Basic Data Structures

วัตถุประสงค์

- เพื่อเข้าใจ abstract data type stack, queue, deque และ list
- สามารถสร้าง ADT stack, queue และ deque โดยใช้ lists ของ Python
- เข้าใจประสิทธิภาพในการทำงานของ data structure เหล่านี้
- เข้าใจ prefix, infix และ postfix expression
- วิเคราะห์ปัญหาแล้วเลือกใช้ stacks, queues และ deques
- สร้าง abstract data type list เป็น linked list และเปรียบเทียบประสิทธิภาพการทำงาน

Linear Structures

- เราจะเริ่มต้นด้วยโครงสร้างข้อมูลพื้นฐาน 4 ตัวที่มีประสิทธิภาพ ได้แก่ stack, queue, deque และ list
- โครงสร้างข้อมูลทั้งสี่ตัวนี้เป็นตัวอย่างของกลุ่มข้อมูลที่ข้อมูลแต่ละตัวจะถูกเรียงลำดับโดยขึ้นกับว่ามันถูกเพิ่มหรือลบอย่างไร
- เมื่อข้อมูลถูกเพิ่ม มันจะอยู่ในตำแหน่งที่สัมพันธ์กับข้อมูลก่อนหน้าและข้อมูลที่มาถัดจากมัน
- กลุ่มข้อมูลจำพวกนี้ถูกเรียกว่า linear data structures

Linear Structures

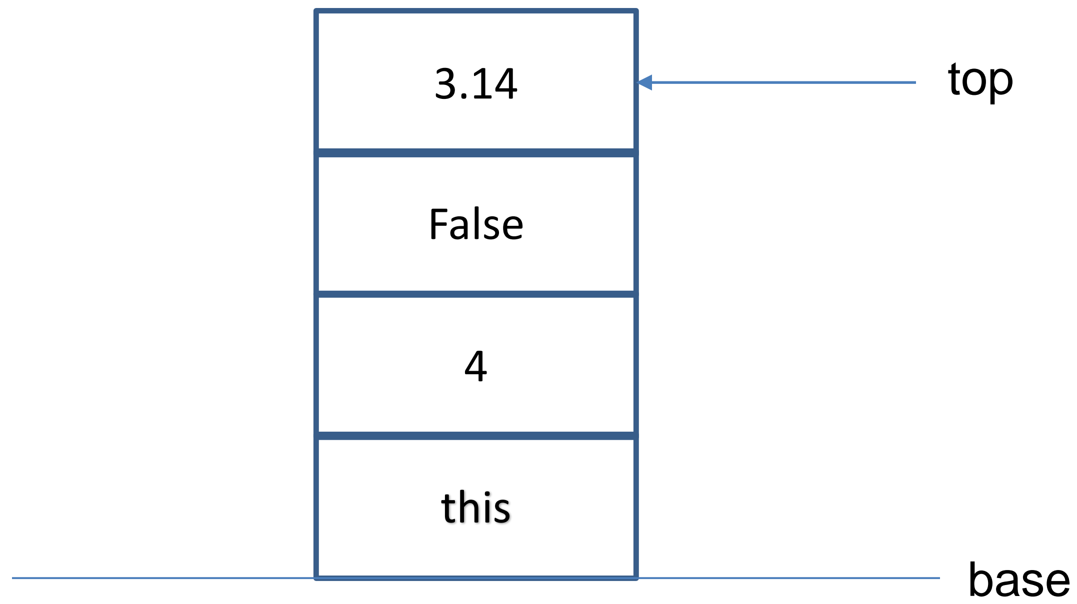
- Linear structures สามารถมองได้ว่ามีจุดปลาย 2 จุด ในบางครั้งเราจะอ้างอิงจุดปลายด้วย left และ right, front และ rear หรือ top และ bottom ทั้งนี้ชื่อเรียกเหล่านี้ไม่ได้สำคัญ แต่เพื่อความเข้าใจตรงกัน
- สิ่งที่ทำให้แต่ละ linear data structure ต่างจากอันอื่นคือเส้นทางที่ข้อมูลถูกเพิ่มหรือเอาออก ตัวอย่างเช่นบางโครงสร้างข้อมูลให้เอาข้อมูลเข้าและออกทางเดียวกัน บางโครงสร้างข้อมูลเข้าและออกคนละทาง

