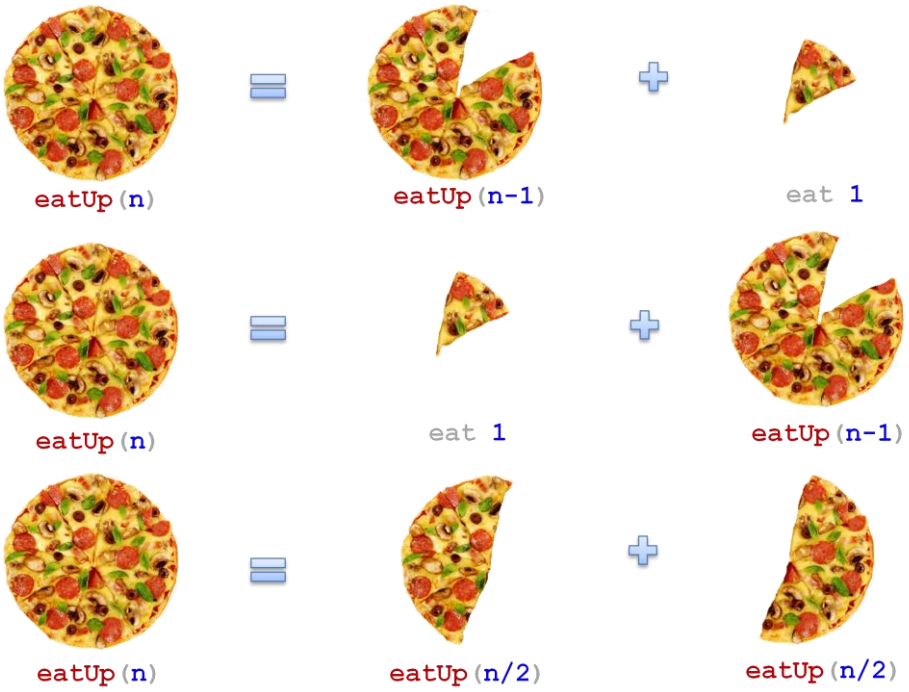
Lab 6-7 : Recursion

วัตถุประสงค์ เข้าใจเรื่อง recursion ทดลองเขียนโปรแกรม recursion knapsack problem

ทฤษฎี

1. **Iteration** : การวนทำซ้ำโดยใช้ loop statements : while, for, ... ใดๆ iterate = repeat ทำซ้ำ
2. **Recursion** : การใช้ปัญหาแบบเดิมที่เล็กลงแก้ปัญหา เช่น การ กินข้าว(ให้หมดจาน n ส่วน) จะทำได้เมื่อ กินข้าว (n-1 ส่วน) ได้ และ กินเพิ่มอีก 1 คำ

การแตกปัญหาให้เล็กลง ทำได้หลายแบบ



จะกิน 3 คำได้ ต้องกิน 2 คำ และ ต้องกิน 1 คำ . . . ถ้าทำไปเรื่อยๆ ไม่หยุดจะเกิด infinite loop การแก้ปัญหาจึงต้องมีกรณีที่ไม่ทำ recursion เรียกว่า **base case** หรือ **simple case**

การเขียน recursion function

1. **ต้องมี parameter**
   มองว่าปัญหาคืออะไร มี parameter อะไร
2. **call recursive case โดยเปลี่ยน parameter**
   แตกปัญหาให้เล็กลง หากแก้ปัญหาที่เล็กกว่านี้ได้ จะต้องทำอะไรเพิ่ม เพื่อให้แก้ปัญหาตั้งต้นได้
3. **หา base case (simple case)** ส่วนมากเกี่ยวข้องกับค่า parameter

* * * การคิด recursion ให้ง่ายขึ้น ไม่ต้องคิดว่า งานที่เล็กลงนั้นต้องทำอย่างไร คิดแค่ว่า หากทำได้ ต้องทำอะไรต่อ ให้งานใหญ่จบ

```
def eat(n):
    if n==1:
        print('eat', n)
    else:
        print('eat', n)   # line 1
        eat(n-1)          # line 2
eat(5)
```

ทุกครั้งที่เรียกฟังก์ชั่น จะมี **stack ของ function** ซึ่งเก็บค่า local variables ของการเรียกครั้งนั้น เมื่อเกิดการย้อนกลับ **backtrack** มาที่เดิม stack จะจดจำสิ่งแวดล้อมเดิมไว้ (ค่า variables ต่างๆ ณ ตอนที่เรียก recursive) ทำให้ recursion เขียน code ได้เร็วและง่ายกว่า iteration ในกรณีมี branch แตกออกไปหลายๆ กิ่ง ซึ่งในกรณีนี้ iteration ใช้เวลาเขียน code นานกว่า เขียนยากกว่า และ debug ยากกว่ามากด้วย ดังนั้นในกรณีเช่นนี้ recursion มีประโยชน์มาก stack ของ recursion เห็นได้ชัดจากตัวอย่างเรื่อง The Eight Queen Problem

