บทที่ 4

ชนิดข้อมูลสตริง

ในบทที่ผ่านมาเราได้รู้จักกับชนิดข้อมูลพื้นฐาน เช่น ชนิดข้อมูลจำนวนเต็ม ชนิดข้อมูลจำนวนทศนิยม ชนิดข้อมูลตรรกะ และชนิดข้อมูลสตริงไปบ้างแล้ว สำหรับชนิดข้อมูลสตริงเป็นชนิดข้อมูลที่มีความสำคัญมากอีกหนึ่งชนิดข้อมูล เพราะการเขียนโปรแกรมต้องมีการดำเนินการกับสตริงหรือข้อความอยู่เสมอ เช่น การตัดข้อความ การเชื่อมต่อข้อความ การแทนที่ข้อความ เป็นต้น บทนี้ผู้อ่านจะได้รู้จักวิธีการจัดการกับชนิดข้อมูลสตริง รวมทั้งการเรียกใช้งานเมธอด (Method) ต่าง ๆ ที่นำมาจัดการกับชนิดข้อมูลสตริง

4.1 รู้จักกับชนิดข้อมูลสตริง

ชนิดข้อมูลสตริง (String) สามารถเป็นได้ทั้งตัวอักขระ ข้อความหรือประโยคที่ประกอบด้วยตัวอักขระหลาย ๆ ตัวที่นำมาต่อกันในภาษาไพธอนการประกาศสร้างตัวแปรเก็บชนิดข้อมูลสตริงจะอยู่ในเครื่องหมาย **(** `'...'` **)** หรือ **(** `"..."` **)** สำหรับภาษาไพธอนเวอร์ชัน 3 ได้เปลี่ยนการจัดเก็บสตริงจาก ASCII ที่ใช้ในภาษาไพธอนเวอร์ชัน 2 มาเป็นแบบ Unicode ซึ่งรองรับการจัดเก็บภาษาทั่วโลกได้มากกว่า ตัวอย่างการประกาศตัวแปรชนิดข้อมูลสตริง แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.1
การเขียนคำสั่งโปรแกรมประกาศตัวแปรเก็บชนิดข้อมูลสตริง

```
# ใช้เครื่องหมาย '...' เป็นชนิดข้อมูลสตริง
str1 = 'Hello Python'

# ใช้เครื่องหมาย "..." เป็นชนิดข้อมูลสตริง
str2 = "Hello World"

print(str1) # แสดงผลค่าตัวแปร str1
print(str2)  # แสดงผลค่าตัวแปร str2
```

```
Hello Python
Hello World
```

หากต้องการแปลงชนิดข้อมูลใด ๆ ให้เป็นชนิดข้อมูลสตริง ให้เรียกใช้งานฟังก์ชัน `str()` ดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.2
การแปลงชนิดข้อมูลจำนวนเต็มเป็นชนิดข้อมูลสตริง ด้วยฟังก์ชัน `srt()`

```
num = 245 # ประกาศตัวแปรชนิดข้อมูลจำนวนเต็ม
str1 = str(num) # แปลงค่าตัวแปร num เป็นชนิดข้อมูลสตริง
print(str1) # แสดงผลค่าตัวแปร str1
print('ชนิดข้อมูล num คือ', type(num))  # แสดงประเภทข้อมูลค่าตัวแปร num
print('ชนิดข้อมูลของ str1 คือ', type(str1))  # แสดงประเภทข้อมูลค่าตัวแปร
↪  str1
```

```
245
ชนิดข้อมูล num คือ <class 'int'>
ชนิดข้อมูลของ strl คือ <class 'str'>
```

หลังจากมีการแปลงชนิดข้อมูลจำนวนเต็มเป็นชนิดข้อมูลสตริงแล้ว และนำตัวดำเนินการคณิตศาสตร์มาดำเนินการกับชนิดข้อมูลทั้งสองจะทำให้เกิดการแจ้งเตือนข้อผิดพลาดขึ้น แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.3
การเขียนคำสั่งโปรแกรมดำเนินการกับชนิดข้อมูลสตริง

```
num = 347
num1 = 245
str1 = str(num) # แปลงค่าตัวแปร num เป็นชนิดข้อมูลสตริง
result = str1 + num1 # บวกค่าตัวแปร str1 กับ num1
print(result)  # แสดงผลลัพธ์ตัวแปร result
```

```
TypeError Traceback (most recent call last)
<ipython-input-1-06da36824501> in <module>
1 num = 347
2 num1 = 245
3 str1 = str(num) # แปลงค่าตัวแปร num เป็นชนิดข้อมูลสตริง
4 result = str1 + num1 # บวกค่าตัวแปร str1 กับ num1
5 print(result) # แสดงผลลัพธ์ตัวแปร result

TypeError: can only concatenate str (not 'int') to str
```

จากตัวอย่างโปรแกรมจะเกิดการแจ้งเตือนข้อผิดพลาดขึ้น เนื่องจากค่าตัวแปร num ซึ่งถูกแปลงเป็นชนิดข้อมูลสตริง ซึ่งไม่สามารถนำมาดำเนินการบวกกับค่าตัวแปรnum1 ซึ่งเป็นชนิดข้อมูลจำนวนเต็มได้

4.2 การเชื่อมต่อสตริง

เมื่อต้องการเชื่อมชนิดข้อมูลสตริงเข้าด้วยกัน สามารถนำค่าตัวแปรมาต่อกันในฟังก์ชัน `print()` แล้วคั่นด้วยเครื่องหมาย Comma `(,)` หรือใช้ตัวดำเนินการ `(+)` เชื่อมต่อ

สตริงเข้าด้วยกัน และเมื่อต้องการเว้นวรรคระหว่างข้อความให้ใช้สตริงช่องว่าง ' ' (White Space String) แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.4
การเขียนคำสั่งโปรแกรมเชื่อมชนิดข้อมูลสตริงเข้าด้วยกัน

```
str1 = 'Python'
str2 = 'is a programming'
str3 = 'language.'
print(str1, str2, str3) # แสดงผลลัพธ์จากตัวแปร
print(str1 + ' ' + str2 + ' ' + str3) # เชื่อมข้อมูลจากค่าตัวแปรเข้าด้วยกัน
```

```
Python is a programming language.
Python is a programming language.
```

กรณีที่เราต้องการทำซ้ำข้อความ ให้ใช้ตัวดำเนินการ * แล้วกำหนดจำนวนครั้งที่ต้องการทำซ้ำ แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.5
การเขียนคำสั่งโปรแกรมทำซ้ำชนิดข้อมูลสตริง ตามจำนวนครั้งที่กำหนด

```
str1 = 'Python '
star = '*'
print(str1 * 3) # แสดงผลลัพธ์ตัวแปร str1 จำนวน 3 ครั้ง
print(3 * str1) # แสดงผลลัพธ์ตัวแปร str1 จำนวน 3 ครั้ง
print (3 * star) # แสดงผลลัพธ์ตัวแปร star จำนวน 3 ครั้ง
```

```
Python Python Python
Python Python Python
***
```

การหาจำนวนอักขระที่อยู่ในข้อความให้เรียกใช้งานด้วยฟังก์ชัน `len()` แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.6
การเขียนคำสั่งโปรแกรมแสดงจำนวนอักขระในข้อความ ด้วยฟังก์ชัน `len()`

```
str1 ='Python'
str2 = 'Python '
str3 = ''
str4 = ' '
str5 = 'IXOHOXI'
print('จำนวนอักขระใน str1 =', len(str1)) # แสดงจำนวนอักขระของ str1
print('จำนวนอักขระใน str2 =', len(str2)) # แสดงจำนวนอักขระของ str2
print('จำนวนอักขระใน str3 =', len(str3)) # แสดงจำนวนอักขระของ str3
print('จำนวนอักขระใน str4 =', len(str4)) # แสดงจำนวนอักขระของ str4
print('จำนวนอักขระใน str5 =', len(str5)) # แสดงจำนวนอักขระของ str5
```

```
จำนวนอักขระใน str1 = 6
จำนวนอักขระใน str2 = 7
จำนวนอักขระใน str3 = 0
จำนวนอักขระใน str4 = 1
จำนวนอักขระใน str5 = 7
```

