บทที่ 10

การใช้งานโมดูลและแพ็คเกจ

ในบทที่ผ่านมาผู้อ่านได้ศึกษาเกี่ยวกับวิธีการสร้างฟังก์ชันขึ้นมาใช้งาน รวมไปถึงการเรียกใช้งานฟังก์ชันในรูปแบบต่าง ๆ และได้รู้จักกับฟังก์ชัน Built-in ของภาษาไพธอนไปแล้ว ในบทนี้ผู้อ่านจะได้เรียนรู้วิธีการนำเอาฟังก์ชันต่าง ๆ มารวมกัน แล้วสร้างเป็นโมดูลและแพ็กเกจ (Module and Package) ซึ่งทำให้เราสามารถเก็บไว้ใช้ในอนาคตที่นอกเหนือจากภาษาไพธอนได้จัดเตรียมไว้ให้ใช้งาน และรวมถึงโมดูลอื่นๆ ที่ได้รับการพัฒนาจากผู้พัฒนาโปรแกรมจากคนอื่น ๆ

10.1 การตรวจสอบโมดูลในภาษาไพธอน

โมดูล คือ การรวมกันของคำสั่งโปรแกรมที่ได้เขียนขึ้นมา ซึ่งได้ออกแบบให้ทำหน้าที่อย่างใดอย่างหนึ่ง และภายในโมดูลอาจจะประกอบด้วยคำสั่งโปรแกรม ฟังก์ชัน คลาส จำนวนมาก ขึ้นอยู่กับการออกแบบการทำหน้าที่ ภาษาไพธอนได้จัดเตรียมโมดูลไว้ให้ผู้ใช้งานเรียกใช้งานอย่างหลากหลาย ผู้อ่านสามารถตรวจสอบได้โดยการใช้คำสั่ง `dir()` หรือเข้าไปที่เว็บไซต์ `https://docs.python.org/3/py-modindex.html` แสดงดังตัวอย่าง

ตัวอย่าง 10.1
การตรวจสอบโมดูลภายในภาษาไพธอน

```
import math

print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__',
↪  '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
↪  'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh',
↪  'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1',
↪  'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
↪  'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
↪  'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log',
↪  'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter',
↪  'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin',
↪  'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

10.2 รูปแบบการเรียกใช้งานโมดูลและฟังก์ชัน

ก่อนการใช้งานโมดูลที่ภาษาไพธอนได้จัดเตรียมไว้ หรือจากโมดูลที่ติดตั้งเพิ่มเติม หรือแม้แต่โมดูลที่เราได้สร้างขึ้นมาใช้งานเองก็ตาม เราต้องใช้คำสั่ง **import** ก่อนเสมอ ถึงจะเรียกใช้งานฟังก์ชันในโมดูลนั้น ๆ ได้ ในส่วนนี้จะขอยกตัวอย่างวิธีการใช้คำสั่งเรียกใช้งานโมดูลที่มีอยู่ในภาษาไพธอนด้วยรูปแบบต่าง ๆ ซึ่งแบ่งออกได้เป็น 5 รูปแบบ ดังต่อไปนี้

1. การใช้งานเฉพาะคำสั่ง **import** เมื่อต้องการเรียกใช้งานฟังก์ชันที่อยู่ภายในโมดูล ที่อยู่นอกเหนือจากฟังก์ชัน Built-in ที่ภาษาไพธอนได้จัดเตรียมไว้ให้ เราต้องโหลดโมดูลเข้ามายังโปรแกรมก่อนด้วยคำสั่ง **import** จากนั้นจึงจะอ้างถึงฟังก์ชันภายในโมดูลได้ โดยมีรูปแบบการใช้งานดังนี้

```
import module1, module2, ..., module_n

module1.function_name
module2.function_name
module_n.function_name
```

`import`	คำสั่งโหลดโมดูลเข้ามาใช้งานในโปรแกรม
`module1, ...,` `module_n`	ชื่อโมดูลที่ต้องการโหลดฟังก์ชันมาใช้งาน
`function_name`	ชื่อฟังก์ชันภายในโมดูล

ตัวอย่าง 10.2
การเรียกดูโมดูลและฟังก์ชันภายในโมดูล `math, os, time`

```
1 import math, os, time
2
3 print(math.pow(2, 3))
4 print(os.mkdir(r'new_module_dir'))
5 print(time.time())
```

```
8.0
None
1621090724.8482409
```

2. การใช้คำสั่ง **`from`** ร่วมกับคำสั่ง **`import`** เมื่อต้องการใช้งานเฉพาะฟังก์ชันที่จำเป็นภายในโมดูล จะใช้คำสั่ง **`from`** และตามด้วยชื่อโมดูลที่ต้องการใช้งาน หลังคำสั่ง **`import`** จะเป็นชื่อฟังก์ชันภายในโมดูลนั้นแทน มีรูปแบบการใช้งานดังนี้

```
from module_name import fn_1, ..., fn_n
```

`from`	คำสั่งโหลดโมดูล
`module_name`	ชื่อโมดูลที่ต้องการโหลดฟังก์ชันเข้ามาใช้งานในโปรแกรม
`import`	คำสั่งเรียกใช้ฟังก์ชันจากโมดูล
`fn_1, ..., fn_n`	ชื่อฟังก์ชันภายในโมดูลที่ต้องการเรียกใช้งาน

ตัวอย่าง 10.3
การใช้คำสั่ง `from ... import ...` และวิธีการเรียกใช้งานฟังก์ชันภายในโมดูล

```
1 from math import pi
2 from os import mkdir
3 from time import time
4
5 print(pi)
6 print(mkdir(r'mydirectories'))
7 print(time())
```

```
3.141592653589793
None
1621091504.998086
```

3. การใช้คำสั่ง `from` ร่วมกับคำสั่ง `import *` มีลักษณะการใช้งานคล้ายกับการใช้คำสั่งแบบที่ 1 แต่แตกต่างกันที่วิธีการเขียนคำสั่งเรียกใช้งาน ซึ่งจะเป็นการโหลดฟังก์ชันที่อยู่ในโมดูลทั้งหมดเข้าสู่โปรแกรม วิธีการนี้จะมีการใช้พื้นที่หน่วยความจำมากกว่าแบบที่ 2 ที่เรียกใช้งานเฉพาะฟังก์ชันที่ต้องการเท่านั้น มีรูปแบบการใช้งานดังนี้