การทดลอง 1 อ่านทฤษฎีจาก power point ให้เข้าใจ และลองเขียน function ต่างๆ เอง แบบ recursive (ซึ่งนั่นคือ ห้ามใช้ for และ while) หากทำไม่ได้ค่อยดูเฉลยใน ทฤษฎี หรือ เฉลยด้านหลัง

```
def eat(n):
    if n==1:
        print('eat', n)                      # base/simple case
    else:                                    # recursive case
        print('eat', n, end = ' ')  # line 1
        eat(n-1)                    # line 2
eat(5)
```

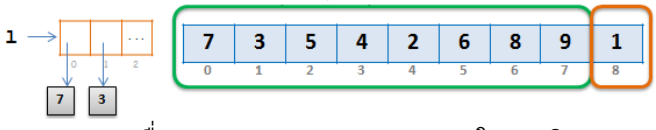
1. เขียนโปรแกรมข้างต้น output จะเป็นอย่างไร และ trace ด้วยมือก่อน ค่อยรันดู output : _____[1]
2. โปรแกรมข้างต้น ลองสลับที่ บรรทัด line 1 และ line 2 คิดว่า output จะเป็นอย่างไร และ รันดูผลลัพธ์ output : _____ นักศึกษาคิดว่าทำไมจึงเป็นเช่นนั้น[2]
3. เขียน recursive `def fac(n)` เพื่อ return ค่า n!
4. เขียน recursive `def sum1ToN(n)` เพื่อ return ค่า sum ตั้งแต่ 1 ถึง n โดย n >= 1
5. เขียน recursive `def printNto1(n)` เพื่อพิมพ์เลข n ถึง 1
6. เขียน recursive `def print1ToN (n)` เพื่อพิมพ์เลข 1 ถึง n
7. เขียน recursive `def fib(n)` เพื่อ return ค่า fibonaci Sequence ลำดับที่ n
8. เขียน recursive `def binarySearch(lo, hi, x, l)` เพื่อ search หา x ใน list l โดย return ค่า index ของ list l ที่มี x และ return None หากหาไม่พบ
9. เขียน recursive `def move(n,'A','C')` เพื่อ พิมพ์ลำดับการเคลื่อน disks แบบ Tower of Hanoi ของ disk n disks จากเสา A to C โดยใช้เสา B เป็นเสาช่วย

---

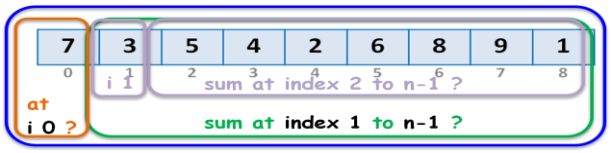
[1] ในครั้งแรก n = 5 line 1 พิมพ์ 5 จึง call recursion ซึ่งพิมพ์ 4 และ การเรียกครั้งต่อๆมา พิมพ์ 3 2 และครั้งสุดท้าย base case พิมพ์ 1

[2] พิมพ์ 1 2 3 4 5 เนื่องจากจะยังไม่พิมพ์จนกว่าจะทำ recursion เสร็จ ดังนั้นจึง recursion ลงไปจนถึง n = 1 พิมพ์ 1 backtrack มาที่ n = 2 พิมพ์ 2 เนื่องจากเป็น stack จึง backtrack ไปที่การ call ครั้งก่อนหน้ามันคือครั้งสุดท้ายบน top ของ stack backtrack ครั้งต่อๆ มาจึงพิมพ์ 5 4 3 ตามลำดับ

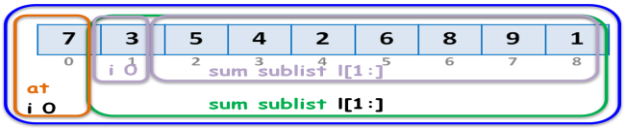
10. เขียน recursive `def sum1(n, l)` เพื่อ return ค่า sum ของ element ใน list l ที่มี size n ตาม algorithm ในรูป (เพื่อความสะดวก ต่อไปนี้จะใช้รูป list ด้านขวา แทน ด้านซ้าย)



11. เขียน recursive `def sum2(....)` เพื่อ sum list ตาม algorithm ในรูป คิด parameter เอง



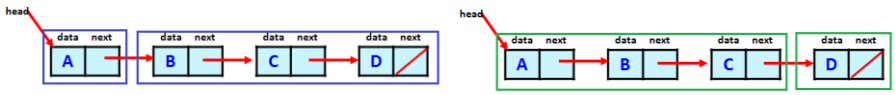
12. เขียน recursive `def sum3(....)` เพื่อ sum list ตาม algorithm ในรูป ใช้ sublist คิด parameter เอง



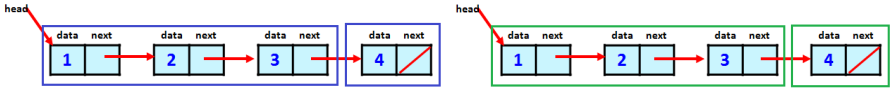
13. เขียน recursive `def printlistForw(...)` และ `def printlistBkw(...)`เพื่อพิมพ์ python list forward และ backward โดยคิด parameter เอง (ทำได้หลายแบบ)
14. เขียน recursive `def app(...)` เพื่อ append 1 ถึง n เข้าไป python list l ตามลำดับ คิด parameter เอง
15. เขียน recursive `def appB(...)` เพื่อ append n ถึง 1 เข้าไป python list l ตามลำดับ คิด parameter เอง
16. สร้าง linked list h ขึ้นมาแบบ iterative เขียน recursive `def printList(h)` เพื่อพิมพ์ data ใน node ของ linked list h โดยกำหนด class node ดังนี้

```
class node():
    def __init__(self, d, nxt = None):
        self.data = d
        if nxt is None:
            self.next = None
        else:
            self.next = nxt
```

17. เขียน recursive `def createLLL(n,..)` เพื่อสร้าง linked list จาก python list คิด parameter เอง ทดสอบโดย `printList(h)` ในรูป สร้างจาก L = ['A', 'B', 'C', 'D'] เพื่อความสะดวก ใช้รูป logical list (ทำได้หลายแบบ เช่น รูปซ้ายและขวา) พิจารณา การเกิดแต่ละ node ให้เข้าใจ



18. เขียน recursive `def createLL (n, ...)` เพื่อสร้าง linked list จาก 1 ถึง n คิด parameter เอง ทดสอบโดย `printList(h)` ในรูป สร้างจาก n = 4 (ทำได้หลายแบบ) พิจารณา การเกิดแต่ละ node ให้เข้าใจ

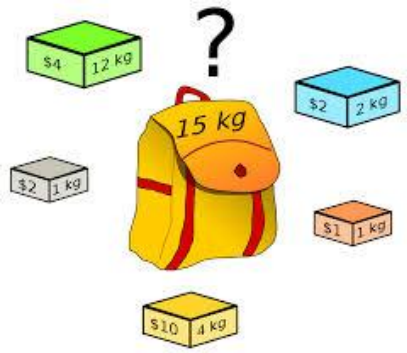


การทดลอง 2 เขียน recursive : knapsack (แนปเสค) problem

มีเงิน k บาท มีของ n ชิ้นราคา b1, b2, ..., bn จะซื้อของให้เงินหมดพอดี (ไม่เหลือและไม่ขาด) ได้หรือไม่ ถ้าได้ ได้ราคาเท่าใดบ้าง (หาทุกค่าที่ซื้อได้)

นั่นคือ B = { b1, b2, ..., bn } เป็น set ของ integers ที่มีค่าซ้ำได้ มี subset ของ B หรือไม่ที่ sum ของสมาชิกทั้งหมด = k พอดี ของราคาเท่ากันถือเป็นคนละอันกัน ดังนั้น

k = 20, B = {20, 10, 5, 5, 3, 2, 20, 10} outputs จะได้ 9 แบบ ดังแสดง



outputs:

20

10 5 5

10 5 3 2

10 5 3 2

10 10

5 5 10

5 3 2 10

5 3 2 10

20

1. จะใช้ data structure อะไรเก็บ **1.** ของ(ราคา)ทั้งหมดในตลาด **2.** ของ(ราคา)ใน sack
2. recursive part คืออะไร ==> โจทย์คืออะไร อะไรที่ทำซ้ำๆ กัน เพียงเปลี่ยนพารามิเตอร์ไป
3. simple part (base part) คืออะไร ==> เราจะหยุดซื้อของเมื่อไรบ้าง ?

เฉลยการทดลอง 1 บางส่วนให้ดูใน lecture

```
def printNdown(n):
    if n > 0:
        print(n, end = ' ')
        printNdown(n-1)
def printToN(n):
    if n > 0:
        printToN(n-1)
        print(n, end = ' ')
def sum1ToN(n): #n>=1
    if n == 1:
        return 1
    else:
       return n + sum1ToN(n-1)
n = 10
printNdown(n)
printToN(n)
print('\nsum1ToN(', n, '):',sum1ToN(n))
#--------------------
def printForw( L, i ): # print list forward
    if i < len(L):
        print(L[i], end = ' ' )
        printForw( L, i+1 )
    else:
        print()
def printBack( L, i ): # print list backward
    if i < len(L):
        printBack( L, i+1 )
        print(L[i], end = ' ' )
    else:
        print()
L = [ 2, 3, 5, 7, 11 ]
print(L)
printForw(L,0)
printBack(L,0)
#-------------------------
def printlistForw(l, n):
    if n > 1:
        printlistForw(l, n-1)
        print(l[n-1], end = ' ')
    elif n == 1:
        print(l[0], end = ' ')
l2 = [1,2,3]
```

```
#-------------------------
def appB(l, n):
    if n == 1:
        l.append(1)
    else:
       l.append(n)
       app(l, n-1)
l = [0]
appB(l, 5)
print(l)
#--------------create linked list -------------
class node():
    def __init__(self, d, nxt = None):
        self.data = d
        if nxt is None:
            self.next = None
        else:
            self.next = nxt

def printList(h):
    if h is not None:
        print(h.data, end = ' ')
        printList(h.next)

#-------------------------
def createLLL1(h, i): #create linked list from
list 1
    global fromList
    if i >= 0:
        last = node(fromList[i], h)
        p = createLLL1(last, i-1)
        return p
    else:
        return h

fromList = [2,5,4,8,6,7,3,1]
print('---- createLLL1 -----' )
h = createLLL1(None, len(fromList)-1) # 2nd para
= last index
printList(h)
print('----------------')
```

```
print(l2)
printlistForw(l2,len(l2))
#-------------------------
def printlistBkw(l, n):
    if n > 1:
        print(l[n-1], end = ' ')
        printlistBkw(l, n-1)
    elif n == 1:
        print(l[0], end = ' ')
l2 = [1,2,3]
print(l2)
printlistBkw(l2,len(l2))
#------------Append--------
def app(l, n):
    if n == 1:
        l.append(1)
    else:
        app(l, n-1)
        l.append(n)
l = [0]
app(l, 5)
print(l)
```

```
#-------------------------
def createLLL2(h, l): #create linked list from
list 2
    if l != []:
        p = node(l[-1], h)
        p = createLLL2(p, l[:-1])
        return p
    else:
        return h
list = [2,5,4,8,6,7,3,1]
print('----- createLLL2  ------')
h = createLLL2(None,list)
printList(h)
print('----------------')
#-------------------------
```

```
def createLLL3(i, last_i, l): #create
linked list from list 3
    if i is last_i:
        return node(l[i])
    else:
        h2 = createLLL3(i+1, last_i, l)
        h = node(l[i], h2)
        return h

print('----- createLLL3  ------')
list = [2,5,4,8,6,7,3,1]
h = createLLL3(0, len(list)-1, list)
printList(h)
print()
#-------------------------
```

```
#-------------------------
def createLL1(st,n): #create linked list 1 to n
    if st is n:
        return node(n)
    else:
        last = createLL1(st+1, n)
        l1 = node(st, last)
        return l1

h = createLL1(1,5)
print('-----createLL 1-----')
printList(h)
print()
#-------------------------
def createLL2(n,back): #create linked list 1 to
n
    if n is 1:
        return node(1,back)
    else:
        l2 = node(n,back)
        l1 = createLL2(n-1, l2)
        return l1

h = createLL2(5,None)
print('-----createLL 2-----')
printList(h)
print()
#-------------------------
```

ตัวอย่าง การทดลองที่ 2 แนปแสค knapsack problem

```
def printSack(sack, maxi):
    global good
    global name
    for i in range (maxi+1):
        print(good[sack[i]], end = ' ')
        # print(name[sack[i]],good[sack[i]], end = ' ')
    print()

def pick(sack, i, mLeft, ig):
    global N
    global good
    if ig < N:    # have something left to pick
        price = good[ig]    # good-price
        if mLeft < price:   # cannot afford that ig
            pick(sack, i, mLeft, ig+1) # try to pick next good
        else:               # can buy
            mLeft -= price  # pay
            sack[i] = ig    # pick that ig to the sack at i
            if mLeft == 0:  # done
                printSack(sack, i)
            else:           # still have moneyLeft
                pick(sack, i+1, mLeft, ig+1)
            pick(sack, i, mLeft+price, ig+1)  # take the item off the sack for other solutions

good = [20,10,5,5,3,2,20,10]
name = ['soap', 'potato chips', 'loly pop', 'toffy', 'pencil', 'rubber', 'milk','cookie']
N = len(good)   # numbers of good

sack = N*[-1]   # empty sack
mLeft = 20      # money left
i = 0           # sack index
ig = 0          # good index
pick(sack, i, mLeft,ig)
```