จากตัวอย่างจะเห็นว่าตัวแปร `str3` และ `str4` ไม่มีตัวอักขระใด ๆ แต่ผลลัพธ์ที่ได้จากตัวแปร `str4` เท่ากับ `1` เนื่องจากเก็บค่าช่องว่างเอาไว้

4.3 การเข้าถึงตำแหน่งตัวอักขระในชนิดข้อมูลสตริง

การเข้าถึงตัวอักขระในชนิดข้อมูลสตริง (Indexing String Operator) จะใช้เครื่องหมาย `[ ]` โดยการระบุตำแหน่งหรือ Index ตำแหน่งแรกเริ่มต้นที่ 0 หากต้องการเข้าถึงตำแหน่งตัวอักขระจากด้านท้ายสุดตำแหน่งแรกจะเป็น -1 แสดงตำแหน่งการเก็บตัวอักขระในสตริงดังต่อไปนี้

Negative Index	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
`str =`	H	E	L	L	O		P	Y	T	H	O	N
Positive Index	0	1	2	3	4	5	6	7	8	9	10	11

การระบุตำแหน่งที่ต้องการ ผลลัพธ์ที่ได้จะแสดงตัวอักขระที่อยู่ ณ ตำแหน่งที่ได้ระบุไว้ มีรูปแบบการใช้งานดังนี้

```
str[index]
```

`str` ตัวแปรชนิดข้อมูลสตริง
`index` ตำแหน่งที่ต้องการแสดงตัวอักขระ

ถ้ากำหนดให้ `str = 'HELLO PYTHON'` เราสามารถเข้าถึงอักขระตำแหน่งแรกของตัวแปร `str` ด้วยคำสั่ง `str[0]` ซึ่งก็คือ `'H'` นั่นเอง ในทำนองเดียวกันเราสามารถเข้าถึงอักขระตำแหน่งสุดท้ายและตำแหน่งรองสุดท้ายของตัวแปร `str` ด้วยคำสั่ง `str[-1]` และ `str[-2]` ซึ่งก็คือ `'N'` และ `'O'` ตามลำดับ และอักขระของตัวแปร `str` ที่มี Index เป็น 5 และ -7 คือ สตริงช่องว่าง (White Space String) ทั้งนี้ภาษาโปรแกรมไพธอนไม่อนุญาตให้เราเข้าถึงอักขระของตัวแปร `str` ที่มี Index สูงกว่า `len(str)` ซึ่งก็คือ 11 และ Index ต่ำกว่า `-len(str)-1` ซึ่งก็คือ -12

ตัวอย่าง 4.7
การเขียนคำสั่งโปรแกรมแสดงตัวอักขระโดยการระบุตำแหน่ง

```
str1 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  # สร้างตัวแปรชนิดข้อมูลสตริง
print('ตำแหน่งที่ 0 คือ', str1[0])  # แสดงตัวอักขระในตำแหน่งที่ 0
print('ตำแหน่งที่ -1 คือ', str1[-1])  # แสดงตัวอักขระในตำแหน่งที่ -1
print('ตำแหน่งที่ 5 คือ', str1[5])  # แสดงตัวอักขระในตำแหน่งที่ 5
print('ตำแหน่งที่ -5 คือ', str1[-5])  # แสดงตัวอักขระในตำแหน่งที่ -5
print('ตำแหน่งที่ -15 คือ', str1[-15])  # แสดงตัวอักขระในตำแหน่งที่ -15
```

```
ตำแหน่งที่ 0 คือ A
ตำแหน่งที่ -1 คือ Z
ตำแหน่งที่ 5 คือ F
ตำแหน่งที่ -5 คือ V
ตำแหน่งที่ -15 คือ L
```

ตัวอย่าง 4.8
การเข้าถึงอักขระโดยการระบุตำแหน่งนอกขอบเขต

```
str1 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
n = len(str1)
print(str1[n+1])  #Index out of range
print(str1[-n-1])  #Index out of range too
```

```
Traceback (most recent call last):
  File "/Users/user/src/test.py", line 3, in <module>
    print(str1[n+1])
IndexError: string index out of range
```

การระบุตำแหน่งเริ่มต้นและตำแหน่งสุดท้าย ผลลัพธ์ที่ได้จะแสดงตัวอักขระตั้งแต่ตำแหน่งเริ่มต้นที่ได้ระบุจนถึงตำแหน่งสุดท้าย มีรูปแบบการใช้ดังนี้

```
str[start:stop]
```

`str`	ตัวแปรชนิดข้อมูลสตริง
`start`	ตำแหน่งเริ่มต้นที่ต้องการแสดงตัวอักขระ
`stop`	ตำแหน่งสุดท้ายที่ต้องการแสดงตัวอักขระ

ตัวอย่าง 4.9
การเขียนคำสั่งโปรแกรมแสดงตัวอักขระโดยการระบุตำแหน่งเริ่มต้นและตำแหน่งสุดท้าย

```
1 str1 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
2 print('ผลลัพธ์ที่ได้จากการตัดคำ 0:4 =', str1[0:4]) # แสดง str1 ตัวที่ 0 ถึง 4
3 print('ผลลัพธ์ที่ได้จากการตัดคำ -6:-1 =', str1[-6:-1]) # แสดง str1 ตัวที่ -6
  ↪ ถึง-1
4 print('ผลลัพธ์ที่ได้จากการตัดคำ 6:12 =', str1[6:12]) # แสดง str1 ตัวที่ 6 ถึง
  ↪ 12
5 print('ผลลัพธ์ที่ได้จากการตัดคำ 0:5 =', str1[:5]) # แสดง str1 ตัวที่ 0 ถึง 5
6 print('ผลลัพธ์ที่ได้จากการตัดคำ =', str1[:]) # แสดงอักขระ str1 ทั้งหมด
```

```
ผลลัพธ์ที่ได้จากการตัดคำ 0:4 = ABCD
ผลลัพธ์ที่ได้จากการตัดคำ -6:-1 = UVWXY
ผลลัพธ์ที่ได้จากการตัดคำ 6:12 = GHIJKL
ผลลัพธ์ที่ได้จากการตัดคำ 0:5 = ABCDE
ผลลัพธ์ที่ได้จากการตัดคำ = ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

การระบุตำแหน่งเริ่มต้น ระบุตำแหน่งสุดท้าย และการก้าวกระโดด ผลลัพธ์ที่ได้จะแสดงตัวอักขระตั้งแต่ตำแหน่งเริ่มต้น แต่จะมีการกระโดดข้ามตัวอักขระตามที่กำหนดไว้จนถึงตำแหน่งสุดท้าย มีรูปแบบการใช้งานดังนี้

```
str[start:stop:step]
```

`str`	ตัวแปรชนิดข้อมูลสตริง
`start`	ตำแหน่งเริ่มต้นที่ต้องการแสดงตัวอักขระ
`stop`	ตำแหน่งสุดท้ายที่ต้องการแสดงตัวอักขระ
`step`	จำนวนการกระโดดข้ามตำแหน่ง

ตัวอย่าง 4.10
การเขียนคำสั่งโปรแกรมโดยการระบุตำแหน่งเริ่มต้น ตำแหน่งสุดท้าย และการก้าวกระโดด