```
from module_name import *
```

`from`	คำสั่งเรียกใช้งานโมดูล
`module_name`	ชื่อโมดูลที่ต้องการโหลดฟังก์ชันเข้ามาใช้
`import *`	คำสั่งโหลดฟังก์ชันทั้งหมดจากโมดูล

ตัวอย่าง 10.4
การใช้คำสั่ง `from ... import *` โหลดฟังก์ชันทั้งหมดภายในโมดูล

```
1 from math import *
2 r = 15
3
4 print(pi * (r**2))
5 print(pow(2, 5))
6 print(sin(90))
7 print(sin(pi))
```

```
706.8583470577034
32.0
0.8939966636005579
1.2246467991473532e-16
```

4. การใช้คำสั่ง **import** เปลี่ยนชื่อโมดูลด้วยคีย์เวิร์ด **as** การใช้คีย์เวิร์ด **as** เป็นการเปลี่ยนชื่อโมดูลที่เรียกเข้ามาใช้งานในโปรแกรม มีประโยชน์อย่างมากในขณะที่เขียนคำสั่งโปรแกรม เพราะบางโมดูลมีชื่อยาวหรือมีชื่อที่จำได้ยาก โดยอาจจะเปลี่ยนชื่อโมดูลให้เป็นชื่อใหม่ที่สั้นลงหรือให้จำได้ง่ายขึ้น มีรูปแบบการใช้งานดังนี้

```
import module_name as name
```

`import`	คำสั่งโหลดโมดูลเข้ามาใช้
`module_name`	ชื่อโมดูลที่ต้องการโหลดฟังก์ชันเข้ามาใช้
`as`	คีย์เวิร์ดที่ใช้เปลี่ยนชื่อโมดูลเป็นชื่อใหม่
`name`	ชื่อโมดูลใหม่ที่ตั้งขึ้นมาแทนชื่อโมดูลที่ถูกเรียกใช้

ตัวอย่าง 10.5
การใช้คำสั่ง `import ... as ....` เปลี่ยนชื่อโมดูล

```
1 import math as m
2 r = 15
3
4 print(2 * m.pi * r)
```

```
94.24777960769379
```

5. การใช้คำสั่ง `from` ร่วมกับคำสั่ง `import` และคีย์เวิร์ด `as` ฟังก์ชันที่ถูกเรียกมาใช้งานจากโมดูลที่ถูกโหลดเข้ามาในโปรแกรมสามารถเปลี่ยนชื่อใหม่ได้ เหมือนกับการเปลี่ยนชื่อโมดูลโดยการใช้คีย์เวิร์ด `as` มีรูปแบบการใช้งานดังนี้

```
from module_n import function_n as name
```

`from`	คำสั่งโหลดโมดูลเข้ามาใช้งานในโปรแกรม
`module_n`	ชื่อโมดูลที่ต้องการนำเข้ามาใช้งาน
`import`	คำสั่งเรียกใช้งานฟังก์ชันจากโมดูล
`function_n`	ชื่อฟังก์ชันภายในโมดูลที่โหลดเข้ามาใช้งาน
`as`	คีย์เวิร์ดที่ใช้เปลี่ยนชื่อโมดูลเป็นชื่อใหม่
`name`	ชื่อฟังก์ชันใหม่ที่ตั้งขึ้นมาแทนชื่อฟังก์ชันที่ถูกเรียกใช้งาน

ตัวอย่าง 10.6
การใช้คำสั่ง `from ... import ... as ...` เปลี่ยนชื่อฟังก์ชัน

```
1 from math import pi as PI
2 from math import cos as cosine
3 from math import pow as power
4
5 r = 15
6 print(2 * PI * r)
7 print(cosine(PI))
8 print(power(2, 5))
```

```
94.24777960769379
-1.0
32.0
```

10.3 การสร้างโมดูลขึ้นใช้งาน

นอกเหนือจากโมดูลที่ภาษาไพธอนได้จัดเตรียมไว้ให้ใช้งานที่มีอยู่จำนวนมาก เรายังสามารถสร้างโมดูลขึ้นมาใช้งานเองได้เช่นกัน และเนื่องจากภาษาไพธอนเป็นภาษาที่นำมาใช้งานโดยไม่เสียค่าใช้จ่าย ทำให้มีนักพัฒนาโปรแกรมต่างพากันพัฒนาโมดูลและเผยแพร่ออกมาให้ใช้งานอยู่เสมอ โมดูลมีลักษณะเหมือนกับไฟล์คำสั่งโปรแกรมภาษาไพธอนที่มีนามสกุล `.py` เพียงแต่เราไม่ได้นำมาใช้งานโดยตรง แต่เราจะเขียนคำสั่งโปรแกรมอ้างถึงโมดูลและเรียกใช้งานฟังก์ชันที่อยู่ภายโมดูลแทน ซึ่งมีประโยชน์อย่างมากในกรณีที่มีผู้พัฒนาโปรแกรมหลายคนและเรียกใช้งานฟังก์ชันชื่อเดียวกัน ขั้นตอนการสร้างโมดูลมีดังนี้

1. ก่อนอื่นให้เราตรวจสอบ Default Path ซึ่งเป็นส่วนที่ภาษาไพธอนจะทำการค้นหาโมดูลเมื่อเราสร้างขึ้นมาใช้งานและถูกเรียกใช้

ตัวอย่าง 10.7
การตรวจสอบ Default Path ของการเรียกโมดูล

```
import sys

print(sys.path)
```

```
['/Users/username/Tasks/src', '/Users/username/.pyenv/
↪ versions/3.9.0/lib/python39.zip', '/Users/
↪ username/.pyenv/versions/3.9.0/lib/python3.9', '/Users/
↪ username/.pyenv/versions/3.9.0/lib/python3.9/lib-
↪ dynload', '/Users/username/.pyenv/versions/3.9.0/lib/
↪ python3.9/site-packages']
```

จากตัวอย่างผู้อ่านจะเห็นว่ามี Default Path จำนวนมาก ซึ่งเป็น `path` ที่ภาษาไพธอนจะทำการเรียกโมดูลต่าง ๆ ที่ได้ทำการติดตั้งไว้ หรือเราได้ทำการติดตั้งโมดูลอื่นเพิ่มเติมในภายหลัง เมื่อโหลดโมดูลด้วยการใช้คำสั่ง **`import`** ไพธอนจะไปค้นหาโมดูลที่อ้างถึงจาก path เหล่านี้

2. สร้างไฟล์ใหม่ขึ้นมาพร้อมทั้งเขียนคำสั่งโปรแกรมให้อยู่ในรูปแบบของฟังก์ชัน ในตัวอย่างนี้เป็นการสร้างโมดูลแบบง่ายที่ไม่ซับซ้อน เพื่อให้ผู้อ่านได้มองเห็นภาพและทำความเข้าใจได้ง่าย แสดงตัวอย่างดังภาพ

ตัวอย่าง 10.8
การสร้างไฟล์ใหม่ `my_calculation_module.py` สำหรับทำโมดูล

```
# my_calculation_module.py

def plus(x, y):
    return x + y

def minus(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    return x / y
```

จากตัวอย่างเป็นการสร้างฟังก์ชันการบวก การลบ การคูณ และการหาร โดยมีพารามิเตอร์คอยรับค่าจากโปรแกรมที่เรียกใช้งาน 2 ค่า คือค่า `x` และค่า `y` และส่งค่ากลับไปยังโปรแกรมที่เรียกใช้งานด้วยคำสั่ง **`return`**

3. ให้ตรวจสอบคำสั่งโปรแกรมหรือทดสอบการทำงานเพื่อให้แน่ใจว่า จะไม่เกิดข้อผิดพลาดเมื่อเรียกใช้งาน จากนั้นให้ทำการบันทึกไฟล์โดยใช้ชื่อ `my_calculation_module.py` ไว้ที่ Default Path ที่หามาได้ก่อนหน้านี้ (สมมติว่าตำแหน่งไฟล์คือ `/Users/username/Tasks/src/my_calculation_module.py`)

4. ทดสอบการโหลดโมดูลที่ได้สร้างไว้เข้ามาใช้งานในโปรแกรม ดังตัวอย่างต่อไปนี้

ตัวอย่าง 10.9
การเขียนคำสั่งโหลดโมดูลมาใช้งาน

```
from my_new_module import *

print('ผลลัพธ์จากการบวก 10 กับ 15 = ', plus(10, 15))
print('ผลลัพธ์จากการลบ 45 กับ 15 = ', minus(45, 15))
print('ผลลัพธ์จากการคูณ 10 กับ 15 = ', multiply(10, 15))
print('ผลลัพธ์จากการหาร 45 กับ 15 = ', divide(45, 15))
```

```
ผลลัพธ์จากการบวก 10 กับ 15 = 25
ผลลัพธ์จากการลบ 45 กับ 15 = 30
ผลลัพธ์จากการคูณ 10 กับ 15 = 150
ผลลัพธ์จากการหาร 45 กับ 15 = 3.0
```

5. เราสามารถตรวจสอบฟังก์ชันต่าง ๆ ภายในโมดูลที่สร้างขึ้นมาได้ โดยคำสั่ง `dir()` ดังตัวอย่างต่อไปนี้

ตัวอย่าง 10.10
การตรวจสอบฟังก์ชันภายในโมดูลที่สร้างขึ้นมาใช้งานเอง

```
import my_new_module

print(dir(my_new_module))
```

```
['__builtins__', '__cached__', '__doc__', '__file__',
↪  '__loader__', '__name__', '__package__', '__spec__',
↪  'divide', 'minus', 'multiply', 'plus']
```

10.4 การสร้างแพ็กเกจ

แพ็กเกจ (Package) เหมือนกับไดเรกทอรี (Directory) ซึ่งเป็นที่เก็บรวบรวมชุดโมดูลและไฟล์คำสั่งโปรแกรมที่จะถูกเรียกใช้งานและภายในโมดูลก็จะประกอบไปด้วยฟังก์ชันต่าง ๆ จำนวนมาก ขึ้นอยู่กับว่าจะให้โมดูลนั้นทำงานเกี่ยวกับอะไรหรืออาจจะแยกโมดูลออกมาเป็นโมดูลย่อย ๆ ก็ได้ ดังนั้นเพื่อความเป็นระเบียบหรือเป็นการรวบรวมไฟล์คำสั่งโปรแกรมที่ทำหน้าที่เหมือนกันให้อยู่ที่เดียวกัน อาจจะต้องสร้างแพ็กเกจเก็บโมดูลรวมไว้ที่เดียวกัน ภายในแพ็กเกจจะมีไฟล์ชื่อ `__init__.py` ซึ่งเป็นไฟล์เปล่า ๆ และจะต้องสร้างขึ้นมาเอง โดยมีขั้นตอนดังต่อไปนี้

1. สร้างไดเรกทอรีและตั้งชื่อเป็น `my_package` เก็บไว้ที่

   `/Users/username/Tasks/src`

2. จากนั้นสร้างโมดูลขึ้นมาใหม่อีกหนึ่งโมดูล คือ `my_animal_module.py`

ตัวอย่าง 10.11
การสร้างโมดูล `my_animal_module.py`

```python
# my_animal_module.py

def animals_l():
    animals = ['Dog','Cat', 'Duck', 'Bird', 'Snake']
    return animals

def animals_s():
    animals = {'Bear', 'Zebra', 'Giraffe', 'Buffola',
    ↪  'Rad'}
    return animals

def animals_t():
    animals = {'Chicken', 'Bat', 'Spider', 'Horse',
    ↪  'Mouse'}
    return animals
```

3. นำโมดูลทั้งหมดที่สร้าง ซึ่งก็คือ

   - `my_calculation_module.py` และ
   - `my_animal_module.py`

   บันทึกไว้ที่ไดเรกทอรี `/Users/username/Tasks/src/my_package` และสร้างไฟล์เพิ่มขึ้นอีกหนึ่งไฟล์ชื่อ `__init__.py`

   จากนั้นสร้างไฟล์เปล่า

   - `test_calculation.py` และ
   - `test_animal.py`

   บันทึกไว้ที่ `/Users/username/Tasks/src`

```
src
├── my_module
│   ├── __init__.py
│   ├── my_animal_module.py
│   └── my_calculation_module.py
├── test_calculation.py
└── test_animal.py
```

4. เขียนคำสั่งโปรแกรมที่ไฟล์ `test_calculation.py` และ `test_animal.py` เพื่อโหลดโมดูลมาใช้งานจากแพ็กเกจที่ได้สร้างไว้ตามตัวอย่างต่อไปนี้

ตัวอย่าง 10.12
การเขียนคำสั่งโหลดโมดูล `my_calculation` จากแพ็กเกจ `my_module` และการแสดงผลลัพธ์

```
# test_calculation.py

from my_module.my_calculation_module import *

print('ผลลัพธ์จากการบวก 10 กับ 15 =', plus(10, 15))
print('ผลลัพธ์จากการลบ 45 กับ 15 =', minus(45, 15))
print('ผลลัพธ์จากการคูณ 10 กับ 15 =', multiply(10, 15))
print('ผลลัพธ์จากการหาร 45 กับ 15 =', divide(45, 15))
```

```
ผลลัพธ์จากการบวก 10 กับ 15 = 25
ผลลัพธ์จากการลบ 45 กับ 15 = 30
ผลลัพธ์จากการคูณ 10 กับ 15 = 150
ผลลัพธ์จากการหาร 45 กับ 15 = 3.0
```

ตัวอย่าง 10.13
การเขียนคำสั่งโหลดโมดูล `my_animal` จากแพ็กเกจ `my_module` และการแสดงผลลัพธ์

```
# test_animal.py

from my_module.my_animal_module import *

for animal in animals_l():
    print(animal, end=', ')
print(' ')

for animal in animals_s():
    print(animal, end=', ')
print(' ')

for animal in animals_t():
    print(animal, end=', ')
print(' ')
```

```
Dog, Cat, Duck, Bird, Snake,
Bear, Giraffe, Zebra, Rad, Buffola,
Spider, Horse, Chicken, Mouse, Bat,
```

10.5 โมดูลสำเร็จรูปในภาษาไพธอน

ภาษาไพธอนได้จัดเตรียมโมดูลที่สำคัญไว้ให้ผู้พัฒนาโปรแกรมเรียกใช้งานจำนวนมาก เช่น โมดูล `math`, `os`, `sys`, `time` เป็นต้น โดยที่ผู้พัฒนาโปรแกรมไม่ต้องเสียเวลาติดตั้งเพิ่มเติม ในที่นี้ผู้เขียนจะขอกล่าวถึงโมดูลที่ผู้อ่านน่าจะมีโอกาสได้นำเอามาใช้งานบ่อยครั้งมากที่สุดในการพัฒนาโปรแกรมคือโมดูลคณิตศาสตร์ (Mathematics Module), โมดูลแสดงปฏิทิน (Calendar Module) และโมดูลแสดงเวลา (Time Module)

10.5.1 โมดูลคณิตศาสตร์

โมดูล `math` เป็นหนึ่งในโมดูลที่ภาษาไพธอนได้ติดตั้งไว้ให้เรียกใช้งาน นอกจากนี้ยังมีโมดูล `cmath` ที่ใช้สำหรับการคำนวณจำนวนเชิงซ้อน และภายในโมดูลมีฟังก์ชันต่าง ๆ เช่น ฟังก์ชันการคำนวณพื้นที่รูปทรงเรขาคณิต การหาค่า `log` การหาค่าเลขยกกำลัง เป็นต้น

ตัวอย่าง 10.14
การเขียนคำสั่งตรวจสอบรายชื่อฟังก์ชันในโมดูล `math`

```
import math

print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__',
↪  '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
↪  'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh',
↪  'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1',
↪  'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
↪  'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
↪  'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log',
↪  'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter',
↪  'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin',
↪  'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

ตัวอย่าง 10.15
การเขียนคำสั่งตรวจสอบรายชื่อฟังก์ชันในโมดูล `cmath`

```
1 import cmath
2
3 print(dir(cmath))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__',
↪  '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
↪  'atanh', 'cos', 'cosh', 'e', 'exp', 'inf', 'infj',
↪  'isclose', 'isfinite', 'isinf', 'isnan', 'log', 'log10',
↪  'nan', 'nanj', 'phase', 'pi', 'polar', 'rect', 'sin',
↪  'sinh', 'sqrt', 'tan', 'tanh', 'tau']
```

ฟังก์ชันภายในโมดูล `math` ของภาษาไพธอน สามารถแยกออกเป็นกลุ่มการใช้งานได้ดังต่อไปนี้ ทั้งนี้ผู้อ่านสามารถดูฟังก์ชันที่อาจมีการเพิ่มเติมในอนาคตได้จาก `https://docs.python.org/3/library/math.html`

ฟังก์ชันทางทฤษฎีจำนวน

`math.ceil(x)`
: ส่งกลับค่าจำนวนเต็มที่ที่น้อยที่สุดที่มากกว่าหรือเท่ากับ `x`

`math.comb(n, k)`
: ส่งกลับจำนวนวิธีที่เลือกของ k สิ่ง จากสิ่งของ n สิ่งที่ไม่ซ้ำกัน คำนวณจาก

$$\frac{n!}{k!(n-k)!}$$

เมื่อ $k \leq n$ และ จะส่งค่ากลับเป็น 0 ถ้า $k > n$ ทั้งนี้คำสั่งนี้จะแสดงข้อผิดพลาดประเภท **`TypeError`** ถ้ามีอาร์กิวเมนท์ตัวใดตัวหนึ่งติดลบ

`math.copysign(x, y)`

ส่งกลับข้อมูลชนิด `float` ที่มีค่าเท่ากับค่าสัมบูรณ์ของ `x` แต่มีเครื่องหมายบวกหรือลบตาม `y` และในกรณีที่ระบบรองรับ signed zero คำสั่ง `copysign(1.0, -0.0)` จะส่งกลับค่า `-1.0`

`math.fabs(x)`

ส่งกลับค่าสัมบูรณ์ของ `x`

`math.factorial(x)`

ส่งกลับข้อมูลชนิด `int` ของ

$$x! = x \cdot (x-1) \cdot (x-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

และจะแสดงข้อผิดพลาดประเภท `ValueError` ถ้า `x` ไม่ได้เป็นเป็นข้อมูลชนิด `integral` หรือค่าติดลบ

`math.floor(x)`

ส่งกลับจำนวนเต็มที่มากที่สุดที่มีค่าไม่เกิน `x`

`math.fmod(x, y)`

ส่งกลับค่าเศษจากการหาร `x` ด้วย `y` นิยามโดย

$$x - n \cdot y$$

สำหรับบางจำนวนเต็ม n ที่ทำให้ $x - ny$ มีเครื่องหมายเดียวกับ x และ

$$|x - ny| < |y|$$

ซึ่งมีค่าไม่เท่ากับคำสั่งการหารเอาเศษในภาษาไพธอน โดยทั่วไปแล้วฟังก์ชัน `fmod()` เหมาะสำหรับการใช้งานกับข้อมูลชนิด `float` และคำสั่งการหารเอาเศษในภาษาไพธอน เหมาะสำหรับการใช้งานกับข้อมูลชนิด `int`

ตัวอย่าง 10.16
เปรียบเทียบการหาค่าเศษจากการหารด้วยฟังก์ชัน `fmod()` และคำสั่งการหารเอาเศษในภาษาไพธอน

```
import math

x = -1e-100
y = 1e100
print('fmod(x, y) =  ', math.fmod(x, y))
print('x % y = ', x % y)
```

```
fmod(x, y) =   -1e-100
x % y =  1e+100
```

`math.frexp(x)`

ส่งกลับค่า mantissa และ exponent ของ x ในรูปคู่อันดับ (m, e) เมื่อ m คือข้อมูลชนิด `float` และ e คือข้อมูลชนิด `int` ที่ทำให้

$$x = m \cdot 2^e$$

ถ้า $x = 0$ จะส่งกลับค่า `(0.0, 0)` และในกรณี $x \neq 0$ จะพบว่า

$$0.5 \leq |m| < 1$$

`math.fsum(iterable)`

ส่งกลับข้อมูลชนิด `float` ของค่าผลรวมสมาชิกใน `iterable` แบบความแม่นยำสูงโดยพยายามลดความสูญเสียจากการบวกเลขทศนิยม

ตัวอย่าง 10.17
เปรียบเทียบการหาค่าผลรวมสมาชิกในลิสต์

```
import math

x = [0.1] * 10
print('x = ', x)
print('sum(x) = ', sum(x))
print('math.fsum(x) = ', math.fsum(x))
```

```
x =  [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
sum(x) =  0.9999999999999999
math.fsum(x) =  1.0
```

`math.gcd(*integers)`

ส่งกลับค่า ห.ร.ม. (หารร่วมมาก) ของอาร์กิวเมนท์จำนวนเต็มที่กำหนดให้ (หลังจากภาษาไพธอนเวอร์ชัน 3.9 ฟังก์ชันนี้สามารถรับอาร์กิวเมนท์จำนวนเต็มได้มากกว่า 2 ตัว) ทั้งนี้ฟังก์ชันนี้จะส่งกลับค่า `0` ถ้าอาร์กิวเมนท์จำนวนเต็มที่กำหนดให้เป็น `0` ทั้งหมด หรือ ไม่ระบุอาร์กิวเมนท์

`math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)`

ส่งกลับค่า `True` ถ้าค่าของ `a` และ `b` ใกล้เคียงกัน และส่งกลับค่า `False` ถ้าเป็นอย่างอื่น ในกรณีที่ตรวจไม่พบข้อผิดพลาดใด ๆ จะพบว่า

$$|a - b| \leq \max\{t_r \cdot \max\{|a|, |b|\}, t_a\}$$

เมื่อ t_r คือ `rel_tol` และ t_a คือ `abs_tol`

`math.isfinite(x)`

ส่งกลับค่า `True` ถ้า `abs(x)` ไม่ใช่ทั้ง `math.inf` และ `math.nan` และส่งกลับค่า `False` ในกรณีอื่น ๆ

`math.isinf(x)`

ส่งกลับค่า `True` ถ้า `abs(x)` คือ `math.inf` และส่งกลับค่า `False` ในกรณีอื่น ๆ

`math.isnan(x)`

ส่งกลับค่า `True` ถ้า `x` คือ `math.nan` และส่งกลับค่า `False` ในกรณีอื่น ๆ

`math.isqrt(n)`

ส่งกลับค่า `math.floor(math.sqrt(n))` เมื่อ `n` คือจำนวนเต็มที่ไม่ติดลบ

`math.lcm(*integers)`

ส่งกลับค่า ค.ร.น. (คูณร่วมน้อย) ของอาร์กิวเมนท์จำนวนเต็มที่กำหนดให้ (หลังจากภาษาไพธอนเวอร์ชัน 3.9 ฟังก์ชันนี้สามารถรับอาร์กิวเมนท์จำนวนเต็มได้มากกว่า 2 ตัว) ทั้งนี้ฟังก์ชันนี้จะส่งกลับค่า `0` ถ้าอาร์กิวเมนท์จำนวนเต็มที่กำหนดให้ตัวใดตัวหนึ่งเป็น `0` และจะส่งกลับค่า `1` ถ้าไม่ระบุอาร์กิวเมนท์

`math.ldexp(x, i)`

ส่งกลับค่า `x * (2**i)`

`math.modf(x)`

ส่งกลับคู่อันดับ `(f, i)` เมื่อ `i` คือข้อมูลชนิด `float` แทนส่วนที่เป็นจำนวนเต็มของ `x` และ `f` คือข้อมูลชนิด `float` แทนส่วนที่เป็นหลังจุดทศนิยมของ `x`

