Python: งูเหลือมตะลุยจักรวาล

1 เริ่มต้นกัน

คุณทะเบียนตื่นขึ้นมาพบว่า ในเครื่องของตนเองมีข้อมูลของการเลือกสาขาวิชาของนิสิตวิศวกรรมศาสตร์เก็บอยู่ วันประกาศผลการเลือกสาขาวิชาคือพรุ่งนี้แล้วแต่เขายังไม่ได้ทำอะไรเลย จะไปไหว้วานคนอื่นมาช่วยก็คงไม่ทัน เพราะว่าวันนี้เป็นวันอาทิตย์ เขาเลยตัดสินใจจะเขียนโปรแกรมเพื่อประมวลผลข้อมูลดังกล่าวด้วยตัวเอง เนื่องจากเขาได้ยินว่าตอนนี้ภาษา Python นั้นฮิปเหลือเกิน เขาจึงเลือกที่จะเขียนโปรแกรมด้วยภาษาดังกล่าว

แฟ้มข้อมูลในเครื่องของเขามีดังนี้ (ดาวน์โหลดได้จาก `http://garnet.cpe.ku.ac.th/~jtf/223/python/`)

- `courses.txt` เก็บรายการของรายวิชาที่นำมาคิดเกรดเฉลี่ย
- `grades.txt` เก็บเกรดทั้งหมดของนิสิต
- `departments.txt` เก็บข้อมูลของแต่ละภาควิชา (เช่นจำนวนรับ)
- `preferences.txt` เก็บลำดับการเลือกภาควิชา

1.1 การติดตั้ง Python

ดาวน์โหลด Python ได้จาก `http://www.python.org/download/`

เมื่อติดตั้งเรียบร้อยเราสามารถเรียกคำสั่ง `python` ได้จาก command-line ถ้าเราเรียกคำสั่งดังกล่าวโดยไม่ระบุชื่อโปรแกรมที่จะให้ทำงาน Python จะทำงานในลักษณะของ intepreter นั่นคือจะรับคำสั่งแล้วโต้ตอบผลลัพธ์แบบทันที ถ้าเราเรียก `python` พร้อมกับระบุแฟ้มโปรแกรมภาษา Python ที่จะให้ทำงาน Python จะอ่านโปรแกรมแล้วทำงานตามนั้น

ในการทดลองวันนี้เราจะใช้ระบบเขียนโปรแกรม (ide) ที่มากับ Python ชื่อว่า IDLE

1.2 อ่านข้อมุลและคำนวณเกรดเฉลี่ย

พิจารณาโปรแกรมแรกของเราดังนี้ (ป้อนลงในแฟ้ม `read-courses.py`)

```
f = open('courses.txt','r')
lines = f.readlines()
course_count = 0
total_credit = 0
for line in lines:
    items = line.strip().split(',')
    print items[0], items[1]
    course_count += 1
    total_credit += int(items[1])
print "Total of", course_count, "courses with", total_credit, "credits"
```

โปรแกรมดังกล่าว เปิดแฟ้ม `courses.txt` จากนั้นอ่านข้อมูลเข้ามาเพื่อพิมพ์รายการรายวิชา และนับจำนวนรายวิชาและหาผลรวมของหน่วยกิต ให้จัดเก็บโปรแกรมดังกล่าวในโฟล์เดอร์เดียวกับแฟ้มข้อมูล จากนั้นเรียกให้โปรแกรมทำงานโดยสั่ง `python read-courses.py`

เราจะค่อย ๆ พิจารณาโปรแกรมดังกล่าวพร้อมกับเรียนรู้ไวยากรณ์ของภาษา Python

□ **ไม่มีเครื่องหมายปิดคำสั่ง.** คำสั่ง (statement) ไม่จำเป็นต้องปิดด้วยเครื่องหมาย '`;`' อย่างไรก็ตามถ้าเราเขียนหลายคำสั่งในบรรทัดเดียวกัน เราจะใช้เครื่องหมาย '`;`' คั่นระหว่างคำสั่ง

□ **ไม่ต้องประกาศตัวแปร ไม่ต้องระบุแบบชนิดข้อมูล (type).** สังเกตว่าเราสามารถใช้ตัวแปรต่าง ๆ ได้เลย โดยไม่จำเป็นต้องประกาศ หรือระบุ type ของตัวแปรนั้น type สำหรับตัวแปรแต่ละตัวจะถูกระบุไปกับตัวแปรเมื่อโปรแกรม

ทำงาน ลักษณะดังกล่าวเป็นคุณสมบัติหนึ่งของภาษาโปรแกรมเชิงพลวัติ (dynamic language) อย่างไรก็ตาม ไม่ใช่ว่าตัวแปรแต่ละตัวจะไม่มี type สังเกตว่าเราต้องแปลงข้อมูล เวลาจะนำค่าจากตัวแปร `items[1]` ไปใช้เป็นจำนวนเต็มด้วยฟังก์ชัน `int`