```
1 str1 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
2
3 # แสดงตัวอักขระจากตำแหน่งที่ -12 ถึง -1 โดยกระโดดครั้งละ 2 ตำแหน่ง
4 print('ผลลัพธ์ที่ได้จากการตัดคำ -12::2 =', str1[-12::2])
5
6 # แสดงตัวอักขระจากตำแหน่งที่ 2 ถึงตำแหน่งสุดท้าย โดยกระโดดครั้งละ 2 ตำแหน่ง
7 print('ผลลัพธ์ที่ได้จากการตัดคำ 1::2 =', str1[1::2])
```

```
ผลลัพธ์ที่ได้จากการตัดคำ -12::2 = OQSUWY
ผลลัพธ์ที่ได้จากการตัดคำ 1::2 = BDFHJLNPRTVXZ
```

4.4 การควบคุมการแสดงผลออกทางจอภาพ

โดยทั่วไปการแสดงผลลัพธ์ออกทางจอภาพ (Control Output) จะเรียกใช้งานฟังก์ชัน `print()` และข้อความที่อยู่ในเครื่องหมาย Single Quote (`'...'`) หรือ Double Quote (`"..."`) ถ้าข้อความมีขนาดความยาวจนต้องขึ้นบรรทัดใหม่ หรือต้องการให้แสดงผลออกทางหน้าจอหลายบรรทัด ให้ใช้เครื่องหมาย Triple Quote (`'''...'''` หรือ `'''...'''`) แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.11
การเขียนคำสั่งโปรแกรมแสดงผลข้อความแบบหลายบรรทัด

```
1 str1 = '''Python is  simple and easy to learn.
2 Python is  simple and easy to learn.
3 Python is  simple and easy to learn.'''
4 print(str1)
```

```
Python is  simple and easy to learn.
Python is  simple and easy to learn.
Python is  simple and easy to learn.
```

ถ้ามีข้อความยาวจนต้องขึ้นบรรทัดใหม่ แต่ต้องการให้แสดงผลอยู่ในบรรทัดเดียวกัน ให้ใส่เครื่องหมาย Backslash (\) ปิดท้ายก่อนขึ้นบรรทัดใหม่ เมื่อแสดงผลจะทำให้ข้อความอยู่ในบรรทัดเดียวกัน แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.12
การเขียนคำสั่งโปรแกรมที่มีข้อความหลายบรรทัดแต่กำหนดให้แสดงผลในบรรทัดเดียวกัน

```
1 str1 = 'Hello world, \
2 Python is simple and easy to learn. \
3 Everyone loves me.'
4 print (str1)
```

```
Hello world, Python is simple and easy to learn. Everyone loves
↪  me.
```

ในภาษาไพธอนสามารถควบคุมการแสดงผลของชนิดข้อมูลสตริงด้วยEscape Sequences ซึ่งเป็นรหัสพิเศษที่ใช้เครื่องหมาย Backslash (\) แล้วตามด้วยตัวอักษร แสดงในตารางที่ 4.1

รหัสพิเศษ	ความหมาย
\a	ให้ส่งเสียงเตือน
\b	ให้เลื่อนเคอร์เซอร์ถอยหลังไป 1 ตัวอักขระ
\e	ให้ยกเลิกคำสั่งสุดท้าย
\f	ให้ขึ้นบรรทัดใหม่
\n	ให้ขึ้นบรรทัดใหม่หลังจากแสดงผล
\r	ให้เคอร์เซอร์อยู่ทางซ้ายมือ
\t	ให้แสดงแท็บตามแนวนอน
\v	ให้แสดงแท็บตามแนวตั้ง
\	ให้แสดงผลเป็นบรรทัดเดียว
\\	ให้แสดงเครื่องหมาย \
\'	ให้แสดงเครื่องหมาย '
\"	ให้แสดงเครื่องหมาย "

ตาราง 4.1: รหัสพิเศษควบคุมการแสดงผลชนิดข้อมูลสตริง (Escape Sequences)

ตัวอย่าง 4.13
การเขียนคำสั่งโปรแกรมแสดงผล

```
str1 = 'Python was conceived in 1980s by Guido van Rossum \
at Centrum Wiskunde & Informatica (CWI) in the Netherlands. \
Python has feature highlights such as: \n\
\t -> Easy-to-learn\n\
\t -> Easy-to-read\n\
\t -> Easy-to-maintain\n\
and \'Everyone can learn and practice in a short time\'.'
print(str1)
```

```
Python was conceived in 1980s by Guido van Rossum at Centrum
↪  Wiskunde & Informatica (CWI) in the Netherlands. Python has
↪  feature highlights such as:
     -> Easy-to-learn
     -> Easy-to-read
     -> Easy-to-maintain
and 'Everyone can learn and practice in a short time'.
```

4.4.1 การจัดรูปแบบแสดงผล

การเปลี่ยนรูปแบบแสดงผล (Formatting) เป็นการจัดการข้อความ ตัวเลขจำนวนเต็ม ตัวเลขทศนิยม ให้อยู่ในลักษณะรูปแบบต่าง ๆ เช่น กำหนดให้มีทศนิยมกี่ตำแหน่ง กำหนดเลขยกกำลัง เป็นต้น โดยการใช้เครื่องหมายเปอร์เซ็นต์ (`%`) แล้วตามด้วยค่าตัวแปร

โครงสร้างการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย %

```
print('Str1: %s, Str2: %f, Str3: %d ' %(Arg1,
Arg2, Arg3))
```

สัญลักษณ์การจัดรูปแบบชนิดข้อมูลสตริง (String Formatting Symbol)

%	ใช้จัดรูปแบบแสดงผลและเพิ่ม % ด้วยคำสั่ง '%%'
%c	ใช้จัดรูปแบบแสดงผลตัวอักขระ
%d, %i	ใช้จัดรูปแบบแสดงผลเลขฐานสิบแบบมีเครื่องหมาย (8, -2, 0, 4)
%e	ใช้จัดรูปแบบแสดงผลเลขยกกำลังตัวอักษรเล็ก (110.15e+03)
%E	ใช้จัดรูปแบบแสดงผลเลขยกกำลังตัวอักษรใหญ่ (110.15E+03)
%f, %F	ใช้จัดรูปแบบแสดงผลตัวเลขทศนิยม
%g	ใช้จัดรูปแบบแสดงผลแบบสั้นของ %f และ %e
%G	ใช้จัดรูปแบบแสดงผลแบบสั้นของ %f และ %E
%o	ใช้จัดรูปแบบแสดงผลเลขฐานแปด
%s	ใช้จัดรูปแบบแสดงผลตัวอักขระหรือสตริงผ่านฟังก์ชัน str()
%u	ใช้จัดรูปแบบแสดงผลเลขฐานสิบแบบไม่มีเครื่องหมาย (8, 2, 0, 4)
%x	ใช้จัดรูปแบบแสดงผลเลขฐานสิบหกแสดงตัวอักษรเล็ก (43acd)
%X	ใช้จัดรูปแบบแสดงผลเลขฐานสิบหกแสดงตัวอักษรใหญ่ (43ACD)

โครงสร้างการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย {} และ (:) โดยเรียกใช้ฟังก์ชัน format()

```
print('Str1: {p0},, Str2: (p1), Str3: (p2)'
.format(p0, p1, p2))
```

โครงสร้างการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย **{}** และ **(:)** โดยเรียกใช้ฟังก์ชัน `format()` และการใช้ keyword อ้างอิงตำแหน่งอาร์กิวเมนต์

```
print('Str1: {k0},, Str2: (k1), Str3: (k2)'
.format(k0=v1, k1=v1, k2=v2))
```

ตัวอย่าง 4.14
การเปรียบเทียบการเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % กับเครื่องหมาย {}

```
num1 = 4945; num2 = 45 # สร้างตัวแปรเป็นชนิดข้อมูลจำนวนเต็ม
result = num1/num2 # หารค่าตัวแปร num1 กับ num2 เก็บผลลัพธ์ไว้ใน result
print('จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % =', '%f' %(result))
      # ใช้เครื่องหมาย % จัดรูปแบบแสดงผล
print('จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย {} =', '{:f}'.format(result))
      # ใช้เครื่องหมาย {} จัดรูปแบบแสดงผลร่วมกับฟังก์ชัน .format()
```

```
จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย % = 109.888889
จัดรูปแบบแสดงผลโดยใช้เครื่องหมาย{} = 109.888889
```

จากตัวอย่างที่ 1.13 เป็นการเปรียบเทียบการเขียนคำสั่งโปรแกรมการจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย **%** และ **{}** ผลลัพธ์ที่ได้ออกมาแสดงผลได้เหมือนกันแต่จะมีวิธีการเรียกใช้งานที่แตกต่างกัน

ตัวอย่าง 4.15
การเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลอ้างอิงตามตำแหน่งค่าอาร์กิวเมนต์

```
n = 4945; o = 45; x = 1453; s = 'Python'
# ประกาศตัวแปร n, o, x เป็นชนิดข้อมูลจำนวนเต็ม และ s เป็นชนิดข้อมูลสตริง
print('จัดรูปแบบแสดงผลข้อความ =','%10s' % s)
# ให้เยื้องค่าตัวแปร s ไปทางขวารวม 10 ตัวอักขระ
print('จัดรูปแบบแสดงผลเป็นเลขทศนิยม =','%.3f' % n)
# กำหนดให้แสดงผลทศนิยม 3 ตำแหน่ง
print('จัดรูปแบบแสดงผลเป็นเลขฐานแปด =',' %o'  %  o)
# กำหนดให้แสดงผลเป็นเลขฐาน 8
print('จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก=','%X' % x)
# กำหนดให้แสดงผลเป็นเลขฐาน 16
```

```
จัดรูปแบบแสดงผลข้อความ =     Python
จัดรูปแบบแสดงผลเป็นเลขทศนิยม = 4945.000
จัดรูปแบบแสดงผลเป็นเลขฐานแปด = 55
จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก = 5AD
```

หากต้องการเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลโดยใช้เครื่องหมาย () สามารถทำได้ดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.16
การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลโดยใช้เครื่องหมาย

```
1 n = 4945; o = 45; x = 1453; s = 'Python'
2 print('จัดรูปแบบแสดงผลข้อความ =','{:20s}'.format(s))
3 print('จัดรูปแบบแสดงผลเป็นเลขทศนิยม =','{:.3f}'.format(n))
4 print('จัดรูปแบบแสดงผลเป็นเลขฐานแปด =','{:o}'.format(o))
5 print('จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก=','{:X}'.format(x))
6 print('แยกจำนวนตัวเลขด้วยเครื่องหมาย , = {:,}'.format(45609823465))'
```

```
จัดรูปแบบแสดงผลข้อความ = Python
จัดรูปแบบแสดงผลเป็นเลขทศนิยม = 4945.000
จัดรูปแบบแสดงผลเป็นเลขฐานแปด = 55
จัดรูปแบบแสดงผลเป็นเลขฐานสิบหก = 5AD
แยกจำนวนตัวเลขด้วยเครื่องหมาย , = 45, 609, 823 , 465
```

จากตัวอย่างการเขียนคำสั่งโปรแกรมที่ 4.15 ในบรรทัดที่ 6 มีการใช้เครื่องหมาย **(,)** เข้ามาช่วยจัดแสดงผล โดยทำหน้าที่แบ่งตัวเลขออกเป็นกลุ่มละ 3 ตัว โดยเริ่มจัดกลุ่มจากข้างหลังมายังด้านหน้า

ตัวอย่าง 4.17
การเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผลอ้างอิงตามตำแหน่งค่าอาร์กิวเมนต์ ด้วยเครื่องหมาย %

```
test1 = 23; test2 = 24.5; test3 = 30; subject = 'Programming';
↪  grade = 'B+'
print('สมชาย เรียนวิชา %s ได้คะแนน\n\
ทดสอบครั้งที่ 1 = %d\n\
ทดสอบครั้งที่ 2 = %.2f\n\
ทดสอบครั้งที่ 3 = %d\n\
ได้เกรด = %s' %(subject, test1, test2, test3, grade))
# แสดงผลค่าอาร์กิวเมนต์ทั้งหมด (subject, test1, test2, test3, test4)
```

```
สมชาย เรียนวิชา Programming ได้คะแนน
ทดสอบครั้งที่ 1 = 23
ทดสอบครั้งที่ 2 = 24.50
ทดสอบครั้งที่ 3 = 30
ได้เกรด = B+
```

ตัวอย่าง 4.18
การเขียนคำสั่งโปรแกรมจัดรูปแบบแสดงผล ด้วยฟังก์ชัน format() โดยอ้างอิงตำแหน่งอาร์กิวเมนต์

```
1 print('สมใจ เรียนวิชา {} ได้คะแนน\n\
2 ทดสอบครั้งที่ 1 = {}\n\
3 ทดสอบครั้งที่ 2 = {:.2f}\n\
4 ทดสอบครั้งที่ 3 = {:.2f}\n\
5 ได้เกรด = {}' .format('Database System', 30, 25.50, 25.50, 'A'))
```

```
สมใจ เรียนวิชา Database System ได้คะแนน
ทดสอบครั้งที่ 1 = 30
ทดสอบครั้งที่ 2 = 25.50
ทดสอบครั้งที่ 3 = 25.50
ได้เกรด = A
```

จากตัวอย่างโปรแกรมใช้ฟังก์ชัน format() จัดรูปแบบแสดงผลโดยใช้เครื่องหมายปีกกา (...) แทรกเข้าไปในส่วนของข้อความ เพื่ออ้างอิงตำแหน่งของอาร์กิวเมนต์ภายในเครื่องหมายวงเล็บหลังฟังก์ชัน format() จากตัวอย่างโปรแกรมมีอาร์กิวเมนต์ทั้งหมด 5 ตัว และเมื่อเราอ้างอิงถึงอาร์กิวเมนต์ต้องมีเครื่องหมายปีกกา 5 ตัว เช่นกัน ถ้ามีจำนวนไม่เท่ากันจะทำให้เกิดแจ้งเตือนข้อผิดพลาดขึ้น นอกจากนี้เรายังสามารถจัดแสดงผลตัวเลขได้เหมือนกับการใช้เครื่องหมายเปอร์เซ็นต์ % แสดงดังตัวอย่างในบรรทัดที่ 3 และบรรทัดที่ 4 มีการกำหนดให้แสดงผลทศนิยมสองตำแหน่ง

ตัวอย่าง 4.19
การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผล ด้วยฟังก์ชัน `format()` โดยอ้างอิงตำแหน่งคีย์เวิร์ด

```
print('สมใจ เรียนวิชา {s} ได้คะแนน\n\
ทดสอบครั้งที่ 1 = {T1}\n\
ทดสอบครั้งที่ 2 = {T2:.2f}\n\
ทดสอบครั้งที่ 3 = {T3:.2f}\n\
ได้เกรด = {G}' \
.format(s='Database System', T1 = 30, T2 = 25.50, T3 = 25.50, G
    ↪  ='A'))
```