`math.perm(n, k=None)`

ส่งกลับจำนวนวิธีการเรียงสับเปลี่ยนของจำนวน k สิ่ง จากของ n สิ่งที่แตกต่างกัน คำนวณจาก

$$\frac{n!}{(n-k)!}$$

เมื่อ $k \leq n$ และฟังก์ชันนี้จะส่งกลับค่า `0` ถ้า $k > n$ ในกรณีที่ k ไม่ถูกระบุหรือเป็น `None` ฟังก์ชันนี้จำกำหนดให้ $k = n$ ทั้งนี้คำสั่งนี้จะระบุข้อผิดพลาดประเภท `TypeError` หากมีอาร์กิวเมนท์ที่ไม่ใช่จำนวนเต็ม และจะระบุข้อผิดพลาดประเภท `ValueError` หากมีอาร์กิวเมนท์ที่ติดลบ

`math.prod(iterable, *, start=1)`

ส่งกลับค่าผลคูณของสมาชิกใน `iterable` หากไม่ระบุค่า `start` จะให้ `start = 1` ถ้า `iterable` ไม่มีสมาชิก ฟังก์ชันนี้จะส่งกลับค่า `start` ทั้งนี้ ฟังก์ชันนี้อาจปฏิเสธสมาชิกใน `iterable` ที่ไม่เป็นตัวเลข

ฟังก์ชันชี้กำลังและลอกาลิทึม

ตัวอย่าง 10.18
การเขียนคำสั่งเรียกใช้งานกลุ่มฟังก์ชันการคำนวณและการแทนค่า

```
import math

print('ค่าจำนวนเต็มของ 15.4588 =', math.ceil(15.4588))
print('แปลงค่าจำนวนเต็มลบ -15 =', math.copysign(-15, 0.0))
print('ค่า factorial(10) =', math.factorial(10))
print('ค่า ldexp(20, 5) =', math.ldexp(20, 5))

lst = [1.5, 3.5, 5.5, 1.2, 4.5]
print('ค่า fsum(lst) = ', math.fsum(lst))
```

```
ค่าจำนวนเต็มของ 15.4588 = 16
แปลงค่าจำนวนเต็มลบ -15 = 15.0
ค่า factorial(10) = 3628800
ค่า ldexp(20, 5) = 640.0
ค่า fsum(lst) =  16.2
```

ตัวอย่าง 10.19
การเขียนคำสั่งเรียกใช้งานกลุ่มฟังก์ชันเลขยกกำลังและลอการิทึมของโมดูล math

```
import math
print('ค่า expm1(5.5) =', math.expm1(5.5))
print('ค่า log(5) =', math.log(5))
print('ค่า log10(2) =', math.log(2))
print('ค่า log1p(2) =', math.log1p(2))
print('ค่า log2(10) =', math.log2(10))
print('ค่า pow(10, 3) =', math.pow(10, 3))
print('ค่า sqrt(27) =', math.sqrt(27))
```

```
ค่า expml(5.5) = 243.69193226422038
ค่า log(5) = 1.6094379124341003
ค่า log10(2) = 0.6931471805599453
ค่า log1p(2) = 1.0986122886681098
ค่า log2(10) = 3.321928094887362
ค่า pow(10, 3) = 1000.0
ค่า sqrt(27) = 5.196152422706632
```

10.5.2 โมดูลแสดงผลวันที่และเวลา

การแสดงผลวันที่และเวลาเป็นส่วนหนึ่งที่สำคัญในการพัฒนาโปรแกรม ซึ่งผู้เขียนจะขอกล่าวถึงโมดูล `calendar` และโมดูล `time` และแสดงตัวอย่างการเรียกใช้งานฟังก์ชันบางตัวที่มีอยู่ภายในทั้งสองโมดูล

โมดูลปฏิทิน (Calendar Module)

โมดูล `calendar` ใช้สำหรับแสดงผลวัน เดือน ปี เราสามารถเรียกใช้งานฟังก์ชันและแอตทริบิวต์ได้ตามตารางต่อไปนี้

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้และคำอธิบาย
`calendar()`	แสดงผลปฏิทินตามปี ค.ศ. ที่กำหนด	`calendar.calendar(year, w=1, l=1, c=3)` โดย `year` คือ ปีที่ต้องการแสดงผล, `w` คือ ระยะช่องไฟการแสดงผลของวันที่, `l` คือ ระยะบรรทัดของแต่ละสัปดาห์, `c` คือ ระยะห่างของแต่ละเดือน
`firstweekday()`	แสดงวันเริ่มต้นในสัปดาห์ค่าเริ่มต้นเป็น 0 คือวันจันทร์ และ 6 คือวันอาทิตย์	`calendar.firstweekday()`
`isleap()`	ตรวจสอบปีที่มี 366 วัน ถ้าใช่คือค่า `True` ถ้าไม่ใช่คืนค่า `False`	`calendar.isleap(year)` `year` คือ ปีที่ต้องการตรวจสอบว่ามี 366 วัน (leap year) หรือไม่
`leapdays()`	แสดงจำนวนตัวเลขปีที่มี 366 วัน โดยกำหนดพารามิเตอร์เป็นช่วงของปี	`calendar.leapdays(y1, y2)` `y1` คือ ปีเริ่มต้น, `y2` คือ ปีสิ้นสุด
`month()`	แสดงเดือนที่ปฏิทินตามปี ค.ศ.ที่กำหนด	`calendar.month(year, month, w=2, l=1)` `year` คือ ปีที่ต้องการแสดงผล, `month` คือ เดือนที่ต้องการแสดงผล, `w` คือ ระยะช่องไฟการแสดงผลของวันที่, `l` คือ ระยะบรรทัดของแต่ละสัปดาห์

