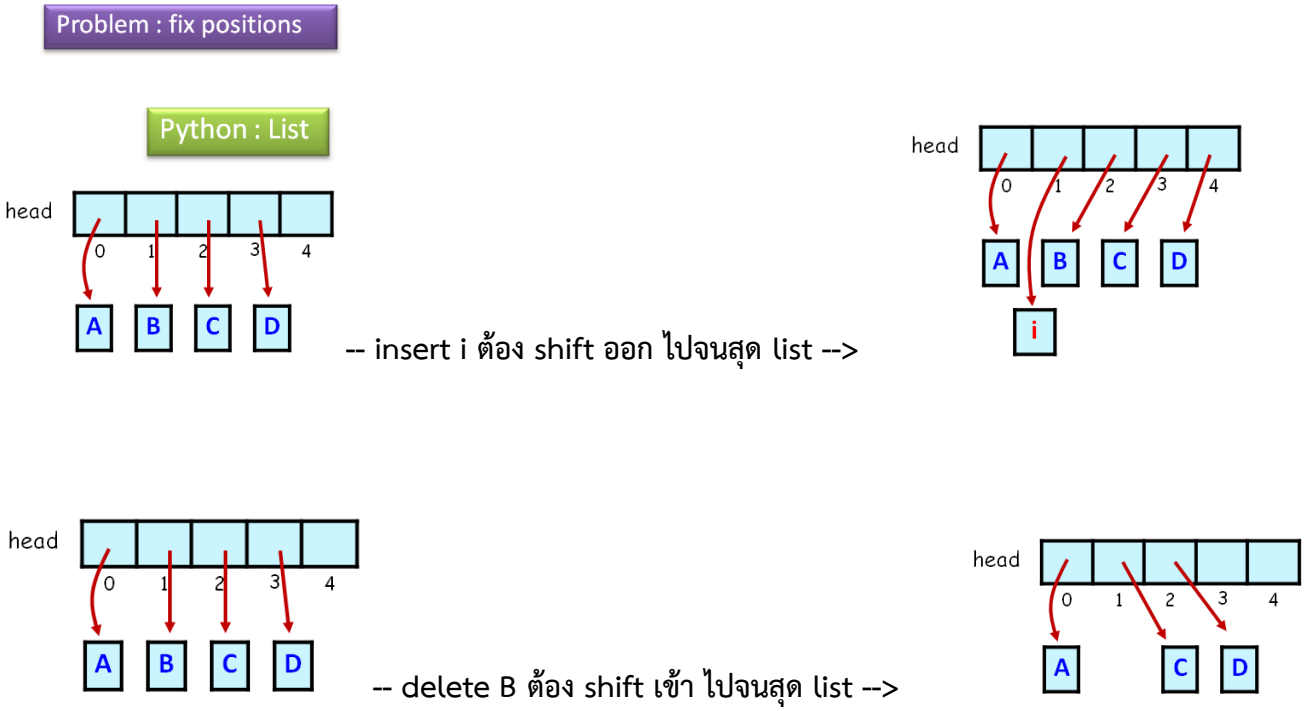
List 2 - 3 Python : Linked List 1 - 2

(lab นี้ ทำ 2 ครั้ง)

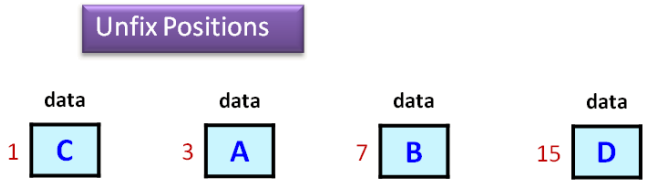
วัตถุประสงค์ ศึกษาเรื่อง Listed List และ การเขียน class บน Python

