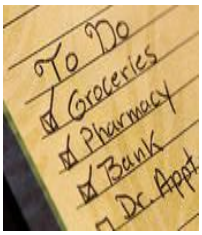
List 1 Python

วัตถุประสงค์ ศึกษาเรื่อง List และ การเขียน class บน Python

1. **ทฤษฎี : List**



List ลิสต์ คือรายการ ที่เราเอาของเข้า และ เอาของออก ที่ตำแหน่งใดก็ได้ ดังนั้น Queue และ Stack เป็น subset ของ List

List Data : เนื่องจากการเอาของเข้าและออก จะทำ ณ ตำแหน่งใดก็ได้ จึงอาจใช้ Python List สร้างได้ แต่เนื่องจากโครงสร้างภายใน Python List เป็น array ของ pointer ดังนั้นจึงทำให้การเข้าหา access ข้อมูล ณ ตำแหน่งอื่นๆ ที่ไม่ใช้ข้างท้าย ต้องทำงานมาก ดังที่ได้เรียนไปในชั่งโมงทฤษฎี ดังนั้นจึงมี data structure แบบอื่นคือ Linked List (ซึ่งจะได้เรียนในชั่วโมงทฤษฎีครั้งหน้า) สำหรับครั้งนี้ให้ใช้ Python List ในการ implement

List Operations สำหรับ list ขึ้นกับ application ที่จะใช้งาน โดยทั่วไปคือ การนำของเข้า นำของออก หาของที่ต้องการ หาจำนวนของในลิสต์ เช็คว่าเป็นลิสต์ว่างหรือไม่ ชื่อของ method ต่างกันไปตาม data structure และภาษาที่ใช้ เช่น python นิยมใช้คำว่า append หมายถึงต่อท้าย แต่หากใช้ list เป็น superset ของ stack ก็อาจสร้าง method pop ซึ่งเป็น operation ที่ทำงานเช่นเดียวกับ append

ข้างล่างเป็นตัวอย่าง methods ของ unordered list (เก็บของไม่เรียงลำดับมาก-น้อย) ในภาษา python จาก text book : problem solving with algorithms and data structures using python ซึ่งครูได้ comment เพิ่มเติมไว้ในส่วนสีม่วงเอน ข้างท้ายคำอธิบาย method

Data : เก็บของที่มีลำดับ เอาของออกที่ตำแหน่งใดก็ได้ใน list

Methods :

`List()` creates a new list that is empty. It needs no parameters and returns an empty list.

`add(item)` adds a new item to the list. It needs the item and returns nothing. Assume the item is not already in the list.

จาก `append(item)` และ `insert(pos,item)` ข้างล่าง add(item) น่าจะ add ที่ตำแหน่งแรกคือต้นลิสต์

`remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.

`search(item)` searches for the item in the list. It needs the item and returns a boolean value.

บางคนใช้ return ตำแหน่งของ item ในลิสต์ ซึ่งในที่นี้ใช้ method index(item) ข้างล่าง

`isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.

`size()` returns the number of items in the list. It needs no parameters and returns an integer.

`append(item)` adds a new item to the end of the list making it the last item in the collection. It needs the item and returns nothing. Assume the item is not already in the list.

`index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.

`insert(pos,item)` adds a new item to the list at position pos. It needs the item and returns nothing. Assume the item is not already in the list and there are enough existing items to have position pos.

`pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.

`pop(pos)` removes and returns the item at position pos. It needs the position and returns the item. Assume the item is in the list.

ข้างล่างเป็นตัวอย่าง methods ของ ordered list (เก็บของเรียงลำดับ) ในภาษา python จาก text book : problem solving with algorithms and data structures using python

Data : เก็บของที่มีลำดับ เอาของออกที่ตำแหน่งใดก็ได้ใน list

Methods :

`OrderedList()` creates a new ordered list that is empty. It needs no parameters and returns an empty list.

`add(item)` adds a new item to the list making sure that the order is preserved. It needs the item and returns nothing. Assume the item is not already in the list.

`remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.

`search(item)` searches for the item in the list. It needs the item and returns a boolean value.

`isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.

`size()` returns the number of items in the list. It needs no parameters and returns an integer.

`index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.

`pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.

`pop(pos)` removes and returns the item at position pos. It needs the position and returns the item. Assume the item is in the list

2. **การทดลอง** : การบวก expression โดยใช้ list

กำหนด expression A และ B เป็น function polynomial ของ n

ต้องการบวก expression ทั้งสอง C = A + B เช่น

$$A = 3n^{95} + n^{7} + 10n^{4}$$
$$B = 21n^{100} - 7n^{95} - 2n^{4} + 15n^{3} + 4n^{2} - n + 1 \quad +$$
$$C = 21n^{100} - 4n^{95} + n^{7} + 8n^{4} + 15n^{3} + 4n^{2} - n + 1$$

ในหัวข้อต่อไปนี้เป็นการบอกวิธีทำตามลำดับ หากนักศึกษาทำเองได้ ให้ทำเองก่อน คิดเองให้มากที่สุด อ่านคำแนะนำทีหลังเพื่อเก็บรายละเอียด

2.1. ลองคิด algorithm ของการบวก (เพื่อหา data structure ที่เหมาะสม) การบวกเราทำอย่าง ? _____
คำตอบ : การบวก expression บวกเฉพาะพจน์ (term) ที่มีกำลัง (power) เท่ากัน โดยเอาสัมประสิทธิ์ (coefficient) มาบวกกัน

2.2. เพื่อให้ข้อที่แล้วทำได้ง่าย เรานิยมเขียน expression เรียงจาก term ที่มีกำลัง มาก ไป น้อย

2.3. จาก 2 ข้อข้างต้น ต้องสร้าง class term และ class orderedList แต่ละ class ต้องทำมี data & method อะไรได้บ้าง

2.4. class term : data & print()

class term ต้องเก็บ data อะไรบ้าง จึงจะใช้ บวกกันได้ ? ในตัวอย่างเราเอา term $3n^{95} - 7n^{95}$ ได้ $-4n^{95}$ เพื่อความสะดวก เขียนฟังก์ชั่น __str__() เพื่อจะได้ print(t) ทดสอบได้ดังเช่นตัวอย่างข้างล่าง

```
t1 = term(-7,95)
print(t1)                    # output -> "-7n95"
t2 = term(3,95)
print(t2)                    # output -> "3n95"
```

2.5. overloading operators : < __lt__()

```
print(t1 < t2)               # output -> false
```

เหมือนกับใน C++ หรือ Java เราสามารถทำ overloading operator สำหรับ class ที่เราสร้างขึ้นได้ คือ overload ความหมายของ operator < less than ให้มีความหมายตาม class object ของเรา โดยฟังก์ชั่นเหล่านี้มีชื่อเฉพาะ เช่น operator < (less than) ใช้ฟังก์ชั่น __lt__() เช่น

```
def __lt__(self, rhs): # less than <
    return self.power < rhs.power
```

ใน class term เพื่อความสะดวก เราควร overload operator + โดยใช้ฟังก์ชั่น __add__() เพื่อให้สามารถใช้เครื่องหมาย + บวก term 2 term ซึ่งจะนำไปใช้ในการบวก expression ได้ ดังแสดงข้างล่าง

```
t3 = t1 + t2
print(t3)                    # output -> "-4n95"
```

นักศึกษาอาจต้อง overload operator อื่นอีก หากตอนนี้ยังคิดไม่ออก ค่อยกลับมาทำเมื่อเขียน class list แล้ว โดยหน้าสุดท้ายเป็นชื่อ overloading ฟังก์ชั่นต่างๆ สามารถดูตัวอย่างการใช้ได้จาก http://www.python-course.eu/python3_magic_methods.php

2.6. ทดสอบ class term

เมื่อเขียน class term เสร็จ ให้ทดสอบความถูกต้องเท่าที่คิด test case ได้

2.7. class orderedList

เขียน class orderedList คล้ายกับที่เราเขียน class stack และ class queue เราจะใช้ python list เป็น data structure โดย __init()__ ให้เป็น list ว่าง ทั้งนี้เพราะเราต้องการให้ของใน list เรียงจากมากไปน้อย การใช้ default argument list อาจไม่ได้เรียงตามที่เราต้องการ

เราจะต้อง overload operator + เพื่อทำการบวก expression ซึ่งเป็น list ของ term 2 list ดังนั้นการเอาของเข้าไปใน list ให้ใช้ method ชื่อ insert ทำการใส่ term เข้าไปทีละ term ไม่ควรใช้ชื่อ add() เพื่อไม่ให้ชื่อสับสนกับ __add__() operator + insert() ต้องทำการ insert ให้เรียงจากน้อยไปมาก

เขียนฟังก์ชั่นอื่นๆที่จำเป็น ให้นักศึกษาคิดเองว่ามีอะไรบ้าง

class orderedList จะต้องใช้กับ object อื่นได้ด้วย ไม่ใช่ใช้กับ term ได้อย่างเดียว เพียงแต่ object class นั้นต้อง overload operator ที่จำเป็น เช่น >, ...

อย่าลืมเขียน __str__() ทำให้ง่ายในการ debug

2.8. ทดสอบ class orderedList

ทดสอบ class orderedList เท่าที่จะนึก test case ได้ เช่น

```
A = orderedList()
A.insert(term(3,95))
A.insert(term(10,4))
A.insert(term(1,7))
print(A)                    +3n95 +n7 +10n4
```

```
B = orderedList()
B.insert(term(-1,1))
B.insert(term(1))
B.insert(term(15,3))
B.insert(term(-2, 4))
B.insert(term(4,2))
B.insert(term(21,100))
B.insert(term(-7,95))
print(B)                    +21n100 -7n95 -2n4 +15n3 +4n2 -n +1

C = A + B
print(C)                    +21n100 -4n95 +n7 +8n4 +15n3 +4n2 -n +1
```

Binary Operators

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__truediv__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)
^	object.__xor__(self, other)
|	object.__or__(self, other)

Extended Assignments

Operator	Method
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other)
**=	object.__ipow__(self, other[, modulo])
<<=	object.__ilshift__(self, other)
>>=	object.__irshift__(self, other)
&=	object.__iand__(self, other)
^=	object.__ixor__(self, other)
|=	object.__ior__(self, other)

Unary Operators

Operator	Method
-	object.__neg__(self)
+	object.__pos__(self)
abs()	object.__abs__(self)
~	object.__invert__(self)
complex()	object.__complex__(self)
int()	object.__int__(self)
long()	object.__long__(self)
float()	object.__float__(self)
oct()	object.__oct__(self)
hex()	object.__hex__(self

Comparison Operators

Operator	Method
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)