□ **โครงสร้างแบบบล็อค.** ภาษา Python เป็นภาษาระดับสูงที่มีโครงสร้างแบบบล็อกเช่นเดียวกับภาษาอื่น ๆ อย่างไรก็ตาม วิธีการระบุบล็อกใน Python ไม่ได้ทำโดยใช้เครื่องหมาย เช่น เครื่องหมายปีกกา ({ หรือ }) อย่างเช่นภาษา C หรือ Java แต่ภาษา Python ใช้การเว้นย่อหน้าแทน ส่วนของโปรแกรมที่มีการเว้นระยะย่อหน้าเท่ากันจะถือว่าอยู่ในบล็อกเดียวกัน และการเว้นย่อหน้าเข้าไปเป็นการแสดงบล็อก

สังเกตด้วยว่าบรรทัดก่อนหน้าจะสิ้นสุดด้วยเครื่องหมาย ‘`:`’ แทนการบอกว่าต่อไปจะเป็นบล็อก

โปรแกรมด้านล่างแสดงอีกตัวอย่างของการใช้ย่อหน้าในการระบุบล็อก ในโปรแกรมดังกล่าว ถ้า `x` น้อยกว่าหรือเท่ากับ 10 โปรแกรมจะพิมพ์คำว่า `Welcome` ออกมา

```
if x>10:
    print "Hello"
    if x<20:
        print "Good bye"
else:
    print "Welcome"
```

โปรแกรมดังกล่าวจะทำงานต่างออกไปถ้าเราเปลี่ยนการย่อหน้า เช่น ถ้าเราเขียน

```
if x>10:
    print "Hello"
    if x<20:
        print "Good bye"
    else:
        print "Welcome"
```

เมื่อ `x<=10` โปรแกรมจะไม่พิมพ์อะไรออกมา

□ **อ่านข้อมูลและลิสต์.** สองบรรทัดด้านล่างเปิดแฟ้มและอ่านข้อมูลจากทั้งแฟ้มมาเก็บในตัวแปร `lines`

```
f = open('courses.txt','r')        # เปิดแฟ้ม
lines = f.readlines()              #อ่านทั้งแฟ้มมา เก็บในตัวแปร lines
```

เมท็อด `readlines` คืนค่าเป็นลิสต์ของสตริง ลิสต์เป็นโครงสร้างข้อมูลพื้นฐานใน Python สำหรับจัดการกับลำดับของข้อมูล (เราจะทดลองต่อไป)

□ **พิจารณาข้อมูลทุกตัวในลิสต์.** คำสั่งควบคุม `for` ใช้เพื่อจะ “วิ่ง” เข้าไปในรายการ ตัวอย่างการใช้คำสั่ง for แสดงด้านล่าง

```
primes = [2,3,5,7,9]               # would print:
print 'First 5 primes are:'        #    First 5 primes are:
for p in primes:                   #    2
    print p                        #    3 ...
```

ในโปรแกรมด้านต้นตัวแปร line จะถูกใช้เพื่อพิจารณาข้อมูลทุกตัวในลิสต์ ตัวอย่างข้อมูลแต่ละบรรทัดเช่น `"204111,3\n"`

จากนั้นเราสั่ง `line.strip()` เพื่อตัดตัวอักษรเช่น `"\n"` (ขึ้นบรรทัดใหม่) ทิ้ง แล้วสั่ง `split(',')` เพื่อแยกสตริงออกมาเป็นส่วน ๆ ด้วยโดยใช้ตัวอักษร ‘,’ เป็นตัวแบ่ง ค่าที่คืนมาจะเป็นลิสต์ ซึ่งเราจะใช้ต่อเป็น `items[0]` (เก็บรหัสวิชา) และ `items[1]` (หน่วยกิต)

1.3 โครงสร้างข้อมูลและโครงสร้างควบคุมพื้นฐานใน Python

ในส่วนนี้เราจะทดลองรูปแบบข้อมูลและโครงสร้างควบคุมพื้นฐานใน Python เราจะทดลองใน python intepreter เมื่อเรียกใช้ interpreter เราจะพบหน้าจอดังนี้

```
Python 2.5.1 (r251:54863, Mar  7 2008, 04:10:12)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

เครื่องหมาย >>> เป็นสัญลักษณ์รอรับคำสั่งของ interpreter เราจะแสดงตัวอย่างการใช้งานบน interpreter นี้

1.3.1 จำนวนเต็ม/จำนวนจริง

ใช้งานไม่ต่างจากภาษาทั่วไป เช่น

