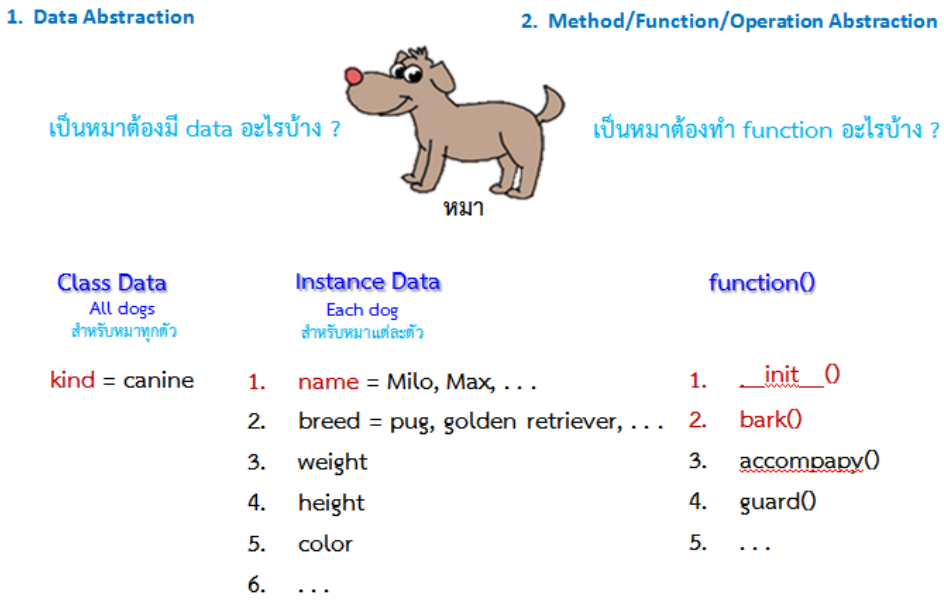
Stack Python

วัตถุประสงค์

1. ศึกษาเรื่อง stack และ การเขียน class บน Python
2. เขียน class stack
3. ใช้ Python เขียน application เกี่ยวกับ stack

1. ทฤษฎี : การสร้าง Class, Stack & Stack Implementation

1.1. Class



เราสามารถสร้าง class ได้จากการ abstract เฉพาะส่วนที่ต้องการของ object ใน 2 ส่วน คือ data และ methods

ส่วน data :

1. หากเป็น data ที่มีใน object ทุก instance เรียกว่า class data มี copy เดียว ใช้ร่วมกันทุก instance
2. data ที่เป็นของเฉพาะสำหรับแต่ละ instance เรียกว่า instance data

ส่วน Methods :

เป็นส่วนของ ฟังก์ชั่นทั้งหลาย ในที่นี้จะมีฟังก์ชั่น __init__() เป็น constructor ปกติใช้ให้ค่าตั้งต้นของ instance ดังนั้น จึงใช้กำหนด instance data

เพื่อให้ง่ายขึ้น จะขอสร้าง class เฉพาะบางส่วน คือ data kind , name และฟังก์ชั่น __init__() และ bark() เท่านั้น

```
class Dog:
    kind = 'canine'    #สำหรับหมาทุกตัว class data shared by all instances

    def __init__(self, name):        # class constructor function()
        self.name = name             # instance data

    def bark(self, num):             # function bark()
        for i in range(num):
            print('Hong', i+1)
```

```
>>> a = Dog('Milo')
>>> b = Dog('Max')
>>> print(a.name, b.name)
Milo Max
>>> print(a.kind, b.kind)
canine canine
>>> a.bark(3)
Hong 1
Hong 2
Hong 3
```

1.2. Stack

Stack คือกองของของ ที่เราเอาของเข้า / ออก ทางด้านบน เรียกว่า top ของ stack การเอาของเข้าไปใน stack เรียกว่า push และ การเอาของออกมาจาก stack เรียกว่า pop จะเห็นว่า ของที่เอาเข้าเป็นอันสุดท้ายถูกเอาออกมาก่อน Last in First out (LIFO)

นอกจากนี้ยังมี operation อื่นๆ อีก ดังแสดงใน ADT (abstract data type) ข้างล่าง

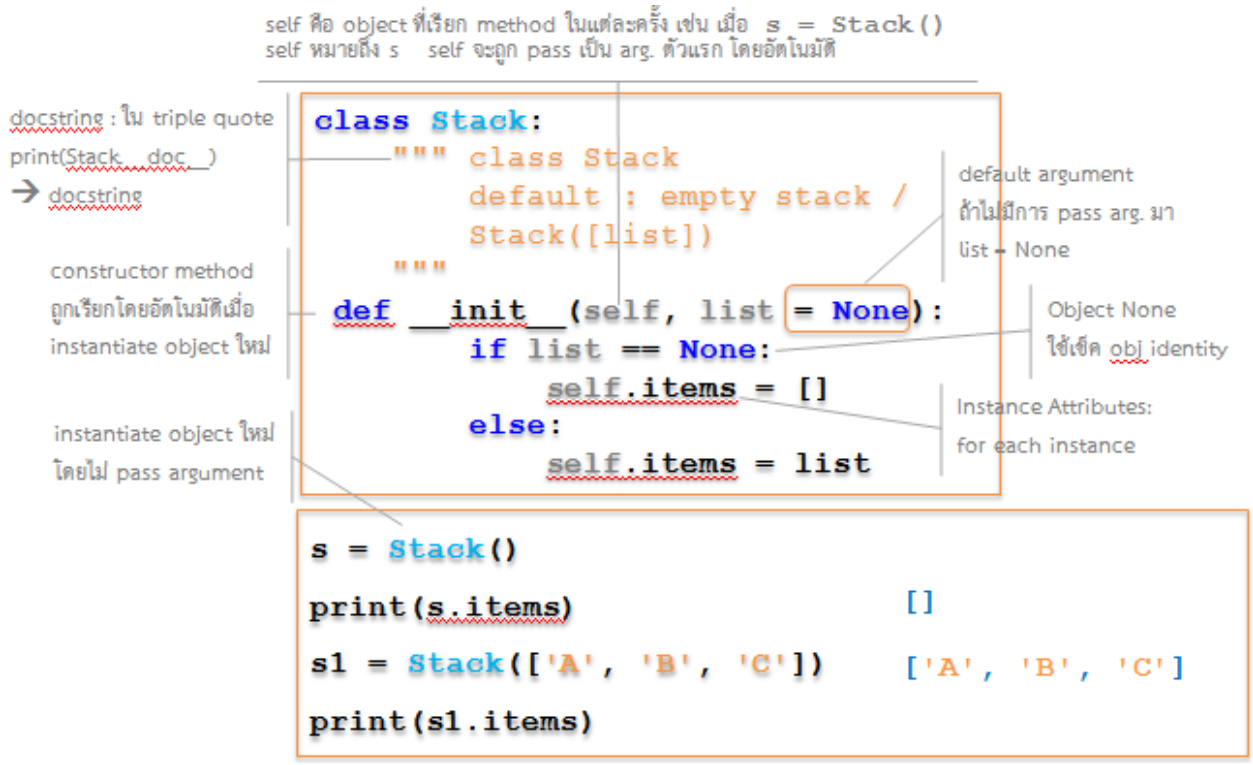
Data : กองของที่มีลำดับ มีปลายด้านบนเพื่อเอาของ เข้า (push) / ออก (pop)	
Methods :	
init()	init empty stack
push(i)	insert i ที่ top
i = pop()	return + เอาของที่ top ออก
i = peek()	return ของที่ top (ไม่เอาออก)
b = isEmpty()	empty ?
b = isFull()	full ?
i = size()	return จำนวนของใน stack

1.3. Stack Data Implementation

เราต้องหา primitive data type ของ Python ซึ่งเก็บของได้หลายอัน และ เอาของเข้าออกที่ปลายด้านบนได้ --> Python List

Python List Type : สัญลักษณ์ []

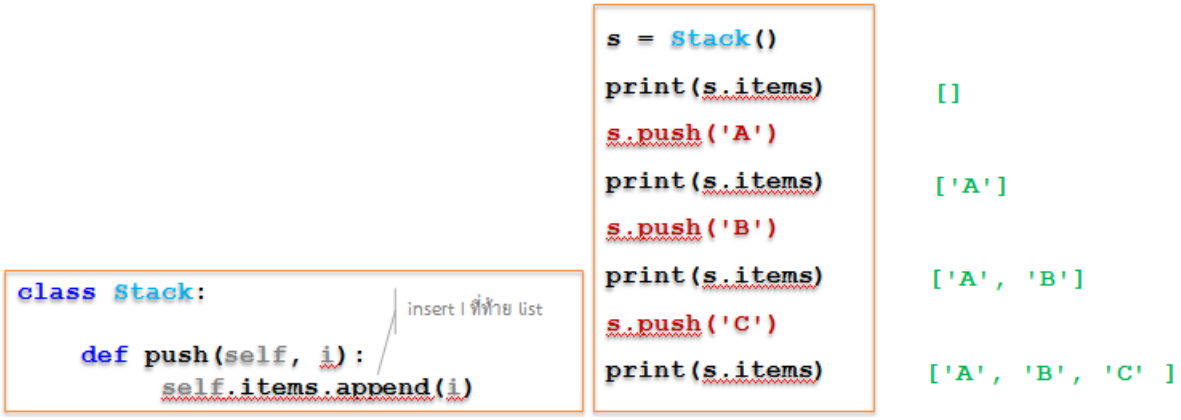
1. เก็บของเรียงลำดับกัน
2. ใส่ของเข้าด้านท้าย append(i)
3. เอาของออกจากท้าย i = pop()
4. i = len(L) returns จำนวนของใน list L



ข้างบนเป็นตัวอย่าง การ implement ส่วน data ของ class stack โดยใช้ Python List

1.4. Stack Method Implementation

1.4.1. push()



2. **การทดลองที่ 1** : Implement Stack Other Methods

2.1. สร้าง class stack โดยอาจเขียนใหม่เอง หรือ ใช้ code ในข้อ 1

เขียน code ทดสอบความถูกต้อง instatiate stack s เป็น empty stack วน loop push() แต่ละ character ของ name = ชื่อนักศึกษา และ print items ของ s ทุก iteration

2.2. เขียน method ของ class stack ในข้อ 1.2 จนครบ พร้อม code ทดสอบ ยกเว้น isFull() (เนื่องจาก Python ขยาย list เมื่อเต็ม)

2.2.1. size() return จำนวนของใน stack

```
def size(self):
    # YOUR CODE
```

code ทดสอบ : ใน loop ข้อ 2.1 print size() ของ stack s ทุก iteration

2.2.2. isEmpty() return True ถ้า stack empty มิฉะนั้น return False

```
def isEmpty(self):
    # YOUR CODE
```

2.2.3. pop() return และ เอาของที่ top ออกจาก stack

```
def pop(self):
    # YOUR CODE
```

code ทดสอบ : loop pop() พร้อม print ของ ที่ pop ออกมา จน stack s empty (ใช้ isEmpty() ในข้อ 2.2.2)

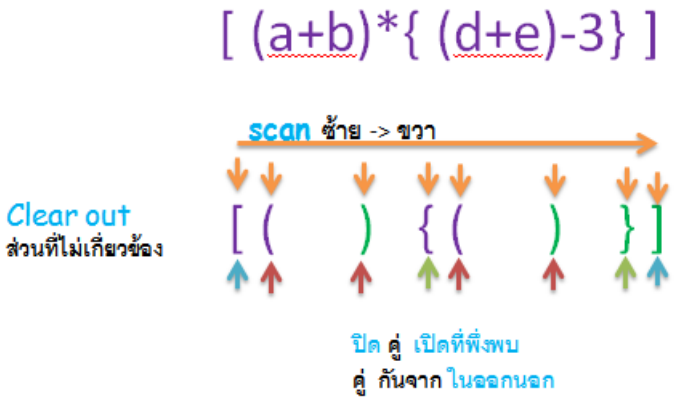
2.2.4. peek() return ของที่ top ของ stack (แต่ไม่ pop ออกมา)

```
def peek(self):
    # YOUR CODE
```

คิดวิธี test เอง

3. **การทดลองที่ 2** : Parenthesis Matching

วงเล็บมีลักษณะเป็น stack เนื่องจาก หากเรา scan จากซ้ายไปขวา วงเล็บปิด คู่กับ วงเล็บเปิดที่พึ่งพบ ดังแสดง