Stacks: Stack คืออะไร

- Stack คือกลุ่มข้อมูลที่เรียงลำดับโดยข้อมูลที่เพิ่มเข้ามาใหม่และข้อมูลที่มีอยู่ที่จะถูกนำออกเกิดขึ้นที่จุดปลายเดียวกัน

- จุดปลายนี้โดยทั่วไปเรียกว่า top จุดปลายฝั่งตรงข้ามกับ top เรียกว่า base โดยที่ base นั้นมีความสำคัญตรงที่ข้อมูลยิ่งอยู่ใกล้ base แสดงว่ายิ่งถูกเก็บใน stack นาน

- ข้อมูลที่ถูกเพิ่มเข้ามาล่าสุดจะถูกนำเอาออกก่อน การเรียงลำดับแบบนี้เรียกว่า LIFO: last-in first-out ซึ่งจะเป็นการเรียงลำดับที่ขึ้นกับเวลาที่ถูกเก็บใน stack
- ข้อมูลใหม่อยู่ใกล้ top ข้อมูลเก่าอยู่ใกล้ base
- ตัวอย่างที่เห็นในชีวิตประจำวัน เช่น แก้วในในร้าน 711 จานสลัดใน sizzler


3.14
False
4
this
top
base

Stack: Stack abstract data type

- Stack abstract data type นิยามด้วยโครงสร้างและการดำเนินการต่อไปนี้
- Stack เป็นโครงสร้างที่กลุ่มข้อมูลถูกเรียงลำดับโดยข้อมูลที่ถูกเพิ่มและเอาออกจากจุดปลายที่เรียกว่า top Stack มีคุณสมบัติแบบ LIFO
- การดำเนินการของ stack มีดังนี้
  - Stack() เป็นการสร้าง stack อันใหม่ที่ว่าง ไม่ต้องการ parameter และคืนค่าเป็น stack ว่าง

- push(item) เพิ่มข้อมูลใหม่ไว้ที่ top ของ stack ต้องการข้อมูลและไม่คืนค่า
- pop() นำข้อมูลที่ top ออกจาก stack ไม่ต้องการ parameter คืนค่าข้อมูล (stack เปลี่ยนแปลง)
- peak() คืนค่าข้อมูลตัวที่ top แต่ไม่นำข้อมูลออก ไม่ต้องการ parameter คืนค่าข้อมูล (stack ไม่เปลี่ยนแปลง)
- is_empty() ทดสอบว่า stack ว่างไหม ไม่ต้องการ parameter คืนค่าข้อมูลเป็นจริงหรือเท็จ
- size() คืนค่าจำนวนข้อมูลของ stack ไม่ต้องการ parameter คืนค่าข้อมูลจำนวนเต็ม

Example stack operations

Stack Operation	Stack Contents	Return Value
s.is_empty()	[]	True
s.push(4)	[4]	
s.push('dog')	[4,'dog']	
s.peek()	[4,'dog']	'dog'
s.push(True)	[4,'dog',True]	
s.size()	[4,'dog',True]	3
s.is_empty()	[4,'dog',True]	False
s.push(8.4)	[4,'dog',True,8.4]	
s.pop()	[4,'dog',True]	8.4
s.pop()	[4,'dog']	True
s.size()	[4,'dog']	2

Implement a stack in Python

- เราได้นิยาม stack ไปแล้วต่อไปเราจะมา implement กัน
- เมื่อเรานำเอา abstract data type ไป implement เราจะเรียกสิ่งที่ implement นั้นว่า data structure
- เนื่องจาก Python เป็น objected-oriented programming language เราก็จะสร้างเป็น class และ stack operations จะเขียนด้วย method (function)
- ใน Python มี list class ซึ่งเป็นที่เก็บกลุ่มข้อมูลและมีการดำเนินการพื้นฐานมาให้ เราจะใช้ list ในการสร้าง stack

- ตัวอย่างถ้าเรามี list [2,5,3,6,7,4] เราเพียงแค่ตัดสินใจว่าปลายด้านไหนของ list ที่เราจะเอามาเป็น top ของ stack และด้านไหนเป็น base ของ stack
- เมื่อเราเลือกได้แล้ว การดำเนินการต่างๆ จะถูกสร้างโดยใช้ function ที่มีให้จาก list เช่น append, pop