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้และคำอธิบาย
`monthcalendar()`	แสดงเดือนในปฏิทินปี ค.ศ. ที่กำหนด แสดงผลในรูปแบบลิสต์	`calendar.monthcalendar(year, month)` `year` คือ ปีที่ต้องการแสดงผล, `month` คือเดือนที่ต้องการแสดงผล
`monthrange()`	แสดงรหัสวันเริ่มต้นของเดือน 0 คือวันจันทร์ และ 6 คือวันอาทิตย์ และจำนวนวันของแต่ละเดือน	`calendar.monthrange(year, month)` `year` คือ ปีที่ต้องการแสดงรหัสวันเริ่มของเดือน, `month` คือเดือนที่ต้องการแสดงรหัสวันเริ่มต้น
`prcal()`	แสดงผลปฏิทินตามปี พ.ศ. หรือ ค.ศ. เหมือนกับฟังก์ชัน `calendar`	`calendar.prcal(year, w=2, l=1, c=6)`
`prmonth()`	แสดงเดือนในปฏิทินตามปี พ.ศ. หรือค.ศ. เหมือนกับฟังก์ชัน `month`	`calendar.prcal(year, w=2, l=1, c=6)`
`setfirstweekday()`	กำหนดวันเริ่มต้นของสัปดาห์ กำหนดตามรหัสวัน 0 คือ วันจันทร์ และ 6 คือวันอาทิตย์	`calendar.setfirstweekday(weekday)` `weekday` คือ รหัสวันในสัปดาห์กำหนดเป็น 0-6

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้และคำอธิบาย
`timegm()`	แสดงค่าเวลาเริ่มจากปี ค.ศ. 1970 จากฟังก์ชัน `gmtime()` ของโมดูล `time`	`calendar.timegm(tuple)` `tuple` คือ รูปแบบเวลาที่ได้จากการคืนค่ากลับมาจากฟังก์ชัน `gmtime()`
`weekday()`	แสดงรหัสวันในสัปดาห์จากการกำหนดปี ค.ศ. เดือน และวันที่	`calendar.weekday(year, month, day)` `year` คือ ปีที่ต้องการแสดงรหัสวันกำหนดได้ตั้งแต่ปี 1970 ถึงปีปัจจุบัน, `month` คือ เดือนที่ต้องการแสดงรหัสวันกำหนดเป็น 1-12, `day` คือ วันที่ต้องการแสดงรหัสวันกำหนดเป็น 1-31

ชื่อแอตทริบิวต์	ความหมาย	รูปแบบการใช้และคำอธิบาย
day_name	ใช้สำหรับแสดงชื่อวันแบบเต็ม	calendar.day_name
day_abbr	แสดงชื่อวันรูปแบบย่อ	calendar.day_abbr
month_name	สำหรับแสดงชื่อเดือนแบบเต็ม	calendar.month_name
month_abbr	แสดงชื่อเดือนรูปแบบย่อ	calendar.month_abbr

ตาราง 10.2: แอตทริบิวต์ภายในโมดูล Calendar

ตัวอย่าง 10.20
การเขียนคำสั่งเรียกใช้งานฟังก์ชันภายในโมดูล calendar

```
import calendar
print('วันเริ่มต้นในสัปดาห์ =', calendar.firstweekday())
print('ปี 2017 มี 366 วัน =', calendar.isleap(2017))
print('วันแรกและจำนวนวัน เดือนมกราคม ปี 2017 =',
    calendar.monthrange(2017, 1))
print('ปี 2017 เดือน 1 วันที่ 31 ตรงรหัสวัน =', calendar.weekday(2017, 1,
    31))
```

```
วันเริ่มต้นในสัปดาห์ = 0
ปี 2017 มี 366 วัน = False
วันแรกและจำนวนวัน เดือนมกราคม ปี 2017 = (6, 31)
ปี 2017 เดือน 1 วันที่ 31 ตรงรหัสวัน = 1
```

ตัวอย่าง 10.21
การเขียนคำสั่งเรียกใช้แอตทริบิวต์ภายในโมดูล `calendar`

```
import calendar

for day in calendar.day_name:
    print(day, end=' ')
print()
print(list(calendar.day_abbr))
```

```
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

โมดูลเวลา (Time Module)

โมดูล `time` ใช้สำหรับแสดงผลเวลา ในส่วนนี้จะขอแนะนำฟังก์ชันบางตัว หากผู้อ่านอยากทราบว่าในโมดูล `time` มีฟังก์ชันอะไรบ้าง ให้ใช้คำสั่ง `help(time)` ตรวจสอบฟังก์ชันได้ หรือเข้าไปที่เว็บไซต์ `https://docs.python.org.3/library/time.html?highlight=time#module-time` มีฟังก์ชันให้เรียกใช้งานตามตารางต่อไปนี้

ชื่อฟังก์ชัน	ความหมาย	รูปแบบการใช้
`asctime()`	แสดงวัน เดือน ปี และเวลา	`time.asctime([t])` `t` คือ ค่าที่ได้จากฟังก์ชัน `gmtime()` หรือ `localtime()`
`clock()`	แสดงผลเวลาประมวลผลของ CPU แสดงผลออกมาเป็นหน่วยวินาที	`time.clock()`
`ctime()`	แสดงเวลาปัจจุบันตาม location	`time.ctime([secs])` `secs` คือ วินาทีเริ่มนับจากปี 1970 กำหนดหรือไม่ก็ได้
`gmtime()`	แสดงเวลาปัจจุบันในรูปแบบโซนเวลา UTC (Coordinated Universal Time - UTC หรือ Greenwich Mean Time - GMT)	`time.ctime([secs])` `secs` คือ วินาทีเริ่มนับจากปี 1970 กำหนดหรือไม่ก็ได้
`localtime()`	แสดงเวลาปัจจุบันในรูปแบบ `struct_time` ตามโซนเวลาของผู้ใช้งาน	`time.ctime([secs])` `secs` คือ วินาทีเริ่มนับจากปี 1970 กำหนดหรือไม่ก็ได้
`sleep()`	ใช้สำหรับหน่วงเวลา	`time.sleep([t])` `t` คือ เวลาที่ต้องการหน่วงมีหน่วยเป็นวินาที

`strftime()`	จัดรูปแบบแสดงผลเวลาของฟังก์ชัน `gmtime()` และ `localtime()`	`time.strftime(format[, t])`
		`format` คือ การจัดรูปแบบแสดงผลเวลา
		`t` คือ ฟังก์ชัน `gmtime()` หรือ `localtime()` ถ้าไม่กำหนดจะแสดงเวลาที่ได้จากฟังก์ชัน `localtime()`

ตาราง 10.3: ฟังก์ชันภายในโมดูล `time`

ตัวอย่าง 10.22
การเขียนคำสั่งเรียกใช้แอตทริบิวต์ภายใน โมดูล calendar