```
สมใจ เรียนวิชา Database System ได้คะแนน
ทดสอบครั้งที่ 1 = 30
ทดสอบครั้งที่ 2 = 25.50
ทดสอบครั้งที่ 3 = 25.50
ได้เกรด = A
```

จากตัวอย่างโปรแกรมใช้วิธีการจัดรูปแบบแสดงผลโดยการอ้างอิงตำแหน่งคีย์เวิร์ด ลักษณะการอ้างอิงแบบนี้ภายในเครื่องหมายวงเล็บปีกกาจะต้องระบุชื่อคีย์เวิร์ดที่ต้องการแสดงผลข้อมูลที่เก็บค่าไว้ และภายในเครื่องหมายวงเล็บปีกกาสามารถจัดรูปแบบการแสดงผลตัวเลขได้เหมือนกับวิธีการอ้างอิงอาร์กิวเมนต์

4.4.2 การจัดรูปแบบแสดงผลด้วย f-string

f-string เป็นวิธีการจัดรูปแบบการแสดงผลในภาษาไพธอนอีกรูปแบบหนึ่งที่ใช้งานง่ายและสะดวก โดยถูกเพิ่มเข้ามาให้ใช้งานได้ตั้งแต่ไพธอนเวอร์ชัน 3.6 การแสดงผลจะมี `f` ข้างหน้าเครื่องหมาย `'...'` ส่วนที่นำมาแสดงผลจะอยู่ภายในเครื่องหมายปีกกา **`{...}`** ซึ่งภายในจะเป็นตัวแปร หรือการดำเนินการต่าง ๆ ก็ได้ มีรูปแบบวิธีการใช้งานดังนี้

โครงสร้างคำสั่งการจัดรูปแบบแสดงผลด้วย f-string

```
print(f'... {var1[:format]} ... {varn[:format]}')
```

ตัวอย่าง 4.20
การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
1 r = float(input('กรอกรัศมีของวงกลม: '))
2 pi = 3.14
3 area = pi * r * r
4 perimeter = 2 * pi * r
5 print(f'พื้นที่ของวงกลมนี้ คือ {area} ตารางหน่วย')
6 print(f'ความยาวเส้นรอบวงของวงกลมนี้ คือ {perimeter} หน่วย')
```

```
กรอกรัศมีของวงกลม: 3.5
พื้นที่ของวงกลมนี้ คือ 38.465 ตารางหน่วย
ความยาวเส้นรอบวงของวงกลมนี้ คือ 21.98 หน่วย
```

ตัวอย่าง 4.21
การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
1 text = 'Hello World'
2 print(f'Length of \'{text}\' is {len(text)}')
```

```
Length of 'Hello World' is 11
```

ตัวอย่าง 4.22
การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
f = int(input('Enter first score: '))
s = int(input('Enter second score: '))
t = int(input('Enter third score: '))
total = f + s + t
print(f'Total Score = {f} + {s} + {t} = {total}')
```

```
Enter first score: 25
Enter second score: 15
Enter third score: 12
Total Score = 25 + 15 + 12 = 52
```

ตัวอย่าง 4.23
การเขียนคำสั่งโปรแกรมจัดรูปแบบการแสดงผลด้วย f-string

```
s = 'Database System'
T1 = 30; T2 = 25.50; T3 = 25.50
G = 'A'
print(f'สมใจเรียนวิชา {s} ได้คะแนน\n \
ทดสอบครั้งที่ 1 = {T1}\n \
ทดสอบครั้งที่ 2 = {T2:.2f}\n \
ทดสอบครั้งที่ 3 = {T3:.4f}\n \
ได้เกรด = {G}')
```

```
สมใจเรียนวิชา Database System ได้คะแนน
 ทดสอบครั้งที่ 1 = 30
 ทดสอบครั้งที่ 2 = 25.50
 ทดสอบครั้งที่ 3 = 25.5000
 ได้เกรด = A
```

สัญลักษณ์	การแสดงผล
<	ใช้จัดรูปแบบแสดงผลให้อยู่ทางด้านซ้ายมือ
>	ใช้จัดรูปแบบแสดงผลให้อยู่ทางด้านขวามือ
=	ใช้กำหนดช่องว่างหน้าข้อความ และใช้ได้กับเฉพาะตัวเลขเท่านั้น
^	ใช้จัดรูปแบบให้อยู่กึ่งกลาง

ตาราง 4.2: สัญลักษณ์กำหนดตำแหน่งรูปแบบแสดงผลข้อความ

4.4.3 การจัดตำแหน่งแสดงผลข้อความ

โดยปกติข้อความที่แสดงผลออกมาจะถูกกำหนดให้อยู่ทางซ้ายมือ ถ้ามีหลายบรรทัดและต้องการให้แสดงแต่ละคอลัมน์ให้ตรงกันจะเกิดปัญหาขึ้น เนื่องจากขนาดข้อความในแต่ละคอลัมน์ไม่เท่ากัน ดังนั้นจึงควรใช้สัญลักษณ์เข้ามากำหนดค่าในการแสดงผลตำแหน่ง ตาราง 4.2 แสดงสัญลักษณ์กำหนดตำแหน่งระยะห่างข้อความที่ต้องการแสดงผล หากต้องการระยะห่างแต่ละคอลัมน์ให้ใส่ตัวเลขต่อท้ายหลังสัญลักษณ์

ตัวอย่าง 4.24
การจัดตำแหน่งแสดงผลข้อความ

```
1 f_name = 'Joe'
2 l_name = 'Biden'
3 code ='1234'
4 print('{:<10s} {:^15s} {:>s}'.format('First Name','Last
  ↪  name','Code'))
5 print('{:<10s} {:^15s} {:>s}'.format(f_name, l_name, code))
```

```
First Name    Last name    Code
Joe             Biden      1234
```

จากตัวอย่าง 4.24 เราอธิบายการทำงานโปรแกรมได้ดังนี้

- บรรทัดที่ 1 สร้างตัวแปร `f_name` สำหรับเก็บชื่อ

สัญลักษณ์	การจัดรูปแบบชนิดข้อมูลจำนวนเต็ม
+	ใช้จัดรูปแบบแสดงผลให้มีเครื่องหมาย + อยู่ด้านหน้า
-	ใช้จัดรูปแบบแสดงผลให้มีเครื่องหมาย - อยู่ด้านหน้า
space	ใช้กำหนดช่องว่างหน้าข้อความ

ตาราง 4.3: สัญลักษณ์กำหนดเครื่องหมายชนิดข้อมูลจำนวนเต็ม

- บรรทัดที่ 2 สร้างตัวแปร `l_name` สำหรับเก็บนามสกุล

- บรรทัดที่ 3 สร้างตัวแปร `code` เก็บรหัส

- บรรทัดที่ 4 จัดรูปแบบแสดงผลด้วยฟังก์ชัน `.format()` ซึ่งเป็นการกำหนดหัวคอลัมน์ได้แก่ อาร์กิวเมนต์ `f_name` กำหนดให้ชิดซ้ายและให้จัดรูปแบบแสดงผลแบบสตริง อาร์กิวเมนต์ `l_name` กำหนดให้อยู่กึ่งกลางและให้มีระยะห่าง 15 ตัวอักขระ พร้อมจัดรูปแบบแสดงผลแบบสตริง ส่วนอาร์กิวเมนต์ code จัดตำแหน่งให้ชิดขวาและจัดรูปแบบแสดงผลแบบสตริง

- บรรทัดที่ 5 จัดรูปแบบแสดงผลด้วยฟังก์ชัน `.format()` โดยการนำเอาตัวแปร `f_name, l_name` และ code มาแสดงผลในรูปแบบสตริงและกำหนดระยะห่าง

เมื่อต้องการจัดรูปแบบแสดงผลให้มีเครื่องหมาย `+` และ `-` เครื่องหมายนำหน้าจำนวนเต็มบวกหรือลบ หรือต้องการให้มีช่องว่างนำหน้าจำนวนเต็มทั้งสอง สามารถนำเครื่องหมายในตาราง 4.3 เข้ามาช่วยจัดรูปแบบแสดงผลได้

ตัวอย่าง 4.25
การจัดรูปแบบแสดงผลเครื่องหมายหน้าชนิดข้อมูลจำนวนเต็ม

```
x = 987.485; y = -225.154
print('{:+f}; {:+f}'.format(x, y))
# กำหนดให้มีเครื่องหมาย + นำอยู่ด้านหน้า
print('{:-.3f}; {:-.3f}'.format(x, y))
# กำหนดให้มีจำนวนทศนิยม 3 ตำแหน่ง และเครื่องหมาย - อยู่ด้านหน้า
print('{: .3f}; {: .3f}'.format(x, y))
# กำหนดให้มีจำนวนทศนิยม 3 ตำแหน่ง และให้มีช่องว่างด้านหน้า
```

```
+987.485000; -225.154000
987.485; -225.154
 987.485; -225.154
```

4.5 เมธอดที่ใช้กับชนิดข้อมูลกลุ่มอักขระหรือสตริง

ในส่วนนี้จะขอยกตัวอย่างบางเมธอดที่นำมาใช้งานสำหรับจัดการกับชนิดข้อมูลสตริง หากต้องการดูข้อมูลรายละเอียดเพิ่มเติมของเมธอดอื่น ๆ ที่นำมาใช้งานร่วมกับชนิดข้อมูลสตริง ให้ใช้คำสั่ง `help(str)` เพื่อดูวิธีการเรียกใช้งาน

4.5.1 เมธอดการเปลี่ยนลักษณะสตริง

เมธอดในกลุ่มนี้จะทำหน้าที่เปลี่ยนลักษณะสตริง เช่น จากตัวอักษรพิมพ์เล็กให้เป็นพิมพ์ใหญ่ หรือจากตัวอักษรพิมพ์ใหญ่ให้เป็นพิมพ์เล็ก หรือจัดการแสดงผลสลับตัวอักษรพิมพ์ใหญ่และพิมพ์เล็ก

เมธอด	ความหมาย	รูปแบบการใช้งาน
`capitalize()`	ใช้สำหรับแปลงตัวอักษรแรกของประโยคให้เป็นตัวอักษรพิมพ์ใหญ่	`str.capitalize()`
`lower()`	ใช้เปลี่ยนสตริงให้เป็นตัวอักษรพิมพ์เล็กทั้งหมด	`str.lower()`
`upper()`	ใช้สำหรับเปลี่ยนสตริงเป็นตัวอักษรพิมพ์ใหญ่ทั้งหมด	`str.upper()`
`title()`	ใช้สำหรับเปลี่ยนตัวอักษรแรกของแต่ละคำให้เป็นตัวอักษรพิมพ์ใหญ่ (เช่น The Best)	`str.tile()`

ตาราง 4.4: เมธอดสำหรับเปลี่ยนลักษณะสตริง เมื่อ `str` คือ ตัวแปรชนิดข้อมูลสตริง

ตัวอย่าง 4.26
การเขียนคำสั่งโปรแกรมเปลี่ยนลักษณะสตริง

```
1 str = 'python is the best Programming Language.'
2 print('เปลี่ยนตัวอักษรแรกให้เป็นตัวพิมพ์ใหญ่ =', str.capitalize())
3 print('เปลี่ยนตัวอักษรให้เป็นพิมพ์เล็กทั้งหมด =', str.lower())
4 print('เปลี่ยนตัวอักษรแรกของคำให้เป็นตัวพิมพ์ใหญ่ =', str.title())
5 print('เปลี่ยนตัวอักษรให้เป็นตัวพิมพ์ใหญ่ทั้งหมด =', str.upper())
6 print('สลับตัวอักษรตัวพิมพ์เล็กและใหญ่ในประโยค =', str.swapcase())
```

```
เปลี่ยนตัวอักษรแรกให้เป็นตัวพิมพ์ใหญ่ = Python is the best programming
↪ language.
เปลี่ยนตัวอักษรให้เป็นพิมพ์เล็กทั้งหมด = python is the best programming
↪ language.
เปลี่ยนตัวอักษรแรกของคำให้เป็นตัวพิมพ์ใหญ่ = Python Is The Best Programming
↪ Language.
เปลี่ยนตัวอักษรให้เป็นตัวพิมพ์ใหญ่ทั้งหมด = PYTHON IS THE BEST PROGRAMMING
↪ LANGUAGE.
สลับตัวอักษรตัวพิมพ์เล็กและใหญ่ในประโยค = PYTHON IS THE BEST pROGRAMMING
↪ lANGUAGE.
```

เมธอด	ความหมาย	รูปแบบการใช้งาน
`center()`	ใช้สำหรับจัดข้อความให้อยู่ตรงกลาง	`str.center(width,」[fillchar])`
`ljust()`	ใช้สำหรับจัดสตริงให้ชิดซ้าย	`str.ljust(width,」[fillchars])`
`rjust()`	ใช้สำหรับจัดสตริงให้ชิดขวา	`str.rjust(length,」[fillchars])`

ตาราง 4.5: เมธอดจัดตำแหน่งสตริง เมื่อ `str` คือ ตัวแปรเก็บชนิดข้อมูลสตริง `width` คือ กำหนดขนาดความกว้างที่รวมข้อความ และ `fillchar` คือ ตัวอักขระหรือสัญลักษณ์ที่ไว้เติมให้เต็มความยาว ปกติจะเป็นช่องว่าง จะระบุหรือไม่ก็ได้

4.5.2 เมธอดการจัดตำแหน่งสตริง

เมธอดในกลุ่มนี้ทำหน้าที่จัดวางตำแหน่งของสตริงให้อยู่ด้านซ้าย ขวา หรือตรงกลางจากค่าที่กำหนดตำแหน่งแสดงผล

ตัวอย่าง 4.27
การเขียนคำสั่งโปรแกรมตรวจสอบตัวเลขภายในชนิดข้อมูลสตริงด้วยเมธอด `isalnum()`

```
1 str = 'Programming'
2 print('จัดตำแหน่งให้อยู่ด้านซ้าย =', str.ljust(20))
3 print('จัดตำแหน่งให้อยู่ตรงกลาง =', str.center(20))
4 print('จัดตำแหน่งให้อยู่ด้านขวา=', str.rjust(20, '-'))
```

```
จัดตำแหน่งให้อยู่ด้านซ้าย = Programming
จัดตำแหน่งให้อยู่ตรงกลาง =      Programming
จัดตำแหน่งให้อยู่ด้านขวา = ----------Programming
```