1. ทฤษฎี : Listed List



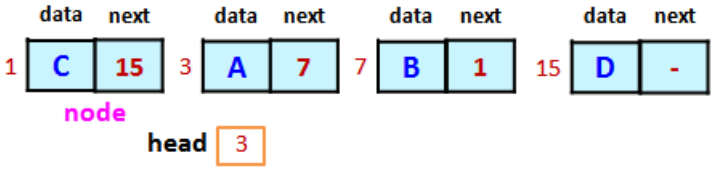
List ที่ใช้โครงสร้างแบบ implicit array (sequential array) หรือ ใช้ Python List ซึ่งโครงสร้างภายในเป็น implicit array of pointer เมื่อต้อง insert จะต้อง shift out ไปจนสุด list ดังนั้นยิ่ง insert ต้น list มากเท่าไหร่ ก็ยิ่งทำงานมากขึ้น เช่นเดียวกับการ delete จะต้อง shift in ไปจนสุด list (สามารถดูภาพเคลื่อนไหวได้ใน lecture note)

ดังนั้น data structure แบบนี้ไม่มีประสิทธิ์ภาพ inefficient สาเหตุคือ ตำแหน่งของแต่ละ element ใน array fix (ตัวที่ 0 1 2 ... อยู่ติดกันใน physical memory) การแก้ไขจึงต้องแก้ไม่ให้ตำแหน่งของแต่ละ element อยู่ติดกัน

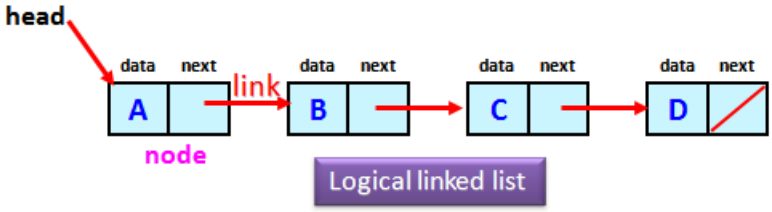


รูปข้างบนแสดงถึงโครงสร้างที่ element แต่ละตัวใน list อยู่ไม่ติดกันใน memory คือ A อยู่ที่ตำแหน่ง 3 ส่วน B C และ D อยู่ที่ตำแหน่ง 7 1 และ 15 ตามลำดับ คำถามคือ แต่ละ element จะเรียงลำดับอย่างไรจึงจะได้ว่า A อยู่ตัวแรก B C และ D อยู่ถัดมาเป็นลำดับ ดังนั้น นอกจากเก็บ data แล้ว เรายังต้องเก็บว่า element ถัดไปอยู่ที่ตำแหน่งใด

และมีตัวเก็บตำแหน่งของ element ตัวแรกไว้ ดังแสดงในรูปข้างล่าง เรียกแต่ละ element ว่า node ซึ่งเก็บ data และตำแหน่งของ next element



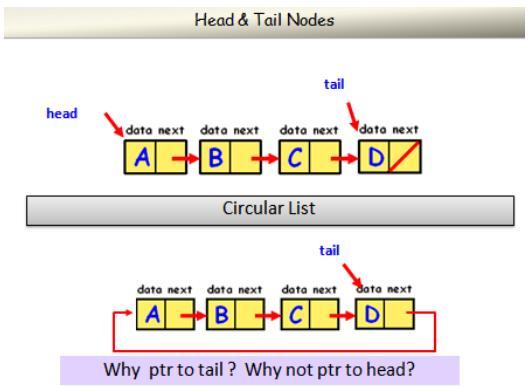
หากเราเขียนตำแหน่งโดยใช้ลูกศร จะได้รูปข้างล่าง ลูกศรแสดงว่า node ถัดไปอยู่ที่ใด จึงเรียกว่า link head แสดงตำแหน่งต้น list โครงสร้างแบบนี้จึงเรียกว่า linked list คือ list ที่ถูกต่อกันด้วย link

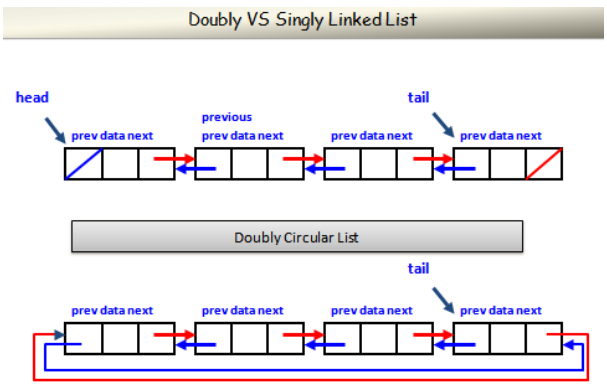


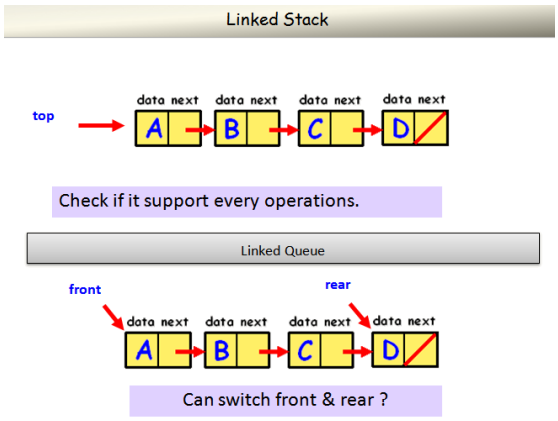
ดังนั้นการสร้าง linked list จึงต้องมีส่วนประกอบของ object 2 classes คือ class node และ class (linked) list

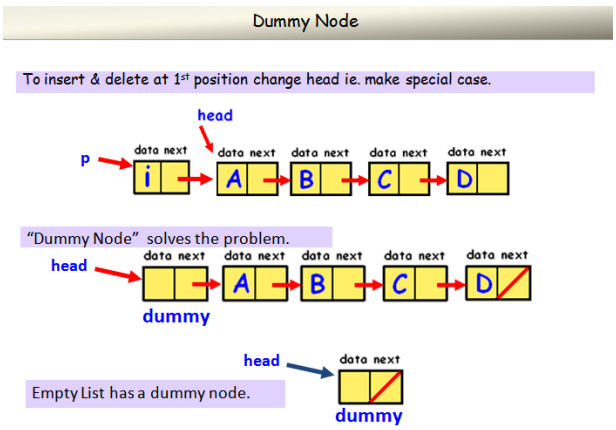
data structure ของแต่ละ class ขึ้นกับแต่ละ application ว่าต้องการเก็บอะไรบ้าง เช่น อาจเก็บเฉพาะ head หรือ เก็บทั้ง head และ tail หรือ อาจเก็บ size ของ list ด้วย ส่วน link อาจเป็นทางเดียว เรียกว่า singly linked list หรือ 2 ทาง เรียกว่า doubly linked list อาจเป็นโครงสร้าง วนกลับ circular ดังแสดงในรูปข้างล่าง data ก็อาจเก็บแบบเรียงลำดับ (เรียกว่า ordered list) หรือ ไม่เรียงลำดับ (unordered list) stack และ queue เป็น subset ของ list ก็สามารถทำโครงสร้างแบบ linked stack และ linked queue ได้

ส่วน methods ต่างๆ ต้องเปลี่ยนแปลงตามโครงสร้างของ data structure (รายละเอียด ดังที่ได้สอนในชั่วโมง lecture) ดังนั้นจึงไม่อาจสอนได้ทั้งหมด นักศึกษาต้อง implement ตามโครงสร้างของ data structure ตามแต่ละ application





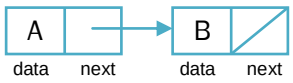




2. **การทดลอง 1** : สร้าง class node และ class (linked) list

เนื่องจากในการทดลองครั้งถัดไปจะต้องใช้ unordered singly linked list ที่ต้องใช้ head , tail และ size ดังนั้นให้นักศึกษา สร้าง class list ที่มี data ทั้ง 3 นี้ การสร้าง class node และ class list ให้ฝึกเขียน methods ต่างๆ ให้ครบ

