Programming Fundamentals I Lab.

5. ลิสต์ และ ดิกชันนารี

ชื่อ_____ รหัสนิสิต_____

ในปฏิบัติการนี้ คุณจะได้รู้จักการใช้งานโครงสร้างข้อมูลพื้นฐานในภาษา Python ได้แก่ลิสต์ (list) และ ดิกชันนารี (dict)

5.1 ลิสต์

เปิด Python Shell แล้วพิมพ์คำสั่งต่อไปนี้

```
>>> a = []
```

หลังจากนั้นให้ลองหาวิธีเพื่อทดสอบว่าตัวแปร `a` มีชนิดข้อมูลเป็นอะไร บันทึกคำตอบ

list เป็นโครงสร้างข้อมูลพื้นฐานประเภทหนึ่งในภาษา Python ให้เราสามารถเพิ่มข้อมูลและดึงข้อมูลออกได้อย่างง่าย โดยใน list จะมีการจัดเก็บข้อมูลเรียงกันเป็นแถว

ทดลองพิมพ์คำสั่งต่อไปนี้ต่อใน Python Shell

```
>>> a.append(3)
>>> a.append(10.1)
>>> a.append('Nobody')
```

จากนั้นลองสั่งให้ Python Shell แสดงค่าของ `a` ออกมา บันทึกผลที่ได้

ทดสอบการเรียกคำสั่งต่อไปนี้ใน Python Shell บันทึกผลที่ได้ในช่องด้านขวา

`a[0]`	
`a[1]`	
`a[2]`	
`a[3]`	
`a[-1]`	
`len(a)`	
`type(a[0])`	
`type(a[-1])`	

จากการทดลอง เราเข้าถึงสมาชิกลำดับที่ `i` ใน list `a` ได้ด้วยคำสั่งใด

เราสามารถหาจำนวนสมาชิกของ list `a` ได้ด้วยคำสั่งใด

สมาชิกใน list จำเป็นจะต้องมีชนิดข้อมูลเหมือนกันทุกตัวหรือไม่

ให้คุณกำหนดค่าให้ `a = []` ใหม่อีกครั้งหนึ่ง จากนั้นให้เรียกคำสั่งต่อไปนี้ตามลำดับ และบันทึกค่าของตัวแปร `a` หลังจากเรียกแต่ละคำสั่งในช่องด้านขวา (คุณสามารถเรียกดูค่าของตัวแปร `a` ได้ด้วยคำสั่ง `print(a)`)

`a.append('M')`	
`a.append((10,9))`	
`a.insert(0,'T')`	
`a.insert(2,23)`	
`a.extend([1,1])`	
`a.remove(1)`	
`a.remove(1)`	
`a.remove(1)`	

จากการทดลอง คำสั่ง `append` มีหน้าที่อย่างไร

คำสั่ง `insert` มีหน้าที่อย่างไร

คำสั่ง `extend` มีหน้าที่อย่างไร

คำสั่ง `remove` มีหน้าที่อย่างไร

ให้คุณทดลองเรียกคำสั่งต่อไปนี้ต่อ และบันทึกผลที่ได้

`a[1:-1]`	
`a[2:]`	
`a[:-1]`	
`a[:]`	

ทดลองสั่งด้วยคำสั่งต่อไปนี้ บันทึกผลที่ได้

```
for element in a:
    print(element)
```

เราสามารถเข้าถึง list ย่อยโดยใช้วิธีการเดียวกันกับชนิดข้อมูลสตริงได้หรือไม่

ใน Python Shell ทดลองเรียกคำสั่งต่อไปนี้และบันทึกผลที่ได้ทางด้านขวามือ

`list(range(5))`	
`list(range(10))`	
`list(range(3, 10))`	
`list(range(3, 10, 2))`	

จากการทดลอง จงสรุปวิธีการใช้งานคำสั่ง `range()` เพื่อสร้าง list ของจำนวนเต็ม

5.2 Mutability

สร้างไฟล์ .py ให้มีโค้ดดังนี้

```
a = []
b = []
a.append(2)
b.append(5)
print('a = ', a)
print('b = ', b)
```

