เอกสารประกอบ

2301172 COMP PROG LAB

ภาคต้น 2563

จารุโลจน์ จงสถิตย์วัฒนา

แจ้งข้อผิดพลาดในเอกสารที่ jaruloj@gmail.com ใช้ subject: errata 2301172

CONTENTS

DATA TYPES, VARIABLES, EXPRESSIONS, AND SOME FUNCTIONS

ค่าคงที่: INTEGERS, FLOATS, STRINGS, TUPLES

ค่าคงที่ (constant) มีหลายชนิด ค่าคงที่แต่ละชนิดมีรูปแบบหรือหน้าตาที่ต่างกัน ดังนี้

Integers	ถูก	`34`
		`-90`
		`0`
			ความหมาย	จำนวนเต็ม 34
				จำนวนเต็ม -90
				จำนวนเต็ม 0

	ผิด	`34.0`	ทำไม	ใส่จุดแล้วจะกลายเป็นจำนวนจริง

Floats	ถูก	`34.0`
		`-90.`
		`3.1e4`
		`0.8E-3`
			ความหมาย	จำนวนจริง 34
				จำนวนจริง -90
				จำนวนจริง 31000
				จำนวนจริง 0.0008

	ผิด	`3.1e1.8`	ทำไม	เลขชี้กำลังเป็นจำนวนจริงไม่ได้

Strings	ถูก	`'13.8'`
		`"it's"`
		`"""It's "no"."""`
		`''''It's gone."''''`
		`'\n', '\t'`
			ความหมาย	สตริงที่เอา 1 3 . 8 มาต่อกัน
				สตริงที่เอา `I t ' s` มาต่อกัน
				สตริงที่เอา `I t ' s " n o " .` มาต่อกัน
				สตริงที่เอา `" I t ' s g o n e . "` มาต่อกัน
				ตัวอักขระแทนการขึ้นบรรทัดใหม่ และ tab

	ผิด	`"Say "no""` `"Test'`	ทำไม	เจอ " ตัวที่สอง ก็ปิดสตริงไปแล้ว ตัวเปิดกับตัวปิดต้องเหมือนกัน

Tuples	ถูก	`(5, 'days')`
		`5,`
		`x, y`
			ความหมาย	ทูเปิลที่เป็นคู่ของค่า 5 กับ สตริง ‘days’
				ทูเปิลที่เป็นค่าเดียว คือ 5
				ทูเปิลที่เป็นคู่ของค่าในตัวแปร x และ y

	ผิด	`(5, 6]` `5`	ทำไม	ตัวปิดต้องเป็น) ถ้าจะไม่ใส่วงเล็บ ต้องมี ,

ตัวแปร และ คำสั่งกำหนดค่าให้ตัวแปร

ตัวแปร (variable) ใช้เก็บค่าที่ใช้ในระหว่างโปรแกรมทำงาน ตัวแปรในภาษาไพธอน**เปลี่ยนชนิดไปตามค่าที่เก็บ** คำสั่งกำหนดค่าให้ตัวแปร (assignment statement) เป็นวิธีหนึ่งที่เอาค่าไปเก็บค่าในตัวแปร

ตัวแปร

ชื่อตัวแปรต้องขึ้นต้นด้วยตัวอักษร a-z A-Z ตามด้วยตัวเลข ตัวอักษร หรือ เครื่องหมาย _ (ไม่ใช่ -) เช่น

`test1`, `_hname`, `max_score`, `top_3_page`

- ต้องไม่มี เว้นวรรค (blank) หรือ tab ในชื่อตัวแปร เพราะใช้สำหรับแบ่งชิ้นส่วนในโปรแกรม
- ต้องไม่มีสัญลักษณ์พิเศษ เช่น `+ - = , :` เพราะสัญลักษณ์เหล่านี้มีความหมายเฉพาะในโปรแกรม เมื่อเจอแล้วจะแยกชิ้นออกมา เช่น `x+y` ถูกแยกเป็นตัวแปร x ตัวกระทำ + และ ตัวแปร y
- ตัวแปรในภาษาไพธอนเปลี่ยนชนิดไปตามค่าที่เก็บ
- ควรตั้งชื่อตัวแปรให้สื่อความหมาย เพื่อให้อ่านโปรแกรมเข้าใจง่าย

คำสั่งกำหนดค่าให้ตัวแปร

คำสั่งกำหนดค่าให้ตัวแปร อยู่ในรูป

ตัวแปร = นิพจน์

- **ตัวแปรต้องอยู่ทางซ้าย**ของ = เสมอ เช่น `temp = x/1.8+32`
- `x=y` หมายถึงเอาค่าในตัวแปร y ไปเก็บในตัวแปร x ดังนั้น ค่าใน x และ y จะเหมือนกัน
- `x,y = 1,'a'` หมายถึงเอาค่าทูเปิล `(1,'a')` ไปเก็บในทูเปิล `(x,y)` นั่นคือ เอาค่า 1 และ `'a'` ไปเก็บในตัวแปร x และ y ตามลำดับ
- ถ้าใช้ค่าในตัวแปรโดยยังไม่ได้เอาค่าไปเก็บในตัวแปร จะเกิด error (Name .. is not defined) หรือเอาค่าในตัวแปรชื่อนั้นที่ใช้ในโปรแกรมที่ทำงานไว้ก่อนมาใช้ ซึ่งอาจได้ค่าที่ไม่ถูกต้อง

EXAMPLE

ถ้าต้องการสลับค่าที่เก็บในตัวแปร x กับตัวแปร y ลองดูโปรแกรม 4 แบบข้างล่างนี้

สมมติว่าก่อนโปรแกรมทำงาน x เก็บค่า 5 และ y เก็บค่า 9

`y = x`
`x = y`
	`x = y`
	`y = x`
		`t = x`
		`x = y`
		`y = t`
			`t = y`
			`y = x`
			`x = t`
				`x,y = y,x`	`(x,y) = (y,x)`

ผลลัพธ์	ผลลัพธ์	ผลลัพธ์	ผลลัพธ์	ผลลัพธ์	ผลลัพธ์
`x: 5` `y: 5`	`x: 9` `y: 9`	`x: 9` `y: 5`	`x: 9` `y: 5`	`x: 9` `y: 5`	`x: 9` `y: 5`

ฟังก์ชันเบื้องต้น

ฟังก์ชันรับค่าที่เรียกว่า**พารามิเตอร์**หรือ **argument** เข้าไป เพื่อทำงาน แล้วคืนค่ามาให้เหมือนฟังก์ชันทางคณิตศาสตร์ เช่น sin(π/2) รับพารามิเตอร์ π/2 และ คืนค่า 1

เราเรียกใช้ฟังก์ชันโดยบอก

- ชื่อฟังก์ชัน และ
- พารามิเตอร์ที่อยู่ในวงเล็บตามหลังชื่อ และ คั่นด้วย , ถ้ามีพารามิเตอร์มากกว่าหนึ่งตัว

เช่น `sin(3.14159/2)` คืนค่า 1 , `log(x*y, 2)` คืนค่า $\log_2(x+y)$

TYPE

ฟังก์ชัน type บอกชนิดของค่าที่เป็นพารามิเตอร์ โดยพารามิเตอร์อาจเป็นค่าคงที่ ตัวแปร หรือ นิพจน์

ฟังก์ชัน type รับค่าคงที่ ตัวแปร หรือ นิพจน์ และคืนค่าเป็นชนิดของค่าคงที่ ตัวแปร หรือ นิพจน์ นั้น

EXAMPLE

`type(2.8+3)`	คืนค่า `float`
`type(x)`	คืนค่า `int` ถ้า x เก็บสตริง `'-8'`
`type(x*3)`	คืนค่า `float` ถ้า x เก็บจำนวนจริง `-6.5`
`type('x='+x)`	คืนค่า `str` ถ้า x เก็บสตริง `'-8'`
`type('29')`	คืนค่า `str`

INT

ฟังก์ชัน int แปลงค่าของพารามิเตอร์ที่รับเป็นจำนวนเต็ม ถ้าแปลงได้

ฟังก์ชัน int รับค่าคงที่ ตัวแปร หรือ นิพจน์ และคืนค่าของค่าคงที่ ตัวแปร หรือ นิพจน์ นั้นที่แปลงเป็นจำนวนเต็ม

EXAMPLE

`int(2.8)`	ให้ค่า `2`
`int(x)`	ให้ค่า `-8` ถ้า x เก็บสตริง `'-8'`
`int(8/3)`	ให้ค่า `2`
`int(x/2)`	เกิด error ถ้า x เก็บสตริง `'-8'` เพราะสตริงใช้กับการหารไม่ได้
`int('2A')`	เกิด error เพราะสตริง `'2A'` แปลงเป็นจำนวนเต็มไม่ได้

FLOAT

ฟังก์ชัน float แปลงค่าของพารามิเตอร์ที่รับเป็นจำนวนจริง ถ้าแปลงได้

ฟังก์ชัน float รับค่าคงที่ ตัวแปร หรือ นิพจน์ และคืนค่าของค่าคงที่ ตัวแปร หรือ นิพจน์ นั้นที่แปลงเป็นจำนวนจริง

EXAMPLE

`float(4-2)`	ให้ค่า `2.0`
`float(x)`	ให้ค่า `200.0` ถ้า x เก็บสตริง `'2e2'`
`float('2x4')`	เกิด error เพราะสตริง `'2x4'` แปลงเป็นจำนวนเต็มไม่ได้

STR

ฟังก์ชัน str แปลงค่าของพารามิเตอร์ที่รับเป็นสตริง ถ้าแปลงได้

ฟังก์ชัน str รับค่าคงