



เขียนโปรแกรมภาษา Python
สำหรับผู้เริ่มต้น (อัปเดตล่าสุด)

แนะนำเนื้อหา

Python คือ ภาษาคอมพิวเตอร์ที่ใช้สำหรับการพัฒนาโปรแกรม โดยผู้พัฒนาต้องเรียนรู้โครงสร้างภาษา Python ว่ามีโครงสร้างการเขียนอย่างไร เพื่อสั่งการให้คอมพิวเตอร์นั้นสามารถทำงานตามวัตถุประสงค์ที่ต้องการได้

โดยการสั่งการให้คอมพิวเตอร์ทำงานได้นั้น ต้องอาศัยส่วนที่เรียกว่า **ตัวแปลภาษา**

ตัวแปลภาษา

ตัวแปลภาษาเปรียบเสมือนกับล่าม ทำหน้าที่แปลงโค้ดภาษาคอมพิวเตอร์ที่มนุษย์เขียนขึ้น (ภาษา Python) ไปเป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจว่าต้องการให้ทำงานอะไร



ประเภทของตัวแปลภาษา

ในปัจจุบันตัวแปลภาษาแบ่งออกเป็น 2 ประเภท

- คอมไพเลอร์ (Compiler)
- อินเตอร์พรีเตอร์ (Interpreter)

	ข้อดี	ข้อเสีย
Compiler	<ul style="list-style-type: none"> ทำงานได้เร็ว เนื่องจากจะทำการแปลคำสั่งทั้งหมดในครั้งเดียว แล้วจึงทำงานตามคำสั่งของโปรแกรมในภายหลัง 	<ul style="list-style-type: none"> เมื่อเกิดข้อผิดพลาดขึ้นจะตรวจสอบหาข้อผิดพลาดได้ยาก เพราะทำการแปลคำสั่งทีเดียวทั้งโปรแกรม
Interpreter	<ul style="list-style-type: none"> แปลคำสั่งทีละบรรทัด ทำให้หาข้อผิดพลาดของโปรแกรมได้ง่าย เนื่องจากแปลคำสั่งทีละบรรทัด สามารถสั่งให้โปรแกรมทำงานเฉพาะจุดได้ ไม่เสียเวลารอการแปลคำสั่งเป็นเวลานาน 	<ul style="list-style-type: none"> ช้า เนื่องจากทำงานทีละบรรทัด

	ข้อดี	ข้อเสีย
Compiler	<ul style="list-style-type: none"> ทำงานได้เร็ว เนื่องจากจะทำการแปลคำสั่งทั้งหมดในครั้งเดียว แล้วจึงทำงานตามคำสั่งของโปรแกรมในภายหลัง 	<ul style="list-style-type: none"> เมื่อเกิดข้อผิดพลาดขึ้นจะตรวจสอบหาข้อผิดพลาดได้ยาก เพราะทำการแปลคำสั่งทีเดียวทั้งโปรแกรม
Interpreter	<ul style="list-style-type: none"> แปลคำสั่งทีละบรรทัด ทำให้หาข้อผิดพลาดของโปรแกรมได้ง่าย เนื่องจากแปลคำสั่งทีละบรรทัด สามารถสั่งให้โปรแกรมทำงานเฉพาะจุดได้ ไม่เสียเวลารอการแปลคำสั่งเป็นเวลานาน 	<ul style="list-style-type: none"> ช้า เนื่องจากทำงานทีละบรรทัด

ภาษา Python จัดเป็นภาษาคอมพิวเตอร์ที่ใช้ตัวแปลภาษาแบบอินเตอร์พรีเตอร์

ข้อดีของภาษา Python

- ทำงานได้หลายระบบปฏิบัติการ (Windows , Mac , Linux)
- เป็น Open Source (ใช้งานได้ฟรี ไม่มีค่าใช้จ่าย)
- โครงสร้างไวยากรณ์ภาษา เข้าใจง่าย
- รองรับการเขียนโปรแกรมเชิงวัตถุ (OOP)
- มีไลบรารีให้ใช้งานจำนวนมาก (เว็บ , เกม , กราฟิก , AI , ML)
- รองรับการทำงานกับฐานข้อมูล (Database)

ตัวอย่างการประยุกต์ใช้งาน

- สามารถพัฒนาแอปพลิเคชันในรูปแบบ GUI ได้ (Tkinter , PyQt)
- สามารถพัฒนาเว็บแอปพลิเคชันได้ (Django Framework , Flask)
- สามารถพัฒนาเกมได้ (Pygame)
- Data Science (Numpy , Pandas , Matplotlib)
- Machine Learning & AI

ตัวอย่างการประยุกต์ใช้งาน

- งานด้านการประมวลผลภาพ (Image Processing)
- การพัฒนาระบบแผนที่
- การสกัดข้อมูลจากเว็บไซต์ (Web Scrapping)
- การพัฒนา API (Fast API , Django REST Framework)
- ทำงานร่วมกับระบบจัดการฐานข้อมูล (SQL , NOSQL)

องค์ประกอบพื้นฐานของภาษา Python

- ไฟล์ที่เก็บโค้ดภาษา Python (Source Code) จะมีนามสกุลไฟล์ .py
- การย่อหน้าโค้ดโปรแกรม (Indentation) คือ การจัดรูปแบบคำสั่งในภาษา Python โดยแบ่งโค้ดออกเป็นกลุ่มย่อยๆ ซึ่งแต่ละกลุ่มจะต้องจัดย่อหน้าให้เท่ากัน ไม่เช่นนั้นจะเกิดข้อผิดพลาดและไม่สามารถรันโปรแกรมได้

ตัวอย่างการย่อหน้า

```
if username == "admin":
```

```
    print("สวัสดีผู้ดูแลระบบ")
```

```
else:
```

```
    print("ข้อมูลไม่ถูกต้อง")
```



ตัวอย่างการย่อหน้า

```
if username == "admin":
```

```
    print("สวัสดีผู้ดูแลระบบ")
```

```
else:
```

```
    print("ข้อมูลไม่ถูกต้อง")
```

```
if username == "admin":
```

```
    ➡ print("สวัสดีผู้ดูแลระบบ")
```

```
else:
```

```
    ➡ print("ข้อมูลไม่ถูกต้อง")
```

ตัวอย่างการย่อหน้า

```
if username == "admin":
```

```
    print("สวัสดีผู้ดูแลระบบ")
```

```
else:
```

```
    print("ข้อมูลไม่ถูกต้อง")
```

```
if username == "admin":
```

```
    print("สวัสดีผู้ดูแลระบบ")
```

```
else:
```

```
    print("ข้อมูลไม่ถูกต้อง")
```



องค์ประกอบพื้นฐานของภาษา Python

- **การเขียนคอมเมนต์ (Comment)** คือ ส่วนหมายเหตุของโปรแกรมเพื่ออธิบายหน้าที่หรือความหมายของโค้ดที่เขียนหรือยกเลิกโค้ดชั่วคราว ส่งผลให้ตัวแปลภาษาไม่สนใจโค้ดในบรรทัดที่ถูกทำหมายเหตุ

*หากต้องการระบุคอมเมนต์ในโปรแกรมให้นำหน้าบรรทัดนั้นด้วยเครื่องหมาย #

โครงสร้างควบคุมพื้นฐาน

คือ กลุ่มคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- คำสั่งแบบลำดับ (Sequence)
- คำสั่งเงื่อนไข (Condition)
- คำสั่งทำซ้ำ (Loop)

IDE คืออะไร

เป็นเครื่องมือสำหรับการใช้ในการเขียนโค้ดเพื่อทดสอบและรันโปรแกรม โดย IDE จะประกอบด้วยส่วนของ **Source Code Editor** สำหรับใช้เขียนโปรแกรม และ ตัวแปลภาษารวมถึง **Debugger** ที่ช่วยตรวจสอบข้อผิดพลาดในโปรแกรม

ใน Python จะมีเครื่องมือมาตรฐานสำหรับเริ่มต้นเขียนโค้ดและทดสอบโปรแกรมเรียกว่า **IDLE Python**

รูปแบบ IDLE Python

- เขียนโปรแกรมขนาดเล็กด้วย **Python Shell** สำหรับทดลองการทำงานคำสั่งต่างๆ ภายในโปรแกรมที่มีการทำงานง่ายๆ ไม่ซับซ้อน
- เขียนโปรแกรมในรูปแบบ **Source Code** จะใช้สำหรับพัฒนาโปรแกรมที่มีการทำงานที่ซับซ้อนโดยบันทึกเป็น **ไฟล์นามสกุล .py**

แสดงผลข้อมูล (Output)

คือคำสั่งสำหรับใช้แสดงผลข้อมูลออกจากจอภาพ
ทั้งข้อมูลที่อยู่ในรูปแบบ ตัวเลขและตัวอักษรหรือ
ผลลัพธ์จากการประมวลผล

แสดงผลข้อมูล (Output)

`print()` คือฟังก์ชันมาตรฐานในภาษา Python ทำหน้าที่
แสดงผลข้อมูล โดยมีโครงสร้างคำสั่ง ดังนี้

```
print(ข้อมูลที่ 1 , ข้อมูลที่ 2 ,.....)
```

ถ้าข้อมูลอยู่ในรูปแบบข้อความ (สตริง) ให้เขียนไว้ในเครื่องหมาย “ ”

หมายเหตุ (Comment)

จุดประสงค์

- อธิบายหน้าที่หรือความหมายของโค้ดที่เขียน
- ยกเลิกโค้ดชั่วคราว ส่งผลให้ตัวแปลภาษาไม่สนใจโค้ดในบรรทัดที่ถูกทำหมายเหตุ

หมายเหตุ (Comment)

วิธีที่ 1 ใช้เครื่องหมาย # ใช้ในการอธิบายคำสั่งสั้นๆ
ในรูปแบบบรรทัดเดียว(เขียนด้านบน หรือ ด้านหลัง)

วิธีที่ 2 เขียนคำอธิบายไว้ในเครื่องหมาย “”” (Triple Quote) ใช้
ในการอธิบายคำสั่งยาวๆหรือแบบหลายบรรทัด

ชนิดข้อมูล (Data Type)

คือ สิ่งที่ใช้กำหนดลักษณะการจัดเก็บและจัดการข้อมูลภายในโปรแกรมคอมพิวเตอร์ โดยในภาษา Python จะแบ่งออกเป็น

- ชนิดข้อมูลพื้นฐาน (Primitive Data Type)
- ชนิดข้อมูลแบบอ้างอิง (Non-Primitive Data Type หรือ Reference Data Types)

ชนิดข้อมูล (Data Type)

คือ สิ่งที่ใช้กำหนดลักษณะการจัดเก็บและจัดการข้อมูลภายในโปรแกรมคอมพิวเตอร์ โดยในภาษา Python จะแบ่งออกเป็น

- ชนิดข้อมูลพื้นฐาน (Primitive Data Type)
- ชนิดข้อมูลแบบอ้างอิง (Non-Primitive Data Type หรือ Reference Data Types)

ชนิดข้อมูลพื้นฐาน

กลุ่มชนิดข้อมูล	ชนิดข้อมูล
ข้อมูลประเภทตัวเลข	int , float , complex
ข้อมูลประเภทตรรกศาสตร์	bool
ข้อมูลประเภทสายอักขระ	str
ข้อมูลแบบไม่ระบุค่า	NoneType

ข้อมูลประเภทตัวเลข (Numbers)

- จำนวนเต็ม (Integers) เป็นตัวเลขจำนวนเต็มบวกหรือจำนวนลบที่ไม่มีจุดทศนิยม
- จำนวนทศนิยม (Floating Point) เป็นตัวเลขที่มีจุดทศนิยม
- จำนวนเชิงซ้อน (Complex) เป็นตัวเลขจำนวนเชิงซ้อน

ตัวอย่าง

- 10 // int
- 99.99 // float
- 3+5j // complex

ข้อมูลประเภทตรรกศาสตร์

เป็นชนิดข้อมูลที่มีค่าได้เพียง 2 ค่า คือ

- True (จริง)
- False (เท็จ)

ข้อมูลประเภทสายอักขระ (String)

เป็นชนิดข้อมูลที่เก็บข้อความหรือกลุ่มตัวอักษร โดยจะเขียน

ข้อความครอบด้วยเครื่องหมาย “” (Double Quote) เช่น

- “Hello Python”
- “KongRuksiam”

ข้อมูลแบบไม่ระบุค่า (None Type)

None หรือค่าว่าง นิยามขึ้นมาเพื่อเก็บข้อมูลที่ไม่สามารถคาดเดาได้ว่าข้อมูลนั้นมีประเภทข้อมูลแบบใด ยกตัวอย่าง เช่น

ไม่มีค่าใดๆ

ไม่เท่ากับ false

ไม่เท่ากับค่าว่างใน String

ไม่ตรงกับชนิดข้อมูลใดเลย

ไม่เท่ากับ 0

ตัวแปร (Variable)

หมายถึง **ชื่อที่ถูกระบุขึ้นมา** เพื่อใช้เก็บค่าข้อมูลสำหรับนำไปใช้งานในโปรแกรม โดยข้อมูลประกอบด้วย ข้อความ ตัวเลข ตัวอักษร หรือผลลัพธ์จากการประมวลผล

*ข้อมูลที่เก็บในตัวแปร สามารถเปลี่ยนแปลงค่าได้

การสร้างตัวแปร

- ชื่อตัวแปร = ค่าเริ่มต้น
- ชื่อตัวแปรที่ 1 , ชื่อตัวแปรที่ N = ค่าเริ่มต้นที่ 1 , ค่าเริ่มต้นที่ N

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

ตัวอย่างการสร้างตัวแปร

- `name="ก้อง"`
- `age = 30`

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

ตัวอย่างการสร้างตัวแปร



The diagram consists of two curved arrows. The first arrow is green and starts from the left, pointing towards the word 'ก้อง' in the text below. The second arrow is purple and starts from the right, pointing towards the number '30' in the text below. Both arrows are positioned above the text 'name , age = "ก้อง" , 30'.

`name , age = "ก้อง" , 30`

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

กฎการตั้งชื่อ

- ขึ้นต้นด้วยตัวอักษร A-Z หรือ a-z หรือ _ เครื่องหมายขีดเส้นใต้เท่านั้น
- อักษรตัวแรกห้ามเป็นตัวเลข
- ตัวพิมพ์เล็ก-พิมพ์ใหญ่มีความหมายต่างกัน (Case Sensitive)
- ห้ามใช้อักขระพิเศษมาประกอบเป็นชื่อตัวแปร เช่น {}, %, ^ และช่องว่าง เป็นต้น
- ไม่ซ้ำกับคำสงวนหรือคำสั่งแบบพิเศษ (keyword) ในภาษา Python

ตัวอย่างคำสงวน (Keyword)

and	as	break	class	continue
def	elif	else	except	False
finally	for	if	in	is
None	not	or	pass	return
True	try	while	import	from

การรับข้อมูลจากผู้ใช้ (Input)

คำสั่งสำหรับรับค่าผ่านทางคีย์บอร์ดและเก็บค่า
ดังกล่าวลงในตัวแปร มีโครงสร้างคำสั่ง ดังนี้

ตัวแปร = input(ข้อความที่จะแสดงผลก่อนรับข้อมูล)

การแปลงชนิดข้อมูล (Casting)

- **int(ข้อมูล)** #แปลงข้อมูลชนิดอื่นเป็นตัวเลขจำนวนเต็ม
- **float(ข้อมูล)** #แปลงข้อมูลชนิดอื่นเป็นตัวเลขที่มีจุดทศนิยม
- **str(ข้อมูล)** #แปลงข้อมูลชนิดอื่นเป็นข้อความหรือสตริง

ตัวดำเนินการ (Operators)

กลุ่มของเครื่องหมายหรือสัญลักษณ์ที่ใช้ในการเขียนโปรแกรม

$$A+B$$

- ตัวดำเนินการ (Operator)
- ตัวถูกดำเนินการ (Operand)

ประเภทของตัวดำเนินการ

- ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)
- ตัวดำเนินการกำหนดค่า (Assignment Operators)
- ตัวดำเนินการเปรียบเทียบ (Comparison Operators)
- ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)
- ตัวดำเนินการเอกลักษณ์ (Identity Operators)
- ตัวดำเนินการสมาชิก (Membership Operators)

ตัวดำเนินการทางคณิตศาสตร์

เครื่องหมาย	คำอธิบาย
+	บวก
-	ลบ
*	คูณ
/	หาร
%	หารเอาเศษ
**	ยกกำลัง
//	หารไม่เอาเศษ (ไม่มีจุดทศนิยม)

ตัวดำเนินการกำหนดค่า

เครื่องหมาย	รูปแบบการเขียน	ความหมาย
<code>+=</code>	<code>x+=y</code>	<code>x=x+y</code>
<code>-=</code>	<code>x-=y</code>	<code>x=x-y</code>
<code>*=</code>	<code>x*=y</code>	<code>x=x*y</code>
<code>/=</code>	<code>x/=y</code>	<code>x=x/y</code>
<code>%=</code>	<code>x%=y</code>	<code>x=x%y</code>
<code>**=</code>	<code>x**=y</code>	<code>x = x**y</code>
<code>//=</code>	<code>x//=y</code>	<code>x = x // y</code>

ตัวดำเนินการเปรียบเทียบ

ผลลัพธ์จากการเปรียบเทียบจะได้ค่าทางตรรกศาสตร์ (True / False)

เครื่องหมาย	คำอธิบาย	ตัวอย่าง
==	เท่ากับ	$a==b$
!=	ไม่เท่ากับ	$a!=b$
>	มากกว่า	$a>b$
<	น้อยกว่า	$a<b$
>=	มากกว่าเท่ากับ	$a>=b$
<=	น้อยกว่าเท่ากับ	$a<=b$

คำสั่งเงื่อนไข

คือ กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if Statement
- Match Statement

คำสั่งเงื่อนไข

คือ กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if Statement
- Match Statement

รูปแบบของ IF Statement

- if
- if-else
- if-elif-else

รูปแบบที่ 1

if condition : #เงื่อนไข

คำสั่งต่างๆเมื่อเงื่อนไขเป็นจริง

รูปแบบที่ 2

if condition :

คำสั่งต่างๆเมื่อเงื่อนไขเป็นจริง

else :

คำสั่งต่างๆเมื่อเงื่อนไขเป็นเท็จ

รูปแบบที่ 2 (เขียนแบบลดรูป)

Ternary Operator

เงื่อนไขเป็นจริง **if** condition **else** เงื่อนไขเป็นเท็จ

รูปแบบที่ 3

if condition 1 :

คำสั่งต่างๆเมื่อเงื่อนไขที่ 1 เป็นจริง

elif condition n :

คำสั่งต่างๆเมื่อเงื่อนไขที่ n เป็นจริง

else :

คำสั่งต่างๆเมื่อทุกเงื่อนไขเป็นเท็จ

คำสั่งที่เกี่ยวข้อง

pass คือ เป็นคำสั่งที่ระบุว่าจะไม่มีการทำงานใดๆ
เกิดขึ้นเลย ต้องการให้โปรแกรมรันได้ก่อน

ใช้ในกรณีที่มีการวางโครงสร้างโปรแกรมเอาไว้
แต่ยังไม่ได้ระบุรายละเอียดการทำงานลงไป

ตัวดำเนินการทางตรรกศาสตร์

Operator	คำอธิบาย
and	และ
or	หรือ
not	ไม่

ตัวดำเนินการทางตรรกศาสตร์

A	not A
True	False
False	True

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

คำสั่งทำซ้ำ

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โดยโปรแกรมจะทำงานไปเรื่อยๆตามเงื่อนไขที่กำหนดไว้

- While Loop
- For Loop

ข้อแตกต่างและการใช้งาน Loop

- For ใช้ในกรณีรู้จำนวนรอบการทำงานซ้ำที่ชัดเจน
- While ใช้ในกรณีที่ไม่รู้จำนวนรอบการทำงานซ้ำ

คำสั่งทำซ้ำ

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โดยโปรแกรมจะทำงานไปเรื่อยๆตามเงื่อนไขที่กำหนดไว้

- While Loop
- For Loop

โครงสร้างคำสั่ง While Loop

while condition :

คำสั่งที่ต้องการทำซ้ำเมื่อเงื่อนไขเป็นจริง

หลักการทำงาน

```
while condition :  
    #statement
```

Output

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

แสดงตัวเลข 1-3

หลักการทำงาน

```
counter=0
```

```
while counter<3 :
```

```
    counter+=1
```

```
    print(counter)
```

Output

หลักการทำงาน

```
counter=0
```

```
while counter<3 :
```

```
    counter+=1
```

```
    print(counter)
```

Output

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1

หลักการทำงาน

```
counter=0
while counter<3 :
    counter+=1
    print(counter)
```

Output

- 1
- 2

หลักการทำงาน

```
counter=0
while counter<3 :
    counter+=1
    print(counter)
```

Output

- 1
- 2

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1
- 2

หลักการทำงาน

```
counter=0
while counter<3 :
    counter+=1
    print(counter)
```

Output

- 1
- 2

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1
- 2
- 3

หลักการทำงาน

```
counter=0  
while counter<3 :  
    counter+=1  
    print(counter)
```

Output

- 1
- 2
- 3

หลักการทำงาน

```
counter=0
while counter<3 :
    counter+=1
    print(counter)
```

Output

- 1
- 2
- 3

#จบการทำงาน

คำสั่งทำซ้ำ

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โดยโปรแกรมจะทำงานไปเรื่อยๆตามเงื่อนไขที่กำหนดไว้

- While Loop
- For Loop

โครงสร้างคำสั่ง For Loop

for ตัวแปร in range(จำนวนรอบ) :

คำสั่งที่ต้องการทำซ้ำ

* จำนวนรอบจะเริ่มต้นที่ 0 และเพิ่มค่าขึ้นทีละ 1 ค่า

โครงสร้างคำสั่ง For Loop

for ตัวแปร in range(ค่าเริ่มต้น , จำนวนรอบ) :

คำสั่งที่ต้องการทำซ้ำ

for ตัวแปร in range(ค่าเริ่มต้น , จำนวนรอบ , การเปลี่ยนแปลงค่า) :

คำสั่งที่ต้องการทำซ้ำ

คำสั่งเกี่ยวกับการทำซ้ำ

- **break** ถ้าโปรแกรมพบคำสั่งนี้จะหลุดจากการทำงานในลูปทันที และไปทำคำสั่งอื่นที่อยู่นอกลูป
- **continue** คำสั่งนี้จะทำให้หยุดการทำงานแล้วย้อนกลับไปเริ่มต้นการทำงานที่ต้นลูปใหม่

Nested-Loop

ในการเขียนโปรแกรมสามารถนำคำสั่ง
รูปแบบต่างๆ ให้มาทำงานซ้อนกันได้เรียกว่า
“ ลูปซ้อนลูป (Nested Loop) ”

Nested-Loop

ลูปหลัก (ลูปนอก) :

ลูปย่อยที่ n (ลูปใน) :

#คำสั่งที่ต้องการทำซ้ำ

Nested-Loop

```
for i in range(.....) : (ลูปนอก)
```

```
    for j in range(.....):
```

```
        #คำสั่งที่ต้องการทำซ้ำ
```


Nested-Loop

```
for i in range(.....) :
```

```
    for j in range(.....) : (ลูปใน)
```

```
        #คำสั่งที่ต้องการทำซ้ำ
```

ตัวอย่าง Nested Loop

```
for i in range(2):  
    for j in range(3):  
        pass
```

ตัวอย่าง Nested Loop

```
for i in range(2):  
    for j in range(3):  
        pass
```

Loop นอกทำงาน 2 รอบ

Loop ในทำงาน 3 รอบ

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

Loop นอก

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

Loop นอก

0

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

Loop นอก

0

Loop ใน

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

Loop นอก

0

Loop ใน

0

1

2

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

Loop นอก

0

1

Loop ใน

0

1

2

ตัวอย่าง Nested Loop

```
for i in range(2):  
    print(i)  
    for j in range(3):  
        print(j)
```

Loop นอก

0

1

Loop ใน

0

1

2

0

1

2

เจาะลึกการใช้งาน String

- การเชื่อมต่อ String (Concatenation)
- String แบบหลายบรรทัด (Three Double Quote)
- จัดรูปแบบ String (Format String)
- การเข้าถึงตัวอักษรใน String (Slice)

เจาะลึกการใช้งาน String

- การเชื่อมต่อ String (Concatenation)
- String แบบหลายบรรทัด (Three Double Quote)
- จัดรูปแบบ String (Format String)
- การเข้าถึงตัวอักษรใน String (Slice)

การเชื่อมต่อ String

ใช้เครื่องหมาย + ในการเชื่อมต่อ String เช่น

“Hello” + “Python”

เจาะลึกการใช้งาน String

- การเชื่อมต่อ String (Concatenation)
- String แบบหลายบรรทัด (Three Double Quote)
- จัดรูปแบบ String (Format String)
- การเข้าถึงตัวอักษรใน String (Slice)

String แบบหลายบรรทัด

ใช้เครื่องหมาย “” (Double Quote 3 ตัว)

“”

ข้อความบรรทัดที่ 1

ข้อความบรรทัดที่ N

“”

เจาะลึกการใช้งาน String

- การเชื่อมต่อ String (Concatenation)
- String แบบหลายบรรทัด (Three Double Quote)
- จัดรูปแบบ String (Format String)
- การเข้าถึงตัวอักษรใน String (Slice)

จัดรูปแบบ String (Format String)

โครงสร้างคำสั่งของ F-String

- **f** “{ตัวแปร}”
- **f** “{Expression}”
- **f** “{ตัวแปรชนิดตัวเลข :.จำนวนทศนิยมf}”

เจาะลึกการใช้งาน String

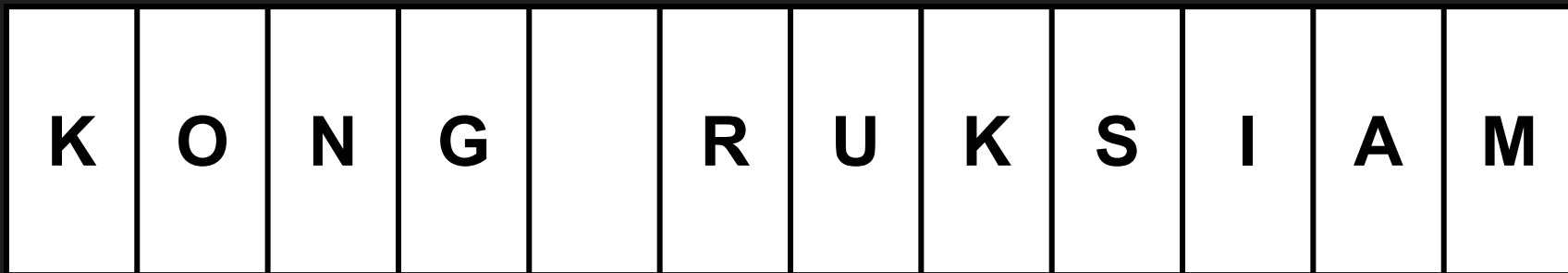
- การเชื่อมต่อ String (Concatenation)
- String แบบหลายบรรทัด (Three Double Quote)
- จัดรูปแบบ String (Format String)
- การเข้าถึงตัวอักษรใน String (Slice)

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
---	---	---	---	--	---	---	---	---	---	---	---

name="KONG RUKSIAM"

องค์ประกอบภายใน String



ความยาวของสตริง (len) = 12

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
---	---	---	---	--	---	---	---	---	---	---	---

*ช่องว่างก็ถือว่าเป็นตัวอักษร

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
---	---	---	---	--	---	---	---	---	---	---	---

index คือ หมายเลขกำกับหรือลำดับตัวอักษรในสตริง
(ตัวเลขจำนวนเต็ม)

องค์ประกอบภายใน String

[illegible]

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11



ลำดับจากซ้ายไปขวาหรือหน้าไปหลังเลขจะเป็นศูนย์หรือค่าบวก

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1



ลำดับจากขวาไปซ้ายหรือหลังไปหน้าเลขจะเป็นค่าลบ

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

name="KONG RUKSIAM"

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`name[0]` หรือ `name[-12]`

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`name[11]` หรือ `name[-1]`

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุดข้อความ
2. [: index สุดท้าย + 1] #จากเริ่มต้นข้อความถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. **[index เริ่มต้น :]** #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุดข้อความ
2. **[: index สุดท้าย + 1]** #จากเริ่มต้นข้อความถึงก่อนตำแหน่งสุดท้าย
3. **[index เริ่มต้น : index สุดท้าย + 1]**

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

name[5 :]

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`name[5 :]` #เริ่มต้น index ที่ 5 ถึง index สุดท้ายของข้อความ

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

name[-4 :] #เริ่มต้น index ที่ -4 ถึง index สุดท้ายของข้อความ

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุดข้อความ
2. [: index สุดท้าย + 1] #จากเริ่มต้นข้อความถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`name[:4]` # ดึงตัวอักษรตัวแรกถึงก่อน index ลำดับที่ 4

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุดข้อความ
2. [: index สดท้าย + 1] #จากเริ่มต้นข้อความถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สดท้าย + 1]

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

name[index เริ่มต้น , index สุดท้าย + 1]

ดึงตัวอักษรจาก index เริ่มต้นก่อนถึง index ตำแหน่งสุดท้าย

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`name[0:4]` # ดึงตัวอักษรลำดับที่ 0 จนถึงก่อนลำดับที่ 4

องค์ประกอบภายใน String

K	O	N	G		R	U	K	S	I	A	M
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`name[-7:-4]` # ดึงตัวอักษรลำดับที่ -7 จนถึงก่อนลำดับที่ -4

ฟังก์ชันจัดการ String

กลุ่มคำสั่งพิเศษที่มีอยู่ในภาษา Python สำหรับจัดการ
ชุดข้อความหรือข้อมูลประเภทสตริง

โครงสร้างคำสั่ง


ตัวแปรประเภทสตริง.ชื่อฟังก์ชัน()

ฟังก์ชันจัดการ String

ชื่อฟังก์ชัน	คำอธิบาย
upper()	แปลงตัวอักษรใน String เป็นตัวพิมพ์ใหญ่
lower()	แปลงตัวอักษรใน String เป็นตัวพิมพ์เล็ก

ชนิดข้อมูล (Data Type)

คือ สิ่งที่ใช้กำหนดลักษณะการจัดเก็บและจัดการข้อมูลภายในโปรแกรมคอมพิวเตอร์ โดยในภาษา Python จะแบ่งออกเป็น

- ชนิดข้อมูลพื้นฐาน (Primitive Data Type) 
- ชนิดข้อมูลแบบอ้างอิง (Non-Primitive Data Type หรือ Reference Data Types)

ข้อจำกัดของชนิดข้อมูลพื้นฐาน

การประกาศตัวแปรแต่ละครั้ง ตัวแปร 1 ตัวสามารถเก็บข้อมูลได้แค่ 1 ค่าเท่านั้น เช่น

score1 = 100

score2 = 80

score3 = 65

name1 = “ก้อง”

name2 = “โจ้”

name3 = “นัท”

ตัวอย่าง

`score1 = 100`

`score2 = 80`

`score3 = 65`

score1



100

score2



80

score3



65

ข้อจำกัดของชนิดข้อมูลพื้นฐาน

“ ถ้าอยากเก็บเลข 10 ค่าต้องทำอะไร
ต้องประกาศตัวแปรจำนวน 10 ตัวแปร หรือไม่ ? ”

ตัวอย่าง

score1 = 100

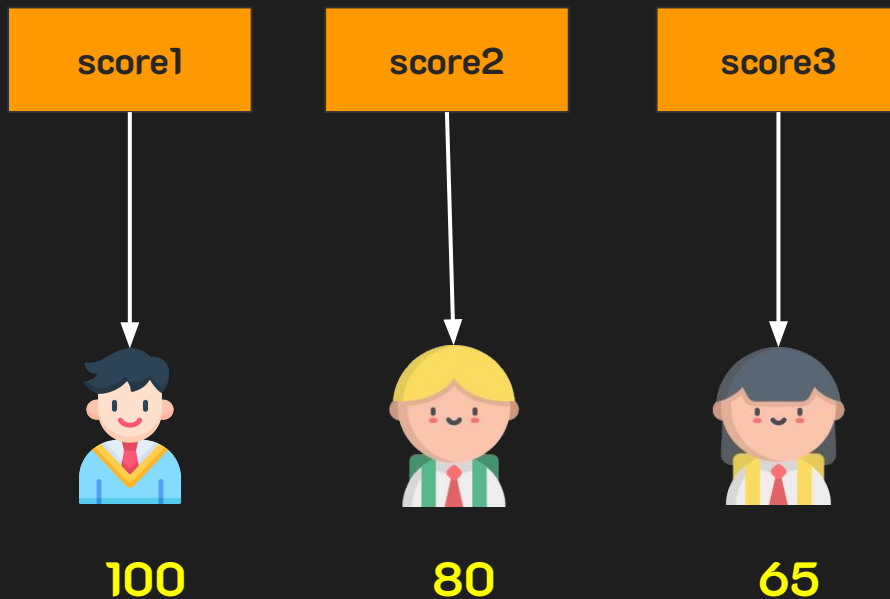
score2 = 80

score3 = 65

score4 = 80

score5 = 90

scoreN = xx



ชนิดข้อมูล (Data Type)

คือ สิ่งที่ใช้กำหนดลักษณะการจัดเก็บและจัดการข้อมูลภายในโปรแกรมคอมพิวเตอร์ โดยในภาษา Python จะแบ่งออกเป็น

- ชนิดข้อมูลพื้นฐาน (Primitive Data Type)
- ชนิดข้อมูลแบบอ้างอิง (Non-Primitive Data Type หรือ Reference Data Types)

ชนิดข้อมูลแบบอ้างอิง

เป็นชนิดข้อมูลแบบพิเศษ ที่ใช้จัดการกลุ่มข้อมูล หรือ
การเก็บข้อมูลหลายๆค่าเอาไว้ด้วยกัน ทำให้เราจัดการ
ข้อมูลได้ง่ายและสะดวกมากขึ้นอีก อีกทั้งยังมีความยืดหยุ่น
สามารถเพิ่มและลดขนาดได้เองอัตโนมัติตามข้อมูลที่มีอยู่
ซึ่งประกอบด้วย List , Tuple , Set , Dictionary

ชนิดข้อมูลแบบอ้างอิง

- List ใช้เก็บข้อมูลแบบลำดับ สามารถเก็บข้อมูลซ้ำกันได้และแก้ไขข้อมูลได้
- Tuple ใช้เก็บข้อมูลแบบลำดับ สามารถเก็บข้อมูลซ้ำกันได้แต่แก้ไขข้อมูลไม่ได้
- Set ใช้เก็บข้อมูลแบบไม่มีลำดับ เก็บข้อมูลซ้ำกันไม่ได้
- Dictionary เก็บความสัมพันธ์ของข้อมูลในลักษณะ key , value

ลิสต์ (List)

เป็นชนิดข้อมูลที่สามารถเก็บกลุ่มของข้อมูลประเภทเดียวกันหรือต่างประเภทกันได้ โดยข้อมูลที่เก็บในลิสต์นั้นจะถูกครอบไว้ด้วยเครื่องหมาย []

ลิสต์ (List)

ลิสต์ถูกนำมาใช้ในการเก็บข้อมูลที่มีลำดับที่ต่อเนื่อง ซึ่งข้อมูลมีค่าได้หลายค่าโดยใช้ชื่ออ้างอิงเพียงชื่อเดียว และใช้หมายเลขกำกับ (index) เพื่อเข้าถึงตำแหน่งของข้อมูลที่เก็บภายในลิสต์

คุณสมบัติของลิสต์ (List)

1. ใช้เก็บกลุ่มของข้อมูล โดยข้อมูลสามารถซ้ำกันได้
2. ข้อมูลที่อยู่ในลิสต์จะเรียกว่าสมาชิก หรือ อิลิเมนต์ (element)
3. แต่ละอิลิเมนต์จะเก็บค่าข้อมูล (value) และ อินเด็กซ์ (Index)
4. Index หมายถึงคีย์ที่ใช้อ้างอิงตำแหน่งของ element เริ่มต้นที่ 0
5. สมาชิกที่เก็บในลิสต์มีชนิดข้อมูลเหมือนกันหรือต่างกันได้
6. สมาชิกในลิสต์จะถูกค้นด้วยเครื่องหมายคอมม่า

การสร้างลิสต์ (List)

โครงสร้างคำสั่งแบบที่ 1

- ชื่อตัวแปร = [ข้อมูล, ...];
ถ้าไม่มีข้อมูลเริ่มต้นจะระบุเป็น []

การสร้างลิสต์ (List)

โครงสร้างคำสั่งแบบที่ 2

- ชื่อตัวแปร = `list(ข้อมูล, ...)`

การจัดการลิสต์ (List)

- จำนวนสมาชิกในลิสต์
- การเชื่อมข้อมูลในลิสต์
- เข้าถึงสมาชิกในลิสต์
- การแก้ไขข้อมูลในลิสต์

จำนวนสมาชิกในลิสต์ (List)

```
score = [100, 90, 80]
```

```
len(score) #จำนวนสมาชิกในลิสต์
```

100	90	80
-----	----	----

len = 3

เชื่อมข้อมูลในลิสต์ (List)

```
score1 = [100,90,80]
```

```
score2 = [50,30,45]
```

```
ตัวแปรลิสต์ใหม่ = score1 + score2
```

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-----	-------	---------	-----	----	-----	--------

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-----	-------	---------	-----	----	-----	--------

**index คือ หมายเลขกำกับหรือลำดับสมาชิกในลิสต์
(ตัวเลขจำนวนเต็ม)**

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6



ลำดับจากซ้ายไปขวาหรือหน้าไปหลังเลขจะเป็นศูนย์หรือค่าบวก

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-7	-6	-5	-4	-3	-2	-1



ลำดับจากขวาไปซ้ายหรือหลังไปหน้าเลขจะเป็นค่าลบ

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[0] หรือ [-7]

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[6] หรือ [-1]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. [: index สุดท้าย + 1] #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. **[index เริ่มต้น :]** #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. **[: index สุดท้าย + 1]** #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. **[index เริ่มต้น : index สุดท้าย + 1]**

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[2:]

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[-3:]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. [: index สุดท้าย + 1] #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6

[: 3]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. [: index สุดท้าย + 1] #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[1:4]

การเข้าถึงสมาชิกในลิสต์ (List)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[-7 : -4]

การเข้าถึงสมาชิกในลิสต์ (Loop)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-----	-------	---------	-----	----	-----	--------

```
for item in list_name:
```

```
    #คำสั่งต่างๆ
```

แก้ไขข้อมูลในลิสต์ (List)

```
score = [100, 90, 80]
```

```
score[index] = value
```

- index = ตำแหน่งข้อมูลใน List
- value = ข้อมูลใหม่

ฟังก์ชันจัดการ List

กลุ่มคำสั่งพิเศษที่มีอยู่ในภาษา Python สำหรับจัดการ
ข้อมูลประเภทลิสต์

โครงสร้างคำสั่ง

ตัวแปรประเภทลิสต์.ชื่อฟังก์ชัน()

ฟังก์ชันจัดการลิสต์ (List)

ชื่อฟังก์ชัน	คำอธิบาย
<code>append(element)</code>	เพิ่มสมาชิกใหม่ 1 รายการ (ต่อท้าย)
<code>extend([element])</code>	เพิ่มสมาชิกใหม่หลายรายการ (ต่อท้าย)
<code>insert(index , element)</code>	แทรกสมาชิกใหม่ลงไปในลิสต์ตาม index ที่กำหนด
<code>clear()</code>	ลบสมาชิกทั้งหมดออกจากลิสต์

ฟังก์ชันจัดการลิสต์ (List)

ชื่อฟังก์ชัน	คำอธิบาย
remove(element)	ลบสมาชิกที่ระบุออกจากลิสต์
count(element)	นับจำนวนสมาชิกที่มีข้อมูลซ้ำกัน
sort()	เรียงลำดับสมาชิก
reverse()	เรียงลำดับสมาชิกแบบย้อนกลับ

ทูเพิล (Tuple)

เป็นชนิดข้อมูลที่ใช้เก็บกลุ่มของข้อมูล มีลักษณะการทำงานคล้ายกับลิสต์ คือใช้เก็บข้อมูลประเภทเดียวกันหรือต่างประเภทกันได้ เก็บข้อมูลซ้ำกันได้ แต่ข้อมูลที่เก็บลงใน Tuple นั้นจะไม่สามารถเปลี่ยนแปลงค่าได้และเขียนครอบด้วยเครื่องหมายวงเล็บ ()

คุณสมบัติของทูเพิล (Tuple)

1. ใช้เก็บกลุ่มของข้อมูล โดยข้อมูลสามารถซ้ำกันได้
2. ข้อมูลที่อยู่ในทูเพิลจะเรียกว่าสมาชิก หรือ อิลิเมนต์ (element)
3. แต่ละอิลิเมนต์จะเก็บค่าข้อมูล (value) และ อินเด็กซ์ (Index)
4. Index หมายถึงคีย์ที่ใช้อ้างอิงตำแหน่งของ element เริ่มต้นที่ 0
5. สมาชิกที่เก็บในทูเพิลมีชนิดข้อมูลเหมือนกันหรือต่างกันได้
6. สมาชิกในทูเพิลจะไม่สามารถแก้ไขหรือเปลี่ยนแปลงได้

การสร้างทูเพิล (Tuple)

โครงสร้างคำสั่งแบบที่ 1

- ชื่อตัวแปร = (ข้อมูล, ...)

ถ้าไม่มีข้อมูลเริ่มต้นจะระบุเป็น ()

การสร้างทูเพิล (Tuple)

โครงสร้างคำสั่งแบบที่ 2

- ชื่อตัวแปร = **tuple**((ข้อมูล, ...))

การจัดการทูเพิล (Tuple)

- จำนวนสมาชิกในทูเพิล
- การเชื่อมข้อมูลในทูเพิล
- การทำซ้ำข้อมูลในทูเพิล
- การกระจายข้อมูลสมาชิกในทูเพิล
- เข้าถึงสมาชิกในทูเพิล

จำนวนสมาชิกในทูเพิล

```
score =(100,90,80)
```

```
len(score) #จำนวนสมาชิก
```

100	90	80
-----	----	----

len = 3

เชื่อมข้อมูลในทูเพิล (Tuple)

```
score1 = (100,90,80)
```

```
score2 = (50,30,45)
```

```
ตัวแปร = score1 + score2
```

การทำซ้ำข้อมูลในทูเพิล

`score1 = (100,90,80)`

`ตัวแปร = score1 * จำนวนที่ทำซ้ำ`

การกระจายข้อมูลสมาชิกในทูเพิล

```
colors = (“แดง”, “เขียว”, “น้ำเงิน”)
```

```
ตัวแปร , ... = colors
```

การกระจายข้อมูลสมาชิกในทูเพิล

```
colors = (“แดง”, “เขียว”, “น้ำเงิน”)
```

```
red , green , blue = colors
```

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-----	-------	---------	-----	----	-----	--------

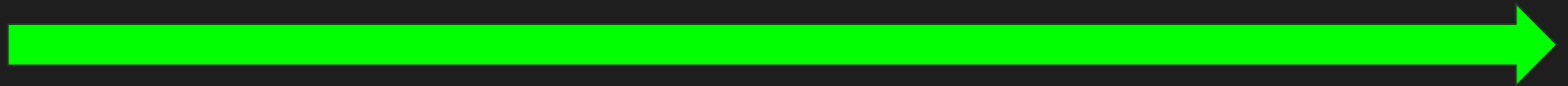
การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-----	-------	---------	-----	----	-----	--------

index คือ หมายเลขกำกับหรือลำดับสมาชิกในทูเพิล
(ตัวเลขจำนวนเต็ม)

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6



ลำดับจากซ้ายไปขวาหรือหน้าไปหลังเลขจะเป็นศูนย์หรือค่าบวก

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-7	-6	-5	-4	-3	-2	-1



ลำดับจากขวาไปซ้ายหรือหลังไปหน้าเลขจะเป็นค่าลบ

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[0] หรือ [-7]

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[6] หรือ [-1]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. [: index สุดท้าย + 1] #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. **[index เริ่มต้น :]** #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. **[: index สุดท้าย + 1]** #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. **[index เริ่มต้น : index สุดท้าย + 1]**

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[2:]

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[-3:]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. [: index สุดท้าย + 1] #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6

[: 3]

แบบกำหนดช่วง

ใช้ร่วมกับเครื่องหมาย :

1. [index เริ่มต้น :] #จาก index ตำแหน่งเริ่มต้นถึงสิ้นสุด
2. [: index สุดท้าย + 1] #จากเริ่มต้นถึงก่อนตำแหน่งสุดท้าย
3. [index เริ่มต้น : index สุดท้าย + 1]

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[1:4]

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

[-7 : -4]

การเข้าถึงสมาชิกในทูเพิล (Tuple)

แดง	เขียว	น้ำเงิน	ส้ม	ดำ	ขาว	เหลือง
-----	-------	---------	-----	----	-----	--------

```
for item in tuple_name:
```

```
    #คำสั่งต่างๆ
```

ฟังก์ชันจัดการทูเพิล (Tuple)

กลุ่มคำสั่งพิเศษที่มีอยู่ในภาษา Python สำหรับจัดการ
ข้อมูลประเภททูเพิล

โครงสร้างคำสั่ง

ตัวแปรประเภททูเพิล.ชื่อฟังก์ชัน()

ฟังก์ชันจัดการทูเพิล (Tuple)

ชื่อฟังก์ชัน	คำอธิบาย
count(element)	นับจำนวนสมาชิกที่มีข้อมูลซ้ำกัน
index(element)	ตรวจสอบลำดับสมาชิกในทูเพิล

เซต (Sets)

เป็นชนิดข้อมูลที่สามารถเก็บกลุ่มของ
ข้อมูลประเภทเดียวกันหรือต่างประเภทกันได้
มีลักษณะการทำงานคล้ายกับทิวเปิลคือข้อมูลที่
เก็บในเซตไม่สามารถเปลี่ยนแปลงค่าได้

เซต (Sets)

ข้อมูลหรือสมาชิกที่เก็บในเซตนั้นต้องมีค่าไม่ซ้ำกัน
และเซตจะไม่มีเลข Index กำกับข้อมูลนั้นหมายถึงข้อมูล
ที่อยู่ในเซตนั้นจะไม่มีกรเรียงลำดับ ข้อมูลที่เก็บในเซต
นั้นจะเขียนครอบด้วยเครื่องหมายปีกกา { }

การสร้างเซต (Sets)

โครงสร้างคำสั่งแบบที่ 1

- ชื่อตัวแปร = {ข้อมูล, ...};

การสร้างเซต (Sets)

โครงสร้างคำสั่งแบบที่ 2

- ชื่อตัวแปร = **set**((ข้อมูล, ...))

จำนวนสมาชิกในเซต

```
score = {100,90,80}
```

```
len(score) #จำนวนสมาชิก
```

100	90	80
-----	----	----

len = 3

การเข้าถึงสมาชิกในเซต (Loop)

```
for item in set_name:
```

```
    #คำสั่งต่างๆ
```

ค้นหาข้อมูลสมาชิกในเซต

ข้อมูลที่ต้องการค้นหา **in** ชื่อตัวแปรเซต

ฟังก์ชันจัดการเซต

กลุ่มคำสั่งพิเศษที่มีอยู่ในภาษา Python สำหรับจัดการ
ข้อมูลประเภทเซต

โครงสร้างคำสั่ง

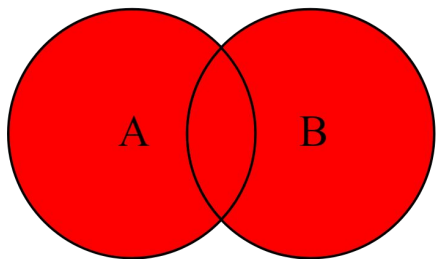
ตัวแปรประเภทเซต.ชื่อฟังก์ชัน()

ฟังก์ชันจัดการ Sets

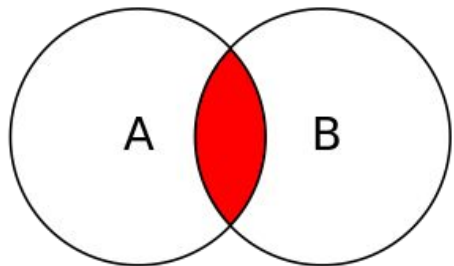
ชื่อฟังก์ชัน	คำอธิบาย
<code>add(element)</code>	เพิ่มสมาชิกใหม่ 1 รายการ
<code>update((element))</code>	เพิ่มสมาชิกใหม่หลายรายการ
<code>discard(element)</code>	ลบสมาชิกที่ระบุออกจาก Sets

การดำเนินการใน Sets

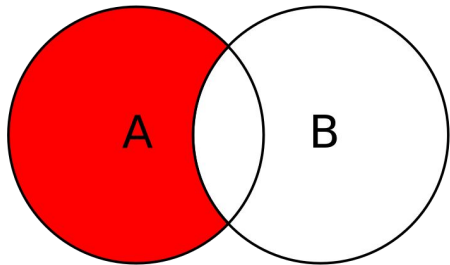
ชื่อฟังก์ชัน	คำอธิบาย
union()	รวมสมาชิกในเซต A และเซต B เข้าด้วยกัน
intersection()	สมาชิกในเซต A และเซต B ที่มีร่วมกัน
difference()	สมาชิกที่มีอยู่ในเซต A แต่ไม่มีอยู่ในเซต B



- **UNION**



- **INTERSECTION**



- **DIFFERENCE**

Dictionary

เป็นรูปแบบการเก็บความสัมพันธ์ของข้อมูลใน
ลักษณะของ **key** และ **value** กล่าวคือ จะใช้ **key** เป็น
index ในการเข้าถึงข้อมูล (**value**) แต่ละตัว
โดยค่า **key** นั้นต้องไม่ซ้ำกัน และสามารถกำหนด
ประเภทข้อมูลได้

การสร้าง Dictionary

โครงสร้างคำสั่ง

```
ชื่อตัวแปร={  
    “key 1” : value 1,  
    “key n” : value n  
}
```

ตัวอย่าง

```
colors = {  
    “red” : “สีแดง”,  
    “yellow” : “สีเหลือง”  
}
```

การเข้าถึงและเพิ่มข้อมูล

- `variable["key"]` #ดึงข้อมูล
- `variable["key"] = value` #แก้ไขข้อมูล
- `variable["new_key"] = value` #เพิ่มข้อมูล