```
>>> 10/20               #> 0
>>> 10.5/20             #> 0.52500000000000002
>>> 10**2+7             #> 107
```

1.3.2 บูเลียน

ในภาษา Python ค่าคงที่ `False` (เขียนขึ้นต้นด้วยตัวใหญ่), `None` (สำหรับแทนวัตถุที่ว่างเปล่า), ค่าศูนย์, และลำดับว่างต่าง ๆ จะมีความหมายเป็นเท็จ ค่าคงที่ True และค่าอื่น ๆ ที่ไม่ใช่ค่าว่างจะมีค่าเป็นจริง

ตัวดำเนินการทางตรรกศาสตร์ที่ Python มีคือ `and`, `or`, และ `not`

1.3.3 สตริง

ดูเมท็อดเพิ่มเติมได้ที่ `http://docs.python.org/lib/string-methods.html`

ค่าคงที่แบบสตริงใน Python สามารถเขียนโดยใช้เครื่องหมายคำพูดเดี่ยวหรือคำพูดคู่ก็ได้ ถ้าต้องการพิมพ์เครื่องหมายคำพูดภายในสตริงสามารถทำได้โดยใช้เครื่องหมาย ‘\’

ตัวอย่างด้านล่างแสดงการใช้งานสตริงพื้นฐาน

```
>>> s = "Hello"
>>> s                    #> 'Hello'
>>> s + "world"          #> 'Helloworld'
>>> t = 'it\'s a happy day'
>>> print t              #> it's a happy day
>>> t*3                  #> "it's a happy dayit's a happy dayit's a happy day"
>>> t*2 + s              #> "it's a happy dayit's a happy dayHello"
>>> t.split('a')         #> ["it's ", ' h', 'ppy d', 'y']
>>> m = "This is\n on many\n lines"
>>> print m
This is
 on many
 lines
>>> y = "     center  yo yo   \n"
>>> y.strip()            #> 'center  yo yo'
```

นอกจากนี้เราสามารถใช้สตริงที่ยาวต่อเนื่องหลายบรรทัดได้โดยใช้เครื่องหมาย """ ยกตัวอย่างเช่น

```
>>> manual = """First you call
... python and then you start
... using it"""
>>> print manual
First you call
python and then you start
using it
```

1.3.4 การอ่านข้อมูล (raw_input)

เราสามารถใช้ฟังก์ชัน raw_input เพื่ออ่านข้อมูลจากผู้ใช้ ค่าที่คืนจากฟังก์ชันนี้เป็นสตริงที่อ่านได้

```
>>> name = raw_input()        #> Somchai
>>> print 'Hello ' + name     #> Hello Somchai
>>> xstr = raw_input()        #> 120
>>> print xstr*10                  # ค่าที่อ่านได้เป็น string
120120120120120120120120120120
>>> print int(xstr)*10             # แปลงเป็น integer ก่อน
1200
```

1.3.5 list (ลิสต์—รายการ)

ลิสต์คือโครงสร้างข้อมุลที่สำคัญที่สุดใน Python เราระบุลิสต์ได้โดยเขียนรายการข้อมูลในลิสต์ ไว้ภายในเครื่องหมายวงเล็บเหลี่ยม คั่นข้อมูลแต่ละตัวด้วยเครื่องหมายลูกน้ำ ข้อมูลในลิสต์จะเป็นข้อมูลใดก็ได้ และยังสามารถเป็นลิสต์ได้ด้วย

เราสามารถอ้างถึงข้อมูลในลิสต์ได้โดยระบุดัชนีของข้อมูล และใช้ฟังก์ชัน len เพื่ออ่านจำนวนข้อมูลในลิสต์ ด้านล่างแสดงตัวอย่างการใช้งาน

```
>>> a = [1,2,3,4,5,6]
>>> a[1] + a[4]               #> 7
>>> b = [1,2,"Welcome",4,5]
>>> len(b)                    #> 5
>>> c = [a,b]
>>> c                         #> [[1, 2, 3, 4, 5, 6], [1, 2, 'Welcome', 4, 5]]
>>> len(c)                    #> 2
>>> c[1]                      #> [1, 2, 'Welcome', 4, 5]
>>> c[1][2]                   #> 'Welcome'
```

ในการวิ่งไปเพื่อประมวลผลข้อมูลในลิสต์ เราสามารถใช้คำสั่ง for ที่มีรูปแบบดังนี้

```
for <variable> in <list>:
    # do something
```

เพื่อใช้ตัวแปร variable วิ่งไปในลิสต์ list ได้

ตัวอย่างด้านล่างแสดงส่วนของโปรแกรมที่หาผลรวมของจำนวนในลิสต์ a