```
import time

print(time.asctime())
print(time.asctime(time.gmtime()))
print(time.asctime(time.localtime()))
print(time.strftime('5a %d %b %Y %H:%M:%S'))
print(time.strftime('5a %d %b %Y %H:%M:%S, time.gmtime()))'
```

```
Tue Sep 10 16:52:49 2019
Tue Sep 10 09:52:49 2019
Tue Sep 10 16:52:49 2019
5a 10 Sep 2019 16:52:49
5a 10 Sep 2019 09:52:49
```

ตัวอย่าง 10.23
การเขียนคำสั่งเรียกใช้งานฟังก์ชัน sirftime() ในโมดูล time

```
import time

print(time.strftime('%c %Z'))
print(time.strftime('%a, %d %b %Y %H:%M:%S %Z', time.gmtime()))
```

```
Tue Sep 10 16:53:23 2019 SE Asia Standard Time
Tue, 10 Sep 2019 09:53:23 SE Asia Standard Time
```

สัญลักษณ์	ความหมาย
%a	แสดงชื่อของวัน เช่น Sun, Mon, Tue, ...
%A	แสดงชื่อของวัน เช่น Sunday, Monday, Tueday, ...
%b	แสดงชื่อของเดือน เช่น Jan, Feb, Mar, ...
%B	แสดงชื่อของเดือน เช่น January, February, March, ...
%c	แสดงวัน เดือน ปี และเวลา ตามตำแหน่งที่อยู่ของผู้เรียกใช้
%d	แสดงวันที่ของเดือน [01-31]
%H	แสดงตัวเลขชั่วโมงในรูปแบบเวลา 24 ชั่วโมง [00-24]
%I	แสดงตัวเลขชั่วโมงในรูปแบบเวลา 12 [01-12]
%j	แสดงจำนวนวันของปี ณ ปัจจุบัน [001-366]
%m	แสดงตัวเลขเดือน [01-12]
%M	แสดงตัวเลขนาที [00-59]
%p	แสดงเวลาแบบ AM หรือ PM ตามตำแหน่งของผู้ใช้งาน
%S	แสดงตัวเลขวินาที [00-61]
%U	แสดงสัปดาห์ปัจจุบันของปี [00-53] โดยที่วันอาทิตย์เป็นวันแรกของสัปดาห์
%w	แสดงตัวเลขประจำวัน 0 คือ วันอาทิตย์ และ 6 คือวันเสาร์
%W	แสดงสัปดาห์ปัจจุบันของปี [00-53] โดยที่วันจันทร์เป็นวันแรกของสัปดาห์
%x	แสดงรูปแบบวัน/เดือน/ปี [09/11/17]
%X	แสดงเวลารูปแบบ H:M:S [11:13:48]
%y	แสดงปี ค.ศ. แบบย่อ
%Y	แสดงปี ค.ศ. แบบเต็ม
%z	แสดงโซนเวลา
%Z	แสดงชื่อโซนเวลา
%%	แสดงเครื่องหมาย %

ตาราง 10.4: สัญลักษณ์การจัดรูปแบบแสดงผลเวลา

สรุปท้ายบท

ในบทนี้ผู้อ่านได้รู้จักวิธีการสร้างโมดูลและแพ็กเกจไว้ใช้งาน ซึ่งภายในโมดูลประกอบด้วยฟังก์ชัน คลาสที่ทำหน้าที่แตกต่างกันออกไปตามที่กำหนด และภายในแพ็กเกจก็จะประกอบด้วยโมดูลที่สร้างขึ้นมา อย่างไรก็ตามผู้อ่านต้องสร้างไฟล์ `__init__.py` ไว้ภายในแพ็กเกจด้วยถึงจะเรียกใช้งานได้ นอกจากนี้ยังได้รู้จักกับโมดูลมาตรฐานของภาษาไพธอนที่ได้จัดเตรียมไว้ให้ผู้ใช้เรียกใช้งาน คือ โมดูล `math` และ `cmath` ซึ่งเป็นโมดูลทางด้านการคำนวณคณิตศาสตร์ รวมทั้งโมดูลการจัดการแสดงผลวัน เดือน ปี และเวลาคือ โมดูล `calendar` และ `time`

แบบฝึกหัด

1. จงเขียนโปรแกรมสร้างโมดูลคิดอัตราส่วนลดสินค้า จากการขายสินค้าจำนวน 3 อย่าง ถ้าลูกค้าซื้อสินค้ารวมเป็นเงิน 1500 บาทขึ้นไป ได้ส่วนลด 5% ถ้าลูกค้าซื้อสินค้ารวมเป็นเงิน 2500 บาทขึ้นไป มีส่วนลด 10% ถ้าลูกค้าซื้อสินค้ารวมเป็นเงินมากกว่า 5000 บาทขึ้นไป ได้ส่วนลด 15% โดยให้ป้อนราคาสินค้าทั้ง 3 อย่างผ่านทางคีย์บอร์ด พร้อมทั้งให้แสดงราคาสินค้าแต่ละชิ้นที่ลูกค้าซื้อ ราคารวม ราคาส่วนลด และราคาที่ต้องจ่ายทั้งหมด

2. จงเขียนโปรแกรมคำนวณค่าอาหารบุฟเฟต์ คิดอัตราผู้ใหญ่คนละ 199 บาท เด็กคนละ 159 บาท บันทึกลงไฟล์ ภายในไฟล์ต้องบอกวันที่และเวลา จำนวนคน โดยให้แยกจำนวนเด็กและผู้ใหญ่ที่เข้ามารับประทานที่ร้าน