Completed implementation of a stack ADT

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
       self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items)-1]
    def size(self):
        return len(self.items)
```

- เมื่อเราสั่ง run(F5) จะยังไม่มีอะไรเกิดขึ้น เนื่องจาก code ข้างต้นเป็นการนิยาม class เหมือนพิมพ์เขียว
- เราจะต้องสร้าง object มาก่อนจึงจะใช้งานได้ โดยการเขียน

```
s = Stack()
```

จากนั้นลองเพิ่ม code ตามตัวอย่างการเรียกใช้งานก่อนหน้า

```python
s = Stack()
print(s.is_empty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print(s.size())
print(s.is_empty())
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

- นอกจากนี้เราสามารถเลือกที่จะสร้าง stack จาก list โดยที่ top เป็นส่วนเริ่มต้นแทนที่จะเป็นส่วนปลายได้
- ในกรณีนี้ เราต้องแก้ไขเนื่องจาก pop และ append ใช้งานไม่ได้ เราต้องจัดการบางอย่างเพื่อที่จะใช้ index ที่ 0 ดังนั้นเราจะใช้ pop และ insert มาช่วย

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def push(self, item):
       self.items.insert(0,item)
    def pop(self):
        return self.items.pop(0)
    def peek(self):
        return self.items[0]
    def size(self):
        return len(self.items)
```

Queue คืออะไร

- Queue คือกลุ่มข้อมูลที่เรียงลำดับโดยข้อมูลที่เพิ่มเข้ามาใหม่จะไปปรากฏอยู่ส่วนท้ายที่เรียกว่า rear และข้อมูลที่มีอยู่ที่จะถูกนำออกเกิดขึ้นที่จุดปลายด้านตรงข้ามที่เรียกว่า front

- ข้อมูลที่เพิ่งถูกเพิ่มเข้าไปใน queue จะต้องรอที่ส่วนท้ายของกลุ่มข้อมูล ข้อมูลที่อยู่ในกลุ่มข้อมูลนานที่สุดจะอยู่ส่วน front
- ลำดับเช่นนี้เรียกว่า FIFO: first-in first-out หรือ first-come first-served ตัวอย่างที่เห็นได้ทั่วไปของ queue คือการเข้าคิวดูหนัง เข้าคิวจ่ายเงิน ซึ่งจะไม่มีการลัดคิวเกิดขึ้น

Queue : Queue abstract data type

- Queue abstract data type นิยามด้วยโครงสร้างและการดำเนินการต่อไปนี้
- Queue เป็นโครงสร้างที่กลุ่มข้อมูลถูกเรียงลำดับโดยข้อมูลที่ถูกเพิ่มจะไปต่อส่วนท้ายที่เรียกว่า rear และข้อมูลที่จะเอาออกจะอยู่ส่วนปลายอีกด้านที่เรียกว่า front Queue มีคุณสมบัติ FIFO

- การดำเนินการของ queue มีดังนี้
  - Queue() เป็นการสร้าง queue อันใหม่ที่ว่าง ไม่ต้องการ parameter และคืนค่าเป็น queue ว่าง
  - enqueue(item) เพิ่มข้อมูลใหม่ให้ส่วน rear ของ queue ต้องการข้อมูลและไม่คืนค่า
  - dequeue() นำข้อมูลที่ front ออกจาก queue ไม่ต้องการ parameter คืนค่าข้อมูล (queue เปลี่ยนแปลง)
  - is_empty() ทดสอบว่า queue ว่างไหม ไม่ต้องการ parameter คืนค่าข้อมูลเป็นจริงหรือเท็จ
  - size() คืนค่าจำนวนข้อมูลของ queue ไม่ต้องการ parameter คืนค่าข้อมูลจำนวนเต็ม

Example queue operations

Queue Operation	Queue Contents	Return Value
q.is_empty()	[]	True
q.enqueue(4)	[4]	
q.enqueue('dog')	['dog',4]	
q.enqueue(True)	[True,'dog',4]	
q.size()	[True,'dog',4]	3
q.is_empty()	[True,'dog',4]	False
q.enqueue(8.4)	[8.4, True,'dog',4]	
q.dequeue()	[8.4, True,'dog',]	4
q.dequeue()	[8.4, True]	'dog'
q.size()	[8.4, True]	2

Implement a queue in Python

- เราก็จะสร้าง class ใหม่สำหรับการ implement abstract data type queue เช่นเดียวกับ stack เราจะใช้ list ในการสร้าง queue
- ทั้งนี้เราต้องตัดสินใจว่าเราจุดปลายของ list ส่วนไหนที่เราจะเอามาเป็น rear และส่วนไหนมาเป็น front
- สมมติว่าเราจะให้ส่วน rear เป็น index 0 ใน list
- ดังนั้นเราจะใช้ insert ที่ตำแหน่ง 0 เวลาที่เพิ่มข้อมูลใหม่
- และใช้ pop เวลาเอาข้อมูลออก