4.5.3 เมธอดสำหรับตรวจสอบสตริง

เมธอดในกลุ่มนี้ทำหน้าที่ตรวจสอบข้อความหรือสตริงว่าประกอบด้วยส่วนประกอบอะไรบ้าง

ตัวอย่าง 4.28
การเขียนคำสั่งโปรแกรมตรวจสอบสตริง

```
str1 ='Programming'; str2 = 'Python2019'
str3 = 'Python programming Language'
print('ประกอบด้วยตัวอักษรทั้งหมดหรือไม่ =', str1.isalpha())
print('ประกอบด้วยตัวอักษรหรือตัวเลขทั้งหมดหรือไม่ =', str2.isalnum())
print('แต่ละคำขึ้นต้นด้วยตัวอักษรพิมพ์ใหญ่ทั้งหมดหรือไม่=', str3.istitle())
```

```
ประกอบด้วยตัวอักษรทั้งหมดหรือไม่ = True
ประกอบด้วยตัวอักษรหรือตัวเลขทั้งหมดหรือไม่ = True
แต่ละคำขึ้นต้นด้วยตัวอักษรพิมพ์ใหญ่ทั้งหมดหรือไม่ = False
```

4.5.4 เมธอดสำหรับการนับค่า ค้นหา และแก้ไขสตริง

เมธอดในกลุ่มนี้ทำหน้าที่นับตัวอักขระหรือคำที่มีอยู่ในสตริง หรือค้นหาคำที่ต้องการได้ระบุ รวมไปถึงเมธอดที่นำมาใช้สำหรับแก้ไขค่าข้อมูลในสตริง

เมธอด	ความหมาย	รูปแบบการใช้งาน
`isalnum()`	คืนค่า True ถ้าสตริงประกอบด้วยตัวอักษรปนกับตัวเลข หรือแค่ตัวอักษร ตัวเลขอย่างใดอย่างหนึ่งเพียงอย่างเดียว ถ้ามีสัญลักษณ์อื่น หรือช่องว่าง จะคืนค่า False	`str.isalnum()`
`isalpha()`	คืนค่า True ถ้าสตริงประกอบด้วยตัวอักษรทั้งหมด ถ้ามีสัญลักษณ์ ตัวเลข หรือช่องว่างจะคืนค่า False	`str.isalpha()`
`isdigit()`	คืนค่า True ถ้าสตริงประกอบด้วยตัวเลขทั้งหมด ถ้ามีตัวอักษร สัญลักษณ์ หรือช่องว่าง จะคืนค่า False	`str.isdigit()`
`isdecimal()`	ใช้สำหรับตรวจสอบตัวเลข แสดงผลลัพธ์เหมือนเมธอด isdigit()	`str.isdecimal()`
`isnumber()`	ใช้สำหรับตรวจสอบตัวเลข แสดงผลลัพธ์เหมือนเมธอด isdigit()	`str.isnumber()`
`islower()`	คืนค่า True ถ้าสตริงประกอบด้วยตัวพิมพ์เล็กทั้งหมด ถ้ามีตัวอักษรพิมพ์ใหญ่ประกอบด้วย จะคืนค่า False	`str.islower()`
`isupper()`	คืนค่า True ถ้าสตริงประกอบด้วยตัวพิมพ์ใหญ่ทั้งหมด ถ้ามีตัวอักษรพิมพ์เล็กประกอบด้วย จะคืนค่า False	`str.isupper()`
`istitle()`	คืนค่า True ถ้าตัวอักษรแรกแต่ละคำในสตริงเป็นตัวอักษรพิมพ์ใหญ่ ถ้าขึ้นต้นด้วยตัวอักษรพิมพ์เล็ก จะคืนค่า False	`str.istitle()`
`isspace()`	คืนค่า True ถ้าสตริงเก็บค่าว่าง ถ้ามีข้อมูลอยู่ จะคืนค่า False	`str.space()`

ตาราง 4.6: เมธอดใช้สำหรับตรวจสอบสตริง

เมธอด	ความหมาย	รูปแบบการใช้งาน
`count()`	ใช้สำหรับนับจำนวนตัวอักขระหรือคำในสตริง	`str.count(x, strart, stop)`
`endswith()`	คืนค่า True ถ้าลงท้ายด้วยตัวอักษรที่ระบุ ถ้าไม่ตรงกับที่ระบุจะคืนค่า False	`str.endswith(x, strart, stop)`
`find()`	ใช้สำหรับค้นหาตำแหน่งอักขระที่ระบุในสตริง ถ้าไม่พบจะคืนค่าเป็น -1	`str.find(x, strart, stop)`
`index()`	ใช้สำหรับตำแหน่งข้อความในสตริง	`str.index(x, strart, stop)`
`startswith()`	คืนค่า True ถ้าขึ้นต้นด้วยตัวอักษรที่ระบุ ถ้าไม่ตรงกับที่ระบุ จะคืนค่า False	`str.startswith(x, strart, stop)`
`rfind()`	ใช้สำหรับค้นหาตำแหน่งอักขระที่ระบุในสตริง จะแสดงตำแหน่งสุดท้ายที่ค้นพบ ถ้าไม่พบจะคืนค่าเป็น -1	`str.rfind(x, strart, stop)`
`replace()`	ใช้สำหรับแทนที่ข้อความที่ต้องการตามที่ระบุ	`str.replace(old, new [, count])`

ตาราง 4.7: เมธอดที่ใช้สำหรับการนับ ค้นหา และแก้ไขค่าสตริง เมื่อ `str` คือ ตัวแปรชนิดข้อมูลสตริงที่ต้องการนับจำนวนตัวอักขระ `x` คือ ตัวอักขระหรือคำที่ระบุ `start` คือ ตำแหน่งเริ่มต้น `stop` คือ ตำแหน่งสุดท้าย `old` คือ ข้อความเดิม `new` คือ ข้อความใหม่ และ `count` คือ จำนวนครั้งที่ต้องการแทนที่จะระบุหรือไม่ก็ได้

ตัวอย่าง 4.29
การเขียนคำสั่งโปรแกรมการตรวจสอบ การแสดง และการนับตำแหน่ง

```
str1 = 'Python is the best programming language.'
print('คำสุดท้ายในประโยคคือ language. =', str1.endswith('language.'))
print('language. เริ่มจากตำแหน่งที่ =', str1.index('language.'))
print('มีตัวอักษร e ทั้งหมด =', str1.count('e'))
```

```
คำสุดท้ายในประโยคคือ language. = True
language. เริ่มจากตำแหน่งที่ = 31
มีตัวอักษร e ทั้งหมด = 3
```

4.5.5 เมธอดสำหรับรวม แยก และตัดสตริง

เมธอดในกลุ่มนี้ทำหน้าทรารวมสตริงเข้าด้วยกันหรือนำสัญลักษณ์เข้ามาใช้งานร่วมกับสตริง หรือใช้เมธอดทำการตัดตัวอักษร ช่องว่าง ออกจากสตริง

ตัวอย่าง 4.30
การเขียนคำสั่งโปรแกรม การดำเนินการเปรียบเทียบกับชนิดข้อมูลสตริง

```
str_1 = '___Python Programming___'
str_2 = 'Python is the best Programming.'
str_rst = str_1.rstrip('_')
str_rsp = str_1.rsplit('m')
str_sp = str_2.split(' ')
print(str_rst)
print(str_rsp)
print(str_sp)
```

```
___Python Programming
['___Python Progra', '', 'ing___']
['Python', 'is', 'the', 'best', 'Programming.']
```

เมธอด	ความหมาย	รูปแบบการใช้งาน
`join()`	ใช้สำหรับเชื่อมสตริงที่เป็นชนิดข้อมูลแบบเรียงลำดับให้ต่อกัน	`str.join(seq)`
`lstrip()`	ใช้สำหรับลบช่องว่างหรือตัวอักขระทางด้านซ้ายของสตริง	`str.lstrip([chars])`
`partition()`	ใช้สำหรับแยกสตริง ผลลัพธ์ที่ได้เก็บอยู่ในรูปแบบชนิดข้อมูลทูเพิล	`str.partition(x)`
`rstrip()`	ใช้สำหรับตัดช่องว่างหรือตัวอักขระทางด้านหลัง (ด้านขวา) ของสตริง	`str.rstrip([chars])`
`split()`	ใช้สำหรับแยกข้อความ ผลลัพธ์ที่ได้อยู่ในรูปแบบชนิดข้อมูลลิสต์	`str.split(sep)`
`strip()`	ใช้สำหรับลบข้อความหรือตัวอักขระและช่องว่าง ทั้งด้านหน้าและด้านหลังสตริง	`str.strip([chars])`