```
>>> a = [1,2,3,10,20,30]
>>> sum = 0
>>> for x in a:
...     sum += x
... print sum
66
```

สังเกตว่าเราใช้การย่อหน้าในการระบุขอบเขตของคำสั่งที่จะทำงานภายใต้คำสั่ง for

1.3.6 แบบฝึกหัด: นับหน่วยกิต

จงเขียนโปรแกรม count-credits.py ที่อ่านแฟ้ม courses.txt จากนั้นพิมพ์ผลรวมของหน่วยกิตของวิชาทั้งหมดออกมา (หมายเหตุ: ให้เขียนเป็นโปรแกรมเดี่ยว ที่ทำงานโดยเรียกใช้ผ่าน command line: python count-credits.py)

1.3.7 เงื่อนไข

Python มีโครงสร้างเงื่อนไข if ที่มีรูปแบบทั่วไปดังนี้

```
if <condition>:
    # do one thing
```

```
elif <condition>:
    # do another thing
else:
    # do something
```

ส่วน `elif` และ `else` สามารถละได้

โปรแกรมด้านล่างพิมพ์รายการวิชาที่ขึ้นต้นด้วยหมายเลขตามที่ระบุใน command line

```
import sys                      # need this to get command-line args

cprefix = sys.argv[1]           # get 1st command-line argument
lines = open('courses.txt').readlines()

for line in lines:
    if line.startswith(cprefix):
        print line,       # note the ',' . "line" has an '\n' at the end already
```

โปรแกรมดังกล่าวเรียกให้ทำงานโดยสั่ง `python find-courses.py <เลขรหัสขึ้นต้น>` เช่นเราสามารถใช้เลขรหัสขึ้นต้นเป็น `420` เพื่อหารายวิชาของภาควิชาฟิสิกส์ได้

สองบรรทัดแรกของโปรแกรมเป็นการอ่าน command-line argument ส่วนเมท็อด `startswith` จะตรวจสอบสตริงว่าขึ้นต้นด้วยสตริงที่ให้มาหรือไม่

ให้สังเกตคำสั่ง `print` ถ้าเราใส่เครื่องหมายลูกน้ำไว้ตอนท้าย คำสั่งจะไม่ขึ้นบรรทัดใหม่ให้ แต่ในกรณีของข้อนี้ แต่ละบรรทัด (ในตัวแปร `line`) มีเครื่องหมายขึ้นบรรทัดใหม่ต่อท้ายอยู่แล้ว

1.3.8 แบบฝึกหัด: นับหน่วยกิตเฉพาะรายวิชา

จงเขียนแก้โปรแกรม `find-courses.py` ด้านบน ให้พิมพ์จำนวนหน่วยกิตรวม ของรายวิชาที่ค้นหามาได้ด้วย ตัวอย่างการทำงานเช่น ถ้าสั่ง `python find-courses.py 420` จะได้ผลลัพธ์เป็น

```
420111,3
420113,1
420112,3
Total: 7 credits
```

1.3.9 การประมวลผลในลิสต์

เรามีฟังก์ชันมากมายในการประมวลผลในลิสต์ ด้านล่างแสดงตัวอย่างการใช้งานที่นำมาจากเว็บของ Python (`http://docs.python.org/tut/node7.html`)

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')   #> 2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a                          #> [66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)               #> 1
>>> a.remove(333)
>>> a                          #> [66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a                          #> [333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a                          #> [-1, 1, 66.25, 333, 333, 1234.5]
```

นอกจากการใช้งานฟังก์ชันดังกล่าวแล้ว เรายังสามารถใช้งานช่วงของลำดับได้ด้วย พิจารณาตัวอย่างด้านล่าง

ฟังก์ชั่นอื่น ๆ ดูได้ที่ `http://docs.python.org/lib/typesseq-mutable.html`

1.3.10 tuple (ทูเปิล)

ข้อมูลแบบทูเปิลมีลักษณะไม่ต่างจากข้อมูลแบบลิสต์ เพียงแต่ว่าเราจะไม่สามารถ**เปลี่ยนค่า**ข้อมูลย่อยในทูเปิลได้ ภาษา Python เรียกชนิดข้อมูลที่ไม่สามารถแก้ไขได้ว่าเป็นข้อมูลแบบ*immutable* ตัวอย่างของข้อมูลแบบ immutable เช่น จำนวนต่าง ๆ สตริง หรือทูเปิล (สังเกตว่าเราสามารถเปลี่ยนค่าของจำนวนต่าง ๆ ได้แต่เมื่อเปลี่ยนแล้วจำนวนนั้นก็ไม่ใช่จำนวนเดิมอีกต่อไป)

