insertionSortRec(data, len(data),
                 key_column)
for row in data:
    print(row)
['Berk',..,68] ['Tim',..,74]
['Mark',..,82] ['Amy',..,90]
```

ถ้าคัดเฉพาะกุญแจมาเรียง ได้แค่ `[68,74,82,90]` แถวไม่ย้ายตาม

### ! กับดัก 4

- Bubble ลูปในจบที่ **1** ไม่ใช่ 0 เพราะต้องอ้าง `a[j-1]`
- `insertionSort` และ `insertionSortRec` **ไม่มี return**
- ทุกฟังก์ชันรับ `N` ต้องส่ง `len(a)` ไปด้วยเสมอ

## 5 ฟังก์ชันเรียกตัวเอง ออกสอบแน่

```
def message(times):
    print("message is called, times = ",
          times)
    if times > 0:
        message(times - 1)
    # -----
    print("message return with ", times)

if __name__ == '__main__':
    message(3)
message is called, times =  3
message is called, times =  2
message is called, times =  1
message is called, times =  0
message return with  0
message return with  1
message return with  2
message return with  3
```

ขาไปนับ 3→0 · ขากลับนับ 0→3 · `message(3)` เรียกตัวเอง **4 ครั้ง**

## 5 Divide and Conquer

- **1. Base Case** แก้ตรง ๆ ถ้าเล็กพอ
- **2. Divide** แบ่งเป็นปัญหาย่อยที่เล็กลง
- **3. Recursively solve** เรียกตัวเองแก้ปัญหาย่อย
- **4. Combine** รวมคำตอบของปัญหาย่อย

```
MERGE-SORT A[0..N-1], Left, Right
1. If N = 1, done.
2. m = N/2
3. Merge-Sort( A, Left, m )
4. Merge-Sort( A, m + 1, Right )
5. "Merge" the 2 sorted lists.
```

## 5 merge Θ(N)

```
def merge(A, p, q, r):
    left  = A[p: q+1]      # slice = copy
    right = A[q+1: r+1]
    i = 0   # index into left
    j = 0   # index into right
    k = p   # index into A[p:r+1]

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            A[k] = left[i]
            i += 1
        else:
            A[k] = right[j]
            j += 1
        k += 1
    if i < len(left):
        A[k: r+1] = left[i:]
    if j < len(right):
        A[k: r+1] = right[j:]
```

เขียนผลลง **A ตัวเดิม** ไม่คืนค่าใหม่ · ครึ่งแรก `p..q` ครึ่งหลัง `q+1..r`

## 5 merge_sort N LOG N

```
def merge_sort(A, p=0, r=None):
    if r is None:
        r = len(A) - 1
    if p >= r:            # 0 or 1 element
        return
    q = (p + r) // 2      # midpoint
    merge_sort(A, p, q)
    merge_sort(A, q + 1, r)
    merge(A, p, q, r)

A = [8, 3, 2, 9, 7, 1, 5, 4]
merge_sort(A)
print(A)
[1, 2, 3, 4, 5, 7, 8, 9]
```

## 5 Quick Sort

```
Quick-Sort(A, left, right)
  if left >= right
      return
  else
      middle <- Partition(A, left, right)
      Quick-Sort(A, left, middle - 1)
      Quick-Sort(A, middle + 1, right)
  end if
```

เลือก pivot **p** เช่นตัวแรก · ส่วนแรกคือที่ **< p** ส่วนที่สองคือ **≥ p**

## 5 Merge vs Quick

| | MERGE | QUICK |
|---|---|---|
| งานหลัก | ตอน merge | ตอน partition |
| Worst | O(n log n) | O($n^2$) |
| หน่วยความจำ | More | Less (in-place) |
| Stability | Stable | Not stable |

**Stable** = ตัวที่ค่าเท่ากันยังคงลำดับเดิม · ทั้งคู่ดีกว่า Insertion Sort

### ! กับดัก 5

- `merge_sort` **ไม่ return** และเรียกครั้งแรกแค่ `merge_sort(A)` เพราะ p, r มีค่าตั้งต้น
- ครึ่งหลังเริ่มที่ `q + 1` ไม่ใช่ `q`
- `print` ก่อนเรียกตัวเอง = ขาไป · หลังเรียกตัวเอง = ขากลับ

## 6 คลาสและวัตถุ

```
class birds():
    name = 'eagle'
    def fly(self):
        if self.name == 'eagle':
            print("I am an ", self.name,
                  " and I can fly")

eagle = birds()
eagle.fly()          # วิธีที่ 1
birds.fly(eagle)     # วิธีที่ 2
```

`self` ต้องเป็นพารามิเตอร์**ตัวแรกเสมอ** (เหมือน `this` ใน Java) · **ADT = properties + operations · class = ตัวแปรของวัตถุ + เมท็อด**

## 6 Constructor และ _str_

```
class rectangle():
    def __init__(self, x, y):
        self._x = x
        self._y = y
    def calArea(self):
        print("Area is ",
              self._x * self._y)
    def __str__(self):
        s = "Width: " + str(self._x) \
            + "\nHeight: " + str(self._y)
        return s

r1 = rectangle(2.0, 3.0)
print(r1);  r1.calArea()
Width: 2.0 / Height: 3.0 / Area is  6.0
```

`__init__` เรียก**ครั้งเดียว**ตอนสร้างวัตถุ · ตัวแปรปกปิดขึ้นต้นด้วย `_` · `__str__` เรียกตอน `print`

## 6 คลาส Stack FILO