เซฟและรันโปรแกรม บันทึกผลที่ได้

จงวิเคราะห์ว่าตัวแปร a และ b อ้างอิงถึง list ตัวเดียวกันอยู่หรือไม่ เพราะอะไร

ครานี้ทดลองแก้ไขบรรทัดที่สองในไฟล์ดังกล่าว จาก `b = []` เป็น `b = a` เซฟและรันโปรแกรมอีกครั้ง บันทึกผลที่ได้

จงวิเคราะห์ว่าตัวแปร a และ b ในการทดลองหลัง อ้างอิงถึง list ตัวเดียวกันอยู่หรือไม่ เพราะอะไร

ถ้าเราสั่ง `c = a[:]` จงทำการทดลองเพื่อวิเคราะห์ว่าตัวแปร a และตัวแปร c อ้างอิงถึง list ตัวเดียวกันหรือไม่ บันทึกผลที่ได้

จากการทดลอง คำสั่ง `a[:]` เมื่อใช้กับ list จะทำการคืนค่าเป็น list ตัวเดียวกับ `a` มาเลยหรือสร้าง list ใหม่อย่างไร จงอธิบาย

ทดลองสร้างไฟล์ .py ให้มีโค้ดดังนี้

```
import random

def f(a):
    a.append(random.randint(1, 10))
    print('In f, a =', a)

l = [1, 2, 3]
print('1st, l =', l)
f(l)
print('2nd, l =', l)
```

ทดลองรันและบันทึกผลที่ได้

คุณคิดว่าทำไมหลังการเรียกใช้ `f(l)` จึงทำให้สมาชิกใน list `l` เพิ่มขึ้นไปด้วย

จงหาทางแก้ไขโค้ดในบรรทัด `a.append(random.randint(1, 10))` (เพียงบรรทัดเดียวเท่านั้น) ให้เพิ่มจำนวนเต็มที่สุ่มในช่วง 1 - 10 ไปต่อท้าย list `a` เพื่อแสดงในบรรทัดต่อไป โดยไม่ทำให้ list `l` เปลี่ยนแปลงไปจากเดิม บันทึกโค้ดที่ต้องแก้ (บรรทัดเดียว)

ทดลองสร้างไฟล์ .py ให้มีโค้ดดังนี้

```
import random

def f(a = []):
    a.append(random.randint(1, 10))
    print('In f, a =', a)

l = [1, 2, 3]
print('1st, l =', l)
f(l)
print('2nd, l =', l)
```

ทดลองรันและบันทึกผลที่ได้

หลังจากนั้น ลองเรียกคำสั่ง `f()` ติดต่อกันจำนวน 3 ครั้ง บันทึกผลที่เกิดขึ้นตามลำดับ

จงอธิบายว่าทำไมการเรียก `f()` แต่ละครั้งจึงมีสมาชิกจากการเรียกครั้งก่อน ๆ ติดมาด้วย

จงหาทางแก้ไขฟังก์ชัน `f()` ให้มี `a` เป็น default parameter โดยทำให้ในการเรียก `f()` แต่ละครั้ง list `a` ไม่นำสมาชิกจากการเรียกครั้งก่อน ๆ มารวมด้วย บันทึกโค้ดของฟังก์ชันหลังแก้ไข

5.3 list แบบซับซ้อน

สร้างตัวแปร `m1` ตามโค้ดดังนี้

```
m1 = [[1, 1, 1],
      [1, 2, 3],
      [5, 9, 9]]
```

เรามักจะใช้ list ที่มีสมาชิกภายในเป็น list เพื่อใช้แทนตารางสองมิติหรือเมตริกซ์ ตัวอย่างเช่นเราใช้ตัวแปร `m1` นี้แทนเมตริกซ์ของจำนวนเต็มที่มี 3 แถว 3 หลัก

จงสร้างฟังก์ชัน `show()` ที่รับเมตริกซ์ 3 แถว 3 หลักแล้วแสดงในรูปตาราง ตัวอย่างเช่น `show(m1)` จะได้ผลเป็น

```
1 1 1
1 2 3
5 9 9
```