เราเขียนทูเปิลโดยระบุสมาชิกเรียงตามลำดับในวงเล็บ เช่น `(2,3,4)` หรือ `('today','monday',2)` เป็นต้น ข้อมูลแบบทูเปิลมักถูกใช้เป็นกุญแจในข้อมูลแบบพจนานุกรมที่อธิบายถัดไป

1.3.11 dictionary (พจนานุกรม)

โครงสร้างข้อมูลอีกชนิดของ Python คือพจนานุกรม โครงสร้างข้อมูลชนิดนี้รองรับการสืบค้นโดยใช้กุญแจ (key) เพื่อโยงไปถึงค่าที่เก็บไว้ โดยกุญแจจะเป็นข้อมูลแบบ immutable เช่น ตัวเลข สตริง หรือลำดับ (tuple) ก็ได้

การระบุพจนานุกรมเราจะเขียนข้อมูลในวงเล็บปีกกา โดยระบุเป็นรายการของคู่ลำดับ กุญแจ-ค่า ดังแสดงในตัวอย่างการใช้งานด้านล่าง (คัดลอกมาจาก `http://docs.python.org/tut/node7.html`)

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel                          #> {'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']                  #> 4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel                          #> {'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()                   #> ['guido', 'irv', 'jack']
>>> tel.has_key('guido')         #> True
>>> 'guido' in tel               #> True
```

ตัวอย่างด้านล่างแสดงวิ่งไปในข้อมูลแบบพจนานุกรมผ่านทางรายการของกุญแจ (เรียกด้วยเมท็อด `keys` ด้วยคำสั่ง `for`

```
>>> x = {'204111':3, '420111':4, '417111':3}
>>> for k in x.keys():
...     print k, x[k]
417111 3
420111 4
204111 3
```

โปรแกรมด้านล่างอ่านแฟ้ม `courses.txt` แล้วรับรหัสวิชาจากผู้ใช้แล้วพิมพ์หน่วยกิต

```
f = open('courses.txt','r')
lines = f.readlines()
credits = {}
for line in lines:
    items = line.strip().split(',')
    credits[items[0]] = int(items[1])

print "Enter course number (type 'end' to quit):"

c = raw_input()
while c!='end':
    if c in credits:
        print credits[c]
    else:
        print 'Course not found'
    c = raw_input()
```

สังเกตว่าโปรแกรมข้างต้นใช้โครงสร้างควบคุมแบบ `while`

1.3.12 แบบฝึกหัด: คำนวณเกรดเฉลี่ย 1 คน

แฟ้ม `my_grades.txt` เก็บเกรดของวิชาต่าง ๆ ของนิสิตคนหนึ่ง ในแต่ละบรรทัดของแฟ้มดังกล่าว จะมีข้อมูลในรูปแบบ `<รหัสวิชา>,<เกรด>` โดยเกรดจะเป็นจำนวนจริงแทนแต้มคะแนนของเกรดนั้น ๆ (เช่น 2.5 แทน C+ เป็นต้น)

ให้เขียนโปรแกรม `compute_gpa.py` ที่อ่านแฟ้มดังกล่าวและแฟ้ม `courses.txt` จากนั้นคำนวณเกรดเฉลี่ยของนิสิตคนนั้น

คำใบ้ ในการคำนวณเราจำเป็นต้องทราบหน่วยกิตของแต่ละวิชา ให้ดูตัวอย่างในส่วน 1.3.11 สำหรับการสร้างพจนานุกรมชื่อ `credits` ที่มีกุญแจเป็นรหัสวิชา และมีค่าเป็นหน่วยกิตของวิชานั้นได้

2 ทำจนเสร็จ

2.1 ฟังก์ชัน

จากแบบฝึกหัดข้อที่ผ่านมา ถ้าข้อมูลเกรดในแฟ้มดังกล่าวถูกเก็บเป็นสตริง เช่น `C+` หรือ `B` ในการคำนวณเกรดเฉลี่ยเราจำเป็นต้องแปลงสตริงให้เป็นค่าแต้มคะแนน อย่างไรก็ตามในภาษา Python ไม่มีโครงสร้าง switch เราจึงจำเป็นต้องเขียนส่วนของโปรแกรมดังเช่นด้านล่าง

```
g = raw_input()
if g=='A':
    score = 4
elif g=='B+':
    score = 3.5
...                  # ละไว้
else:
    score = 0
print score
```

อย่างไรก็ตามเราสามารถใช้ข้อมูลแบบพจนานุกรมเพื่อทำให้โปรแกรมอ่านง่ายขึ้นได้ เช่น