2.1. เขียน class node และ methods ต่างๆ เมื่อเสร็จแล้ว ทดสอบ

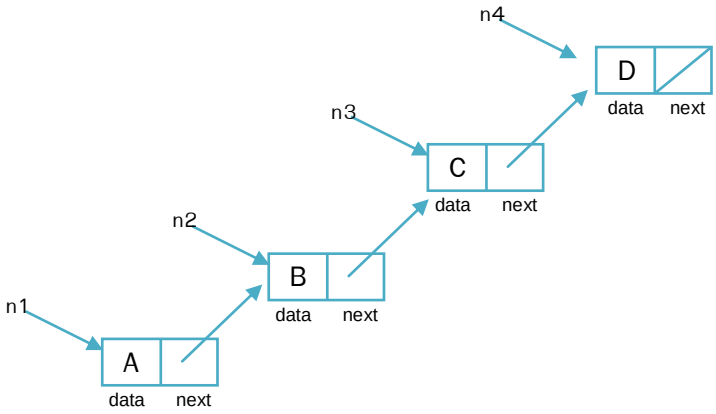


instant data : กำหนดให้ node มี 2 fields ชื่อ data และ next เพื่อเก็บ data และ next node รูปข้างต้นแสดง 2 nodes node ที่ 2 มี data เป็น B และ next เป็น None ซึ่งต่อไปนี้จะเขียนขีดดังแสดง ส่วนอีก node มี data เป็น A และ next เป็น node B

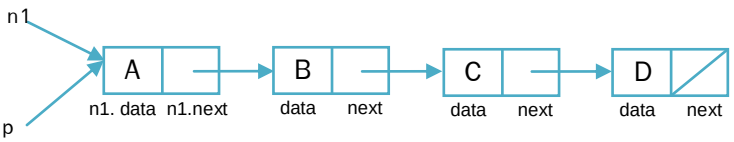
methods : เขียน methods ต่อไปนี้ตามกำหนด

1. `def __init__(self, data, next = None):` เพื่อ instantiate new node ที่มี field data และ next ตาม parameter ที่ pass เข้าไป โดย next default parameter เป็น None ตัวอย่างแสดงในรูป

```
n4 = node('D')
n3 = node('C', n4)
n2 = node('B', n3)
n1 = node('A', n2)
```



2. `def __str__(self):` return string ของ data ของ node



p = n1

ในรูป n1 reference (ชี้ไปที่, เก็บ address ของ) node A หากเราต้องการให้ตัวแปรใดชี้ไปที่ node ให้ดูว่า node นั้นมีตัวแปรอะไรชี้อยู่ ให้ assign ให้ p = ตัวแปรตัวนั้น เช่น ต้องการให้ p ชี้ไปที่ node A ซึ่งมี n1 ชี้อยู่ p = n1 จะทำให้ p ชี้ที่ n1 เช่นเดียวกัน ดังแสดงในรูป

เมื่อตัวแปรใด reference (ชี้ไปที่, เก็บ address ของ) object ใด data ต่างๆของ object นั้นสามารถ access ผ่านทางตัวแปรนั้นได้โดยใช้ ตัวแปร.data ดังนั้น ในรูป

print(n1.data) หรือ print(p.data) จะได้ output A และ

n1.next หรือ p.next หมายถึง field next ของ node A ได้ทั้งคู่ เพราะตอนนี้ทั้ง n1 และ p reference ที่ node A

code ข้างล่างวน loop พิมพ์ node แต่ละ node ได้ output เป็น A B C D

```
p = n1
while p != None:
    print(p.data, end = ' ')      # output เป็น A B C D
    p = p.next
```

เพื่อความสะดวก ให้เขียนฟังก์ชั่น `def __str__(self)` : เพื่อให้สามารถพิมพ์ print(n1) ได้ output A ดังนั้น code ข้างล่างให้ output ดังแสดง

```
q = node('A')
print(q)              output : A
r = node('B', q)
print(r)              output : B
print(r.next)         output : A
```