ตัวอย่างการเพิ่มข้อมูล

```
colors={  
    “red” : “สีแดง”,  
    “yellow” : “สีเหลือง”  
}  
colors[“blue”] = “สีน้ำเงิน”
```

ฟังก์ชันจัดการ Dictionary

กลุ่มคำสั่งพิเศษที่มีอยู่ในภาษา Python สำหรับจัดการ
ข้อมูลประเภท Dictionary

โครงสร้างคำสั่ง

ตัวแปรประเภทDict.ชื่อฟังก์ชัน()

ฟังก์ชันจัดการ Dictionary

ชื่อฟังก์ชัน	คำอธิบาย
keys()	keys ทั้งหมดใน Dictionary
values()	values ทั้งหมดใน Dictionary
get(key)	ดึงข้อมูลใน Dictionary ตาม key ที่ระบุ
items()	ดึงข้อมูล keys และ values ทั้งหมด

ฟังก์ชันจัดการ Dictionary

ชื่อฟังก์ชัน	คำอธิบาย
<code>clear()</code>	ลบสมาชิกทั้งหมดออกจาก Dictionary
<code>pop(key)</code>	ลบข้อมูลใน Dictionary ตาม key ที่ระบุ
<code>update({key, value})</code>	เพิ่มหรืออัปเดตข้อมูลใน Dictionary

ประเภทของตัวดำเนินการ

ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

ตัวดำเนินการกำหนดค่า (Assignment Operators)

ตัวดำเนินการเปรียบเทียบ (Comparison Operators)

ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)

- ตัวดำเนินการเอกลักษณ์ (Identity Operators)
- ตัวดำเนินการสมาชิก (Membership Operators)

ตัวดำเนินการเอกลักษณ์ (Identity)

ใช้สำหรับเปรียบเทียบข้อมูลว่าเหมือนกันหรือไม่ ?

โดยผลลัพธ์จากการเปรียบเทียบจะได้ค่าทางตรรกศาสตร์มา
ใช้งาน ประกอบด้วยคำสั่ง

- is เหมือนกัน
- is not ไม่เหมือนกัน

ประเภทของตัวดำเนินการ

ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

ตัวดำเนินการกำหนดค่า (Assignment Operators)

ตัวดำเนินการเปรียบเทียบ (Comparison Operators)

ตัวดำเนินการทางตรรกศาสตร์ (Logical Operators)

- ตัวดำเนินการเอกลักษณ์ (Identity Operators)
- ตัวดำเนินการสมาชิก (Membership Operators)

ตัวดำเนินการสมาชิก (Membership)

ใช้สำหรับเปรียบเทียบข้อมูลว่า **อยู่ในกลุ่มข้อมูลหรือไม่ ?**

โดยผลลัพธ์จากการเปรียบเทียบจะได้ค่าทางตรรกศาสตร์มา
ใช้งาน ประกอบด้วยคำสั่ง

- in **อยู่ในกลุ่ม**
- not in **ไม่อยู่ในกลุ่ม**

ฟังก์ชัน (Function)

ชุดคำสิ่งที่นำมาเขียนรวมกันเป็นกลุ่มเพื่อเรียกใช้งานตามวัตถุประสงค์ที่ต้องการและลดความซ้ำซ้อนของคำสิ่งที่ใช้งานบ่อย

ฟังก์ชันสามารถนำไปใช้งานได้ทุกที่และแก้ไขได้ในภายหลัง
ทำให้โค้ดในโปรแกรมมีระเบียบและใช้งานได้สะดวกมากยิ่งขึ้น

รูปแบบของฟังก์ชัน (Function)

- ฟังก์ชันมาตรฐาน (Built-in Functions) คือ ฟังก์ชันที่มีอยู่ในภาษา Python สามารถเรียกใช้งานได้ทันที เช่น `print()` , `input()`
- ฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง (User-Defined Function) คือ ฟังก์ชันที่ถูกสร้างขึ้นมาให้ทำงานตามวัตถุประสงค์ที่ผู้ใช้ต้องการ

การสร้างฟังก์ชัน (Function)

- ฟังก์ชันแบบปกติ
- ฟังก์ชันแบบมีพารามิเตอร์
- ฟังก์ชันแบบมีค่าส่งกลับ
- ฟังก์ชันแบบรับและส่งค่า

กฎการตั้งชื่อฟังก์ชัน

- ชื่อฟังก์ชันไม่ควรตั้งชื่อไม่ซ้ำกัน
- ชื่อฟังก์ชันสามารถตั้งเป็นตัวอักษรหรือตัวเลขได้
- ชื่อของฟังก์ชันต้องไม่ขึ้นต้นด้วยตัวเลข

คำสั่งเกี่ยวกับฟังก์ชัน

pass คือ เป็นคำสั่งที่ระบุว่าจะไม่มีการทำงานใดๆ
เกิดขึ้นเลยในฟังก์ชัน ต้องการให้โปรแกรมรันได้ก่อน
ใช้ในกรณีที่มีการวางโครงสร้างฟังก์ชันเอาไว้
แต่ยังไม่ได้ระบุรายละเอียดการทำงานลงไป

ฟังก์ชันแบบปกติ

โครงสร้างคำสั่ง

```
def ชื่อฟังก์ชัน():
```

```
    // คำสั่งต่างๆ
```

การเรียกใช้งานฟังก์ชัน

```
ชื่อฟังก์ชัน()
```

ฟังก์ชันแบบมีพารามิเตอร์ (Parameter)

โครงสร้างคำสั่ง

```
def ชื่อฟังก์ชัน(parameter1,parameterN,.....):
```

```
    #กลุ่มคำสั่งต่างๆ
```

การเรียกใช้งานฟังก์ชัน

```
ชื่อฟังก์ชัน (argument1,argumentN,.....)
```

ฟังก์ชันแบบมีพารามิเตอร์ (Parameter)

โครงสร้างคำสั่ง

```
def ชื่อฟังก์ชัน(parameter1,parameterN,...):
```

```
#กลุ่มคำสั่งต่างๆ
```

การเรียกใช้งานฟังก์ชัน

```
ชื่อฟังก์ชัน (argument1,argumentN,...)
```

- อาร์กิวเมนต์ คือ ตัวแปรหรือค่าที่ต้องการส่งมาให้กับฟังก์ชัน (ตัวแปรส่ง)
- พารามิเตอร์ คือ ตัวแปรที่ฟังก์ชันสร้างไว้สำหรับรับค่าที่จะส่งเข้ามาให้กับฟังก์ชัน (ตัวแปรรับ)

ข้อกำหนดการใช้งานพารามิเตอร์

- มีชนิดข้อมูลเหมือนกันหรือต่างกันได้
- มีพารามิเตอร์กี่ตัวก็ได้ ขึ้นอยู่กับวัตถุประสงค์ในการใช้งาน
โดยจะคั่นพารามิเตอร์แต่ละตัวด้วยเครื่องหมายคอมม่า
- มีขอบเขตการใช้งานเฉพาะพื้นที่ภายในฟังก์ชัน (Local Variable)
- สามารถนำไปใช้งานในฟังก์ชันได้คล้ายกับการใช้งานตัวแปร

ฟังก์ชันแบบมีค่าส่งกลับ

โครงสร้างคำสั่ง

```
def ชื่อฟังก์ชัน():
```

```
    return ค่าที่จะส่งออกไป
```

การเรียกใช้งานฟังก์ชัน

```
ตัวแปรที่รับค่าจากฟังก์ชัน = ชื่อฟังก์ชัน()
```