```
g = raw_input()
scores = {'A':4, 'B+':3.5, 'B':3, 'C+':2.5,
          'C':2, 'D+':1.5, 'D':1, 'F':0 }
print scores[g]
```

ในโปรแกรมที่เราจะเขียนให้คุณทะเบียน เราจำเป็นต้องใช้การแปลงข้อมูลดังกล่าวหลายครั้ง เราสามารถเขียนโปรแกรมให้เป็นส่วน ๆ ได้โดยยุบรวมความสามารถดังกล่าวเป็นฟังก์ชัน ส่วนของโปรแกรมดังกล่าวสามารถเขียนเป็นฟังก์ชันได้ดังนี้

```
def convert_score(s):
    scores = {'A':4, 'B+':3.5, 'B':3, 'C+':2.5,
              'C':2, 'D+':1.5, 'D':1, 'F':0 }
    return scores[s]
```

เมื่อเราเรียกใช้ `convert_score('B')` เราจะได้ผลลัพธ์เป็น 3 (สังเกตว่าถ้าเราเขียนข้อมูลในวงเล็บ เราสามารถใช้ย่อหน้าอย่างไรก็ได้ แต่คำสั่ง `return` ต้องย่อหน้าตรงกับบรรทัด `scores = ...`)

2.1.1 แบบฝึกหัด: คำนวณเกรดเฉลี่ยทั้งหมด

แฟ้ม `grades.txt` เก็บเกรดของวิชาต่าง ๆ ของนิสิตคณะวิศวกรรมศาสตร์ ในแต่ละบรรทัดของแฟ้มดังกล่าว จะมีข้อมูลในรูปแบบ `<รหัสนิสิต>,<รหัสวิชา>,<เกรด>` โดยเกรดจะเป็นรหัสคะแนน ('A' ถึง 'F') ข้อมูลในแฟ้มดังกล่าวอาจจะเรียงตามลำดับอย่างไรก็ได้ (ไม่จำเป็นต้องเรียงตามรหัสนิสิตหรือรหัสวิชาแต่อย่างใด)

ให้เขียนโปรแกรม `compute_all_gpa.py` ที่อ่านแฟ้มดังกล่าวและแฟ้ม `courses.txt` จากนั้นคำนวณเกรดเฉลี่ยของนิสิตทุกคน โดยพิมพ์ผลลัพธ์ออกมาในรูปแบบ `<รหัสนิสิต>,<เกรดเฉลี่ย>`

ในโปรแกรมของแบบฝึกหัดนี้ ให้ใช้ฟังก์ชัน `convert_score` ข้างและเขียนฟังก์ชัน `build_credits` ที่อ่านแฟ้ม `courses.txt` แล้วคืนค่าเป็นพจนานุกรมดังที่อธิบายในส่วน 1.3.12

คำใบ้ เนื่องจากข้อมูลเกรดนั้นไม่เรียงลำดับ สามารถใช้โครงสร้างข้อมูลแบบพจนานุกรมในการเก็บคะแนนรวมสะสมและหน่วยกิตสะสมของนิสิตแต่ละคนได้

หมายเหตุ ลองดูตัวอย่างด้านล่างในการจัดรูปแบบการพิมพ์ โดยใช้ operator `%` และใส่ข้อมูลในทูเปิล

```
>>> print 50051111,1.23
50051111 1.23
>>> print 50051111,',',1.23
50051111 , 1.23
>>> print '%s,%s' % (50051111,1.23)      # เราต้องการรูปแบบนี้
50051111,1.23
```

2.1.2 แบบฝึกหัด: ฟังก์ชันอ่านแฟ้มรูปแบบข้อมูลคั่นด้วยเครื่องหมายลูกน้ำ

สังเกตว่าในการทำงานต่อ ๆ ไป เราจะต้องอ่านแฟ้มอีกหลาย ๆ แฟ้มที่มีรูปแบบการจัดเก็บข้อมูลในรูปแบบเดียวกัน นั่นคือเก็บเป็นรายการของบรรทัดข้อมูล โดยในแต่ละบรรทัดประกอบด้วยข้อมูลหลายตัวคั่นด้วยเครื่องหมายลูกน้ำ (comma) ดังนั้นเพื่อความสะดวกเราจะเขียนฟังก์ชัน `readcsvfile` ที่อ่านข้อมูลจากแฟ้มเหล่านี้ ฟังก์ชัน `readcsvfile` จะคืนข้อมูลเป็นลิสต์ของลิสต์ การประกาศฟังก์ชันและตัวอย่างการเรียกใช้ฟังก์ชันแสดงดังด้านล่าง

```
def readcsvfile(fname):
    # your code here

data = readcsvfile('courses.txt')
print data
```

ผลลัพธ์เมื่อเรียกใช้งาน (สมมติว่าเก็บโปรแกรมในแฟ้ม `readcsv.py`)