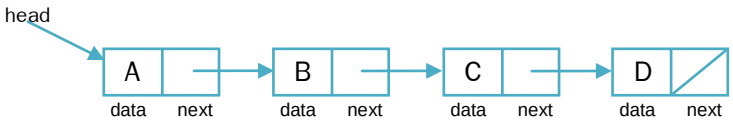
3. `def getData(self):` `#accessor` เขียนไม่ได้ดูใน lecture power point
4. `def getNext(self):` `#accessor`
5. `def setData(self, data):` `#accessor`
6. `def setNext(self, next):` `#accessor`

2.2. เขียน class list และ methods ต่างๆ เมื่อเสร็จแล้ว ทดสอบ

instant data : กำหนดให้ list มี data อย่างน้อยคือ head นักศึกษาอาจเลือกให้มี data อื่นด้วย เช่น tail size ...

methods : เขียน methods ต่อไปนี้ตามกำหนด ดู code ตัวอย่างใน power point lecture

1. `def __init__(self, head = None):` อาจ ใส่ค่า head ให้ชี้ไป list อื่นที่มีอยู่แล้วได้
2. `def __str__(self):` return string แสดง list



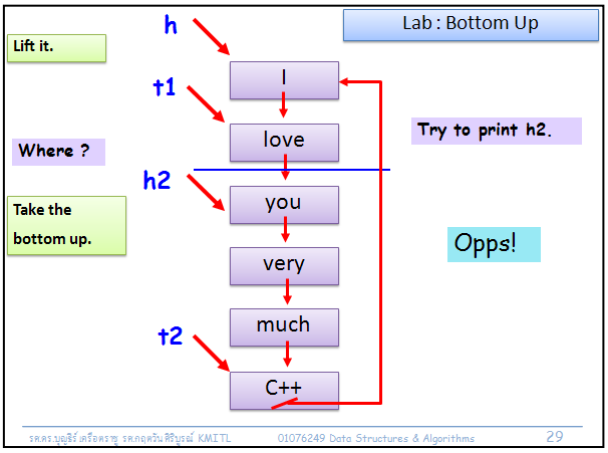
เช่น list l ในรูป เมื่อ print(l) อาจให้ output : size 4 : A B C D

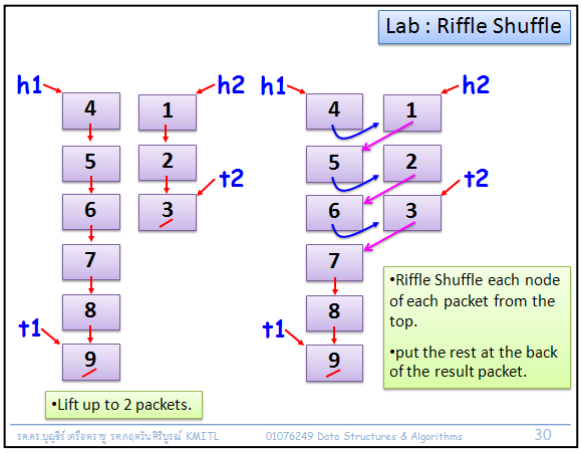
3. `def size(self):` return size ของ list
4. `def isEmpty(self):` return ว่า list empty ใช่หรือไม่
5. `def append(self, data):` add node ที่มี data เป็น parameter ข้างท้าย list
6. `def addHead(self, data):` add node ที่มี data เป็น parameter ข้างต้น list
7. `def isIn(self, data):` เช็คว่ามี data อยู่ใน list หรือไม่
8. `def before(self, data):` return reference ของ node ก่อน node ที่มี data ที่ให้
9. `def remove(self, data):` remove & return node ที่มี data (ดังนั้นข้อนี้จึงต้องใช้ข้อที่แล้ว)
10. `def removeTail(self):` remove & return node สุดท้าย
11. `def removeHead(self):` remove & return node แรก
12.