```
class Stack:
    def __init__(self, size):
        self._top = -1
        self._data = [None] * size
        self._size = size
    def isFull(self):
        if (self._top + 1) == self._size:
            return True
        else:
            return False
    def push(self, item):
        if self.isFull() == True:
            print("Stack is full : ", item)
        else:
            self._top += 1
            self._data[self._top] = item
            print("Pushed element: ", item)
    def isEmpty(self):
        if self._top == -1:
            return True
        else:
            return False
    def pop(self):
        if (self.isEmpty() == False):
            item = self._data[self._top]
            self._top -= 1
            return item
        else:
            print("Stack is empty")
            return -1
```

**อย่าลืม** `push` พิมพ์ `"Pushed element: "` ทุกครั้งที่ใส่สำเร็จ

## 6 ผลการรัน Stack ออกสอบ

`s1 = Stack(3)` · push 5, 6, 1, 4 · pop 4 ครั้ง

```
Pushed element:  5
Pushed element:  6
Pushed element:  1
Stack is full :  4
poped:  1
poped:  6
poped:  5
Stack is empty
```

Stack ขนาด 3 · ค่า 4 ถูกปฏิเสธ · pop ได้ **1 → 6 → 5** แล้วครั้งที่ 4 ว่างจึงคืน –1

## 6 การประยุกต์ใช้ Stack

- จดจำย้อนกลับ Backtracking เช่น undo
- ใน compiler ส่งค่าไปยังโปรแกรมย่อย
- คำนวณพจน์คณิตศาสตร์ `(1 + 2) * 3`
- เรียงอักษรย้อนกลับ **THAI → IAHT**

**คำนวณพจน์ ใช้ Stack 2 อัน** Operand เก็บตัวเลข / Operator เก็บเครื่องหมาย · เจอ `(` ไม่ทำอะไร · เจอ `)` pop เลข 2 ตัวกับเครื่องหมาย 1 ตัว คำนวณแล้ว push กลับ · `( (4 + 2 ) * 3 )` = **18**

## 7 คลาส Node

```
class Node():
    def __init__(self, datum):
        self.data = datum
        self.next = None
    def __str__(self):
        return str(self.data)
```

โหนด = **ข้อมูล + ตัวชี้ next** ตัวท้ายชี้ `None` · `data` กับ `next` ไม่มีขีดล่างนำหน้า คือเป็น public

## 7 LinkedList · การเพิ่มโหนด

```
class LinkedList():
    def __init__(self):
        self._head = None
        self._tail = None
        self._size = 0
    def insertAtTail(self, datum):
        new_node = Node(datum)
        if self._tail == None:
            self._tail = new_node
            self._head = new_node
        else:
            self._tail.next = new_node
            self._tail = self._tail.next
    def insertAtHead(self, datum):
        new_node = Node(datum)
        if self._head == None:
            self._tail = new_node
            self._head = new_node
        else:
            new_node.next = self._head
            self._head = new_node
```

มี `_tail` เก็บไว้ `insertAtTail` จึงไม่ต้องไล่ทั้งรายการ

## 7 removeAtHead และ _str_

```
    def removeAtHead(self):
        if (self._head != None):
            datum = self._head.data
            self._head = self._head.next
            return datum
        else:
            print("List is empty")
            return None
    def __str__(self):
        curr = self._head
        i = 1
        s = ""
        while curr != None:
            s = s + "node : " + str(i) \
                + " " + str(curr) + "\n"
            curr = curr.next
            i += 1
        return(s)

mylist.insertAtTail(["65-1", "Mark"])
print(mylist)
node : 1 ['65-1', 'Mark']
```

## 7 Queue FIFO

```
from collections import deque

class Queue:
    def __init__(self):
        self.items = deque()
    def is_empty(self):
        return len(self.items) == 0
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        if self.is_empty():
            raise IndexError(
                "Queue is empty")
        return self.items.popleft()
    def size(self):  return len(self.items)
```

ใส่หลัง `append` · ออกหน้า `popleft` · **สไลด์เรียก `is_empty()` แต่ไม่ได้เขียนไว้** ต้องเขียนเพิ่มเอง

## 7 Circular และ Doubly

**Circular** ท้ายชี้กลับหัว · **Doubly** มี `dummy` จำหัว-ท้าย และมี `prev` เพิ่ม

```
# ใส่ก่อนใหม่ด้านหน้า Doubly
1. new_node.prev = header
2. new_node.next = header.next
3. header.next.prev = new_node
4. header.next = new_node
```

### ! กับดัก 6-7

- `insertAtHead` ต้อง `new_node.next = self._head` **ก่อน** แล้วจึง `self._head = new_node`
- LinkedList ใช้ `_head` `_tail` มีขีดล่าง แต่ Node ใช้ `data` `next` ไม่มีขีดล่าง
- ไล่รายการใช้ `curr` เดิน ห้ามเลื่อน `self._head`