```
$ python readcsv.py               # or c:\> python readcsv.py
[['417167', '4'], ['420111', '3'], ['420113', '1'], ... , ['402114', '1']]
```

คำใบ้ การเพิ่มข้อมูลเข้าไปท้ายลิสต์ใช้ฟังก์ชัน `append`

2.1.3 แบบฝึกหัด: คำนวณเกรดเฉลี่ยของทุกคน (แยกฟังก์ชัน)

แก้โปรแกรมในแบบฝึกหัด 2.1.1 ให้เขียนหรือใช้ฟังก์ชันดังต่อไปนี้

- `convert_score(g)` ดังส่วน 2.1
- `readcsvfile(fname)` ที่เขียนในแบบฝึกหัด 2.1.2
- `build_credits(course_data)` แก้ฟังก์ชันที่ทำในแบบฝึกหัดก่อนหน้า ให้ไม่ต้องอ่านแฟ้ม `courses.txt` โดยตรง แต่รับข้อมูลที่ได้จากการอ่านด้วยฟังก์ชัน `readcsvfile` แทน
- `compute_gpa(credits,grade_data)` รับข้อมูลที่อ่านจากแฟ้ม `courses.txt` และ `grades.txt` ด้วยฟังก์ชัน readcsvfile จากนั้นคำนวณ gpa โดยคืนค่าเป็น dictionary ที่มีกุญแจเป็นรหัสประจำตัวนิสิต

โดยส่วนของโปรแกรมหลักที่เรียกใช้ฟังก์ชันต่าง ๆ ให้ใช้ตามด้านล่างนี้

```
course_data = readcsvfile('courses.txt')
credits = build_credit(course_data)
grade_data = readcsvfile('grades.txt')
gpa = compute_gpa(credits,grade_data)

for id in gpa.keys():
    print id, gpa[id]
```

ตัวอย่างผลลัพธ์เมื่อเรียกใช้งาน

```
50000043 2.53703703704
50000042 1.51851851852
50000041 2.44444444444
...                              #ละไว้
```

2.1.4 แบบฝึกหัด: อ่านลำดับการเลือก

เขียนฟังก์ชัน `build_pref` ที่ใช้ข้อมูลที่ได้จากการอ่านลำดับการเลือกที่อยู่ในแฟ้ม `preferences.txt` ผ่านทางฟังก์ชัน `readcsvfile` แล้วคืนค่าข้อมูลแบบพจนานุกรมของลิสต์ในลักษณะที่จะได้อธิบายต่อไป

ในแต่ละบรรทัดของแฟ้มดังกล่าว จะมีข้อมูลในรูปแบบ `<รหัสนิสิต>,<อันดับการเลือก>,<รหัสภาควิชา>` โดยอันดับการเลือกจะมีค่าตั้งแต่ 1 ถึง 8 (มี 8 ภาควิชา) ข้อมูลในแฟ้มเรียงลำดับตามเลขประจำตัวและอันดับการเลือก

ค่าที่ฟังก์ชันดังกล่าวคืนมาจะเป็นพจนานุกรมที่มีกุญแจเป็นรหัสประจำตัวนิสิต มีค่าเป็นรายการของภาควิชาเรียงตามลำดับการเลือก

ตัวอย่างเช่น ถ้าในแฟ้มมีข้อมูล

```
1234,1,204
1234,2,210
1234,3,208
5678,1,208
5678,2,204
5678,3,210
```

ค่าที่คืนจากฟังก์ชันจะเป็น `{'1234':['204','210','208'], '5678':['208','204','210']}`

ทดลองนำไปใช้กับโปรแกรมหลักด้านล่างนี้

```
pref_data = readcsvfile('preferences.txt')
pref = build_pref(pref_data)

for id in pref.keys():
    print '(%s)' % (id,),
    for r in pref[id]:
        print r,
    print
```

สังเกตว่าเราเขียน `(id,)` เพื่อระบุข้อมูลทูเปิลที่มีสมาชิกหนึ่งตัว เราใส่เครื่องหมายลูกน้ำ เพื่อแยกแยะระหว่าง `(id)` กับ `(id,)`

ตัวอย่างผลลัพธ์เมื่อเรียกใช้งาน

```
(50000043) 215 206 213 204 203 202 205 208
(50000042) 204 213 203 208 206 202 205 215
...                              #ละไว้
```

2.1.5 คำสั่งและการใช้งาน Python เพิ่มเติม

□ **การกำหนดค่าแบบขนาน.** ใน Python เราสามารถสั่งกำหนดค่าแบบขนาน (คือการกำหนดค่าตัวแปรหลายตัวพร้อมกัน) ได้ ดังเช่น