สร้างไฟล์ .py ให้มีโค้ดดังนี้

```
def f1(a, b):
    m = []
    for row_index in range(3):
        row = []
        for col_index in range(3):
            row.append(a[row_index][col_index]+b[row_index][col_index])
        m.append(row)
    return m

m1 = [[1, 1, 1],
      [1, 2, 3],
      [5, 9, 9]]

m2 = [[1, 0, 0],
      [4, 5, 6],
      [2, 3, 2]]
```

โดยให้มีการสร้างฟังก์ชัน `show()` ของคุณเองเข้าไปด้วย จากนั้นรันโปรแกรม และเรียก `show(f1(m1, m2))` บันทึกผลที่ได้

ทดลองเปลี่ยนค่าต่าง ๆ ใน `m1` และ `m2` และเรียกใช้ `f1(m1, m2)` ดู วิเคราะห์และสรุปว่าฟังก์ชัน `f1()` ทำหน้าที่อะไร

ถ้าต้องการให้ฟังก์ชัน `f1()` คืนค่าเป็น list ที่แทนเมตริกซ์ที่เป็นผลจาก `a-b` จะต้องแก้ไขโค้ดในบรรทัดไหนอย่างไร

5.4 ดิกชันนารี

ใน Python Shell ทดลองเรียกคำสั่งต่อไปนี้

```
>>> a = {}
```

หลังจากนั้นให้ลองหาวิธีเพื่อทดสอบว่าตัวแปร `a` มีชนิดข้อมูลเป็นอะไร บันทึกคำตอบ

จากนั้นลองเรียกคำสั่งต่อไปนี้ต่อกันทั้งหมด

```
a['name'] = 'Wittaya'
a['id'] = 4632
a[(3,7)] = True
a[4.75] = [6,3,2]
```

เมื่อเสร็จสิ้นแล้ว หากลองสั่งให้แสดงค่าของ `a` ออกมาด้วยคำสั่ง `print` จะได้ผลอย่างไร

หากเรียกดูจำนวนสมาชิกใน `a` ด้วยคำสั่ง `len` จะได้ผลอย่างไร

หากลองแสดงข้อมูลด้วยคำสั่งต่อไปนี้

```
for key in a:
    print(key)
```

จะได้ผลอย่างไร

จากคำถามล่าสุด ถ้าต้องการแก้ไขให้แสดงค่า (value) ที่อยู่ใน dict `a` ทั้งหมดแทนการแสดง key จะต้องแก้ไขบรรทัดไหนอย่างไร

5.5 โจทย์ปัญหา

1. จงเขียนฟังก์ชัน `intersect()` ที่รับ list สองตัว `a` และ `b` และคืนค่าเป็น list ของ `x` ที่เป็นสมาชิกของทั้ง `a` และ `b` ทั้งหมด ตัวอย่างเช่น

```
>>> intersect([1, 5, 8, 59,72, 101], [3, 4, 5, 7, 8, 70, 72, 96])
[5, 8, 72]
>>> intersect([1, 2, 3, 4], [5, 6, 7])
[]
```

2. จงเขียนฟังก์ชัน `transpose()` ที่รับเมตริกซ์ M ขนาดเท่าใดก็ได้ และคืนค่าเป็นเมตริกซ์ transpose ของ M ตัวอย่างการใช้งานเช่น

```
>>> m = [[5, 4], [4, 0], [7, 10], [-1, 8]]
>>> transpose(m)
[[5, 4, 7, -1], [4, 0, 10, 8]]
```

3. จงเขียนฟังก์ชัน `multiply()` ที่รับเมตริกซ์ A และ B ขนาดเท่าใดก็ได้ และคืนค่าเป็นผลคูณ $A \times B$ โดยหากไม่สามารถคูณได้ให้คืนค่าเป็น `None` ตัวอย่างการใช้งานเช่น

```
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> b = [[7, 8], [9, 10], [11, 12]]
>>> multiply(a, b)
[[58, 64], [139, 154]]
```