ตาราง 4.8: เมธอดที่ใช้สำหรับรวม แยก และตัดสตริง เมื่อ `str` คือ ตัวแปรเก็บชนิดข้อมูลสตริงที่ต้องการนับจำนวนตัวอักขระ, `seq` คือ ชนิดข้อมูลแบบเรียงลำดับที่ต้อกงารเชื่อมต่อกัน, `chars` คือ ตัวอักขระที่ต้องการลบ ค่าปกติเป็นช่องว่าง, `x` คือ ข้อความหรือเครื่องหมายที่กำหนดใช้แยกสตริง และ `sep` คือ สัญลักษณ์ที่กำหนดใช้แยกข้อความ ค่าปกติเป็นช่องว่าง

4.6 การดำเนินการกับชนิดข้อมูลอักขระหรือสตริง

สามารถนำเอาตัวดำเนินการต่าง ๆ มาใช้กับชนิดข้อมูลอักขระหรือสตริงได้เหมือนกับชนิดข้อมูลอื่น ๆ เช่น การเปรียบเทียบ การเชื่อม การตรวจสอบ การมีอยู่หรือไม่มีอยู่ เป็นต้น แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.31
การเขียนคำสั่งโปรแกรม การดำเนินการเปรียบเทียบกับชนิดข้อมูลสตริง

```
1 str1 = 'Hello Python'; str2 = 'Programming'; str3 =
  ↪  'Programming'
2 print('str1 < str2 หรือไม่ =', str1 < str2)
3 print('str1 > str2 หรือไม่ =', str1 > str2)
4 print('str1 >= str2 หรือไม่ =', str1 >= str2)
5 print('str1 != str2 หรือไม่ =', str1 != str2)
6 print('str2 == str3 หรือไม่ =', str2 == str3)
```

```
str1 < str2 หรือไม่ = True
str1 > str2 หรือไม่ = False
str1 >= str2 หรือไม่ = False
str1 != str2 หรือไม่ = True
str2 == str3 หรือไม่ = True
```

นอกจากนี้ยังสามารถใช้ตัวดำเนินการ **`in`** และ **`not in`** ตรวจสอบว่ามีหรือไม่มีตัวอักขระที่ระบุในข้อความผลลัพธ์ที่ได้จะเป็น **`True`** หรือ **`False`** แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 4.32
การเขียนคำสั่งโปรแกรม การดำเนินการตรวจสอบชนิดข้อมูลสตริง

```
str1 = 'Python Programming'
str2 = 'อักขระหรือสตริง'

# ตรวจสอบตัวอักขระ i ในค่าตัวแปร str1
print('มี i อยู่ใน str1 หรือไม่ =', 'i' in str1)

# ตรวจสอบตัวอักขระ A ในค่าตัวแปร str1
print('มี A อยู่ใน str1 หรือไม่ =', 'A' in str1)

# ตรวจสอบตัวอักขระ พ ในค่าตัวแปร str2
print('มี พ อยู่ใน str2 หรือไม่ =','พ' in str2)

# ตรวจสอบ 'สตริง' ในค่าตัวแปร str2
print('มี ' + 'สตริง' + ' อยู่ใน str2 หรือไม่ =', 'สตริง' in str2)
```

```
มี i อยู่ใน str1 หรือไม่ = True
มี A อยู่ใน str1 หรือไม่ = False
มี พ อยู่ใน str2 หรือไม่ = False
มี 'สตริง' อยู่ใน str2 หรือไม่ = True
```

สรุปก่อนจบบท

ชนิดข้อมูลกลุ่มอักขระหรือสตริงคือ ตัวอักขระที่นำมาเรียงต่อกันเป็นข้อความหรือประโยค เมื่อประกาศตัวแปรเก็บชนิดข้อมูลนี้จะอยู่ในเครื่องหมาย Single Quote (`'...'`) หรือเครื่องหมาย Double Quote (`"..."`) ซึ่งในภาษาไพธอนใช้ได้ทั้งสองเครื่องหมาย ในบทนี้ผู้อ่านได้เรียนรู้วิธีการประกาศชนิดข้อมูลสตริง การเข้าถึงตำแหน่งสตริง การจัดรูปแบบข้อความ รวมไปถึงสัญลักษณ์และรหัสพิเศษที่นำมาใช้ควบคุมการแสดงผลลัพธ์และได้แนะนำเมธอดต่าง ๆ ที่นำมาใช้งานร่วมกับชนิดข้อมูลสตริง

แบบฝึกหัด

1. กำหนดให้

```
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

ให้เขียนคำสั่งโปรแกรมแสดงผลเฉพาะสระภาษาอังกฤษและแสดงตำแหน่งที่อยู่ของสระแต่ละตัว

2. จากคำสั่งโปรแกรมต่อไปนี้

```
alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
x1 = alphabet[?:?]
x2 = alphabet[?:?:?]
x3 = alphabet[?:?:?]
print(f'x1 = {x1}, x2 = {x2},  x3 = {x3}')
```

จงเติมตัวเลขที่ถูกต้องลงในตำแหน่ง ? เพื่อให้โปรแกรมแสดงผลลัพธ์เป็น

```
x1 = FGHIJ, x2 = LNPRTVX,  x3 = ZWTQ
```

3. จงเขียนโปรแกรมรับข้อมูลผ่านทางแป้นพิมพ์ และจัดรูปแบบแสดงผลตามที่กำหนดให้

```
Sent from
Name _____________________
Address __________________
Zip code _________________
                          To
                          Name _____________________
                          Address __________________
                          Zip code _________________
```

4. จงเขียนโปรแกรมเพื่อรับ ชื่อเต็ม (Full Name) ทางแป้นพิมพ์โดยให้รับค่าด้วยตัวแปรตัวเดียวเท่านั้น โดยใช้การเว้นวรรคเพื่อแยกชื่อแรกและนามสกุล จากนั้นให้ทำการแยก

ชื่อแรก และ นามสกุลและแสดงจำนวนอักขระของชื่อแรกและนามสกุลดังตัวอย่างที่กำหนดให้

```
Enter the full name: Tony Woodsome
----------------------------------------
Item        |    Value    |  String Length
----------------------------------------
First Name  |  Tony       |   4
Last Name   |  Woodsome   |   8
----------------------------------------
```

5. จากโปรแกรมในข้อ 4 จงแก้ไขให้โปรแกรมรองรับเมื่อชื่อเต็มอาจประกอบไปด้วย ชื่อแรก ชื่อกลาง และนามสกุล และในกรณีที่ไม่มีชื่อกลางให้โปรแกรมแสดงผลลัพธ์ดังตัวอย่างที่กำหนดให้

```
Enter the full name: Tony Sinatra Woodsome
----------------------------------------
Item        |    Value    |  String Length
----------------------------------------
First Name  |  Tony       |   4
Middle Name |  Sinatra    |   7
Last Name   |  Woodsome   |   8
----------------------------------------

Enter the full name: Tony Woodsome
----------------------------------------
Item        |    Value    |  String Length
----------------------------------------
First Name  |  Tony       |   4
Middle Name |  -          |   0
Last Name   |  Woodsome   |   8
----------------------------------------
```