```
>>> a,b,c = 1,2,3              # a=1, b=2, c=3
>>> e,f,g = a,b,c              # e=a=1, f=b=2, g=c=3
>>> a,b = b,a                  # a=2, b=1 (สลับค่า)
>>> a,b = [1,2]                # unpack: a=1, b=2
>>> a,b,c = [1,2,[3,4]]        # unpack: a=1, b=2, c=[3,4]
>>> [a,b,[c,d]] = [1,2,[3,4]]  # unpack: a=1, b=2, c=3, d=4
```

ในตัวอย่างตอนต้นหลายตัวอย่าง เราอาจเขียนคำสั่ง `id,course,grade = items` หรือกระทั่ง `id,course,grade = line.split(',')` เพื่อแยกข้อมูลจากลิสต์มาใส่ในตัวแปรย่อย ๆ ได้

□ **ฟังก์ชัน** `range`**.** เราอาจสงสัยว่าคำสั่งวนรอบเช่น `for(i=0; i<10; i++)` ในภาษา C จะนำมาเขียนใน Python ได้อย่างไร เราจะใช้ฟังก์ชัน `range` ซึ่งรับค่าเป็นจำนวนเต็มแล้วคืนค่าเป็นลิสต์[1] ที่เริ่มตั้งแต่ 0 จนถึงค่าจำนวนเต็มดังกล่าวลบหนึ่ง เราจะใช้ฟังก์ชันดังกล่าวประกอบกับคำสั่ง `for` ตัวอย่างเช่น

```
>>> for i in range(10):
...     print i,
0 1 2 3 4 5 6 7 8 9
```

□ **การจัดเรียงข้อมูล.** ลิสต์มีเมท็อดที่สำคัญคือเมท็อด `sort` ในการจัดเรียงข้อมูล เมท็อดดังกล่าวทำลายค่าในลิสต์เดิม นอกจากนี้เมท็อดดังกล่าวยังไม่คืนค่าลิสต์ออกมาด้วย พิจารณาตัวอย่างการใช้งาน

```
>>> a = [1,4,3,2,5]
>>> a.sort()
>>> a                                          #> [1, 2, 3, 4, 5]
>>> b = [1,5,4,3,2]
>>> for x in b.sort():
...     print x
...
Traceback (most recent call last):            # เมท็อด sort ไม่คืนค่า
  File "<stdin>", line 1, in <module>         #
TypeError: 'NoneType' object is not iterable  #
```

ถ้าต้องการฟังก์ชันที่คืนค่าและไม่เปลี่ยนค่าในลิสต์ให้ใช้ฟังก์ชัน `sorted` แทน ดังตัวอย่างต่อไปนี้

```
>>> c = [1,5,4,3,2]
>>> for x in sorted(c):
...     print x
1
..                                            # ละไว้
5
>>> c                                          #> [1, 5, 4, 3, 2]
```

ในกรณีที่ข้อมูลในลิสต์ไม่ใช่จำนวน (เช่นเป็นทูเปิล) คำสั่งจัดเรียงดังกล่าวจะเรียงโดยอิงจากหลักหน้าไปก่อน เช่น (1,2) น้อยกว่า (2,1) หรือ (2,4) น้อยกว่า (2,5) เป็นต้น

2.1.6 แบบฝึกหัด: โปรแกรมจัดอันดับการเข้าสาขา

จากส่วนของฟังก์ชันที่ได้เขียนไว้ ให้เขียนโปรแกรมจัดอันดับการเข้าสาขาของนิสิต ในกรณีที่มีนิสิตได้คะแนนเฉลี่ยเท่ากันเพื่อความง่ายให้เลือกนิสิตที่มีเลขประจำตัวต่ำกว่า

อ่านข้อมูลเกี่ยวกับแต่ละภาควิชาได้จากแฟ้ม `departments.txt` แฟ้มดังกล่าวเก็บข้อมูลภาควิชาละหนึ่งบรรทัด โดยเริ่มจากรหัสภาควิชา ชื่อภาควิชา และจำนวนรับ (มากที่สุด)

ให้โปรแกรมพิมพ์ผลลัพธ์ในรูปแบบดังนี้

```
ชื่อภาควิชา                  (ตามลำดับในแฟ้ม departments.txt)
รหัสประจำตัว เกรดเฉลี่ย      (เรียงตามลำดับจากมากไปน้อยจนหมด)
รหัสประจำตัว เกรดเฉลี่ย
...

ชื่อภาควิชา
รหัสประจำตัว เกรดเฉลี่ย      (เรียงตามลำดับจากมากไปน้อยจนหมด)
รหัสประจำตัว เกรดเฉลี่ย
...
```

[1] จริง ๆ แล้วฟังก์ชันดังกล่าวไม่ได้สร้างลิสต์ขึ้นมาทั้งหมดในครั้งเดียว