3. การทดลอง 2 : เขียนโปรแกรม คลุกคำ

เขียนโปรแกรมคลุกคำ (scramble) สร้าง singly linked list ของคำในจดหมาย scramble จดหมายโดยทำคล้ายตัดไพ่และกรีดไพ่ ผู้รับจดหมาย descramble กรีดกลับและตัดกลับจนได้จดหมายฉบับเดิมที่อ่านได้ (หากออกแบบดีๆ สามารถ scramble กี่ครั้งก็ได้ ขึ้นแรกให้ทำ ครั้งเดียวก่อน)

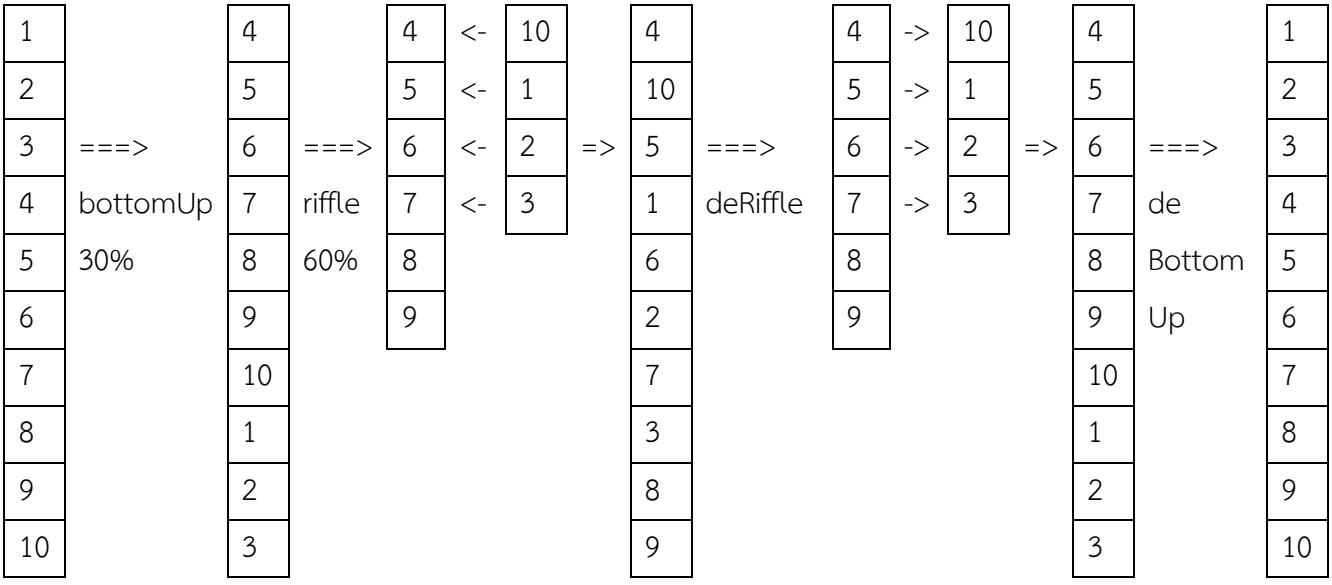
1. เพื่อง่ายต่อการทดสอบ ขั้นแรกให้สร้าง singly linked list ของ int 1 ถึง 10





2. bottomUp ตัด : ยกส่วนบน (lift) ออกตาม % input ที่รับเข้ามา นำส่วนล่างมาซ้อนทับส่วนบน รูปซ้ายแสดงลิสต์ของจดหมาย h, t2 คือหัวและท้ายของลิสต์ก่อนทำ h2, t1 คือหัวและท้ายของลิสต์หลังทำ เขียน bottomUp และ lift เป็นฟังก์ชั่น ออกแบบการเรียก-การส่ง parameter-return type printList ทดสอบ
3. riffleShuffle กรีด (จากด้านบน) : lift ตาม % นำ node ของแต่ละลิสต์มาสลับกันทีละ node จากต้นลิสต์ ส่วนเกินนำมาต่อท้าย รูปขวาข้างขวา h1, t1 คือผลลัพธ์ ห้าม สร้าง node ใหม่ ให้ย้ายค่า link ทดสอบ % หลายๆ แบบ
4. ทำกลับ deRiffle (กรีดกลับ) แล้ว deBottomUp (ตัดกลับ) การออกแบบที่ดีทำให้สามารถ bottomUp และ riffle ได้หลายๆ หน

ตัวอย่าง:



5. เปลี่ยนเป็น list ของ string