```python
class Queue:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def enqueue(self, item):
       self.items.insert(0,item)
    def dequeue(self):
        return self.items.pop()
    def size(self):
        return len(self.items)
```

- ลองสร้างไฟล์ใหม่ แล้วทดสอบด้วย

```
q = Queue()
q.enqueue('hello')
q.enqueue('dog')
q.enqueue(3)
q.dequeue()
```

Deque คืออะไร

- Deque(อ่านว่า deck) เรียกอีกอย่างว่า double-end queue คือกลุ่มข้อมูลที่เรียงลำดับคล้ายกับ queue มันมีจุดปลาย 2 จุด front และ rear
- สิ่งที่ทำให้ deque ต่างออกไปคือการเพิ่มและลบข้อมูล ข้อมูลใหม่สามารถถูกเพิ่มได้ทั้งด้าน front และ rear เช่นเดียวกัน ข้อมูลสามารถถูกลบได้ทั้งด้าน front และ rear
- ข้อสังเกต deque ไม่มีคุณสมบัติทั้ง LIFO และ FIFO

Deque : Deque abstract data type

- Deque abstract data type นิยามด้วยโครงสร้างและการดำเนินการต่อไปนี้
- Deque เป็นโครงสร้างที่กลุ่มข้อมูลถูกเรียงลำดับโดยข้อมูลสามารถถูกเพิ่มและเอาออกได้ทั้งทาง front และ rear
- การดำเนินการของ deque มีดังนี้
  - deque() เป็นการสร้าง deque อันใหม่ที่ว่าง ไม่ต้องการ parameter และคืนค่าเป็น deque ว่าง
  - add_front(item) เพิ่มข้อมูลใหม่ให้ส่วน front ของ deque ต้องการข้อมูลและไม่คืนค่า

- add_rear(item) เพิ่มข้อมูลใหม่ให้ส่วน rear ของ deque ต้องการข้อมูลและไม่คืนค่า
- remove_front() นำข้อมูลที่ front ออกจาก dequeue ไม่ต้องการ parameter คืนค่าข้อมูล (deque เปลี่ยนแปลง)
- remove_rear() นำข้อมูลที่ rear ออกจาก dequeue ไม่ต้องการ parameter คืนค่าข้อมูล (deque เปลี่ยนแปลง)
- is_empty() ทดสอบว่า deque ว่างไหม ไม่ต้องการ parameter คืนค่าข้อมูลเป็นจริงหรือเท็จ
- size() คืนค่าจำนวนข้อมูลของ deque ไม่ต้องการ parameter คืนค่าข้อมูลจำนวนเต็ม

Example deque operations

Queue Operation	Queue Contents	Return Value
d.is_empty()	[]	True
d.add_rear(4)	[4]	
d.add_rear('dog')	['dog',4]	
d.add_front('cat')	['dog',4,'cat']	
d.add_front(True)	['dog',4,'cat',True]	
d.size()	['dog',4,'cat',True]	4
d.is_empty()	['dog',4,'cat',True]	False
d.add_rear(8.4)	[8.4, 'dog',4,'cat',True]	
d.remove_rear()	['dog',4,'cat',True]	8.4
d.remove_front()	['dog',4,'cat']	True

Implement a deque in Python

- เราก็จะสร้าง class ใหม่สำหรับการ implement abstract data type deque เช่นเดียวกับ stack queue เราจะใช้ list ในการสร้าง deque
- สมมติว่าเราจะให้ส่วน rear เป็น index 0 ใน list

```python
class Deque:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def add_front(self, item):
        self.items.append(item)
    def add_rear(self, item):
        self.items.insert(0, item)
    def remove_front(self):
        return self.items.pop()
    def remove_rear(self):
        return self.items.pop(0)
    def size(self):
        return len(self.items)
```