เขียน Python code เพื่อทดสอบ input string ว่ามีวงเล็บถูกต้อง (match) หรือไม่ เช่น

s1 = **(a+b-c *[d+e]/{f*(g+h) }** ไม่ match เพราะ วงเล็บเปิดตัวแรกไม่มีคู่

s2 = **[(a+b-c }*[d+e]/{f*(g+h) }** ไม่ match

s3 = **(3 + 2) / { 4**5 }** **match**

โดยให้ใช้ algorithm ดังนี้

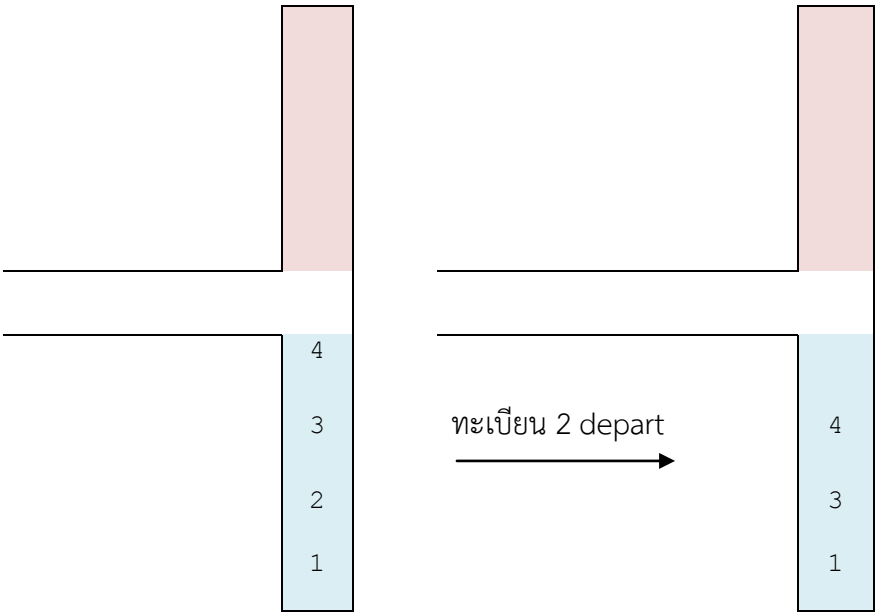
```
str = expression string
s = empty stack
error = False
loop scan each character c from string str until found error or end of string str
    if c is open-parenthesis
        s.push(c)
    else if c is close-parenthesis
        if s is empty
            error = True                # lack of open paren
        else
            open = s.pop()
            if not match(open, c)
                error = True            # open & close paren. not match
if error
    print('MISMATCH')
else
     if stack s is not empty
        print('MISMATCH open paren. exceed')
    else print('MATCH')
```

ลองทดสอบ code ของท่าน และอาจทำให้ดีกว่าเดิมโดยหาว่าผิดแบบไหน

4. **การทดลองที่ 3** : ที่จอดรถแคบ Narrow Parking Lot

ที่จอดรถของนาย ก เป็นส่วนที่แรเงาสีฟ้า ส่วนสีแดงเป็นที่ของนาย ข ซึ่งเป็นญาติกัน ที่จอดรถของนาย ก และ นาย ข แคบมาก จอดรถได้เรียงเดี่ยว นาย ข ไม่ได้ใช้ที่จอดรถ แต่ อนุญาตให้นาย ก ใช้ที่จอดรถของเขาได้โดยไม่จอดรถแช่ไว้ เนื่องจากซอยแคบ ดังนั้นการมาจอด (arrive) และการรับรถ (depart) จะเป็นลักษณะของ stack เงื่อนไขคือ ในการรับรถ x ใดๆ อยากให้ลำดับรถเป็นเหมือนเดิมเพียงแต่ไม่มี x เท่านั้น ดังรูป

simulate การจอดรถในที่จอดรถของนาย ก โดยใช้ operation ของ stack ข้างล่างเป็นตัวอย่าง output ที่จอดรถของนาย ก มีขนาดจำกัด ดังนั้น นศ.อาจเพิ่ม data structure ใน class stack เพื่อแสดงความจำกัดของที่จอดรถ และเพิ่ม method isFull() เพื่อเช็คว่าที่จอดรถเต็มหรือไม่



car 6 cannot depart: soi empty
car 1 arrive space left 3
car 2 arrive space left 2
car 3 arrive space left 1
car 4 arrive space left 0
car 5 cannot arrive : SOI FULL
print soi = [1, 2, 3, 4]
car 7 cannot depart: No car 7
car 2 depart :
pop 4, pop 3, pop 2, push 3, push 4
space left 3
print soi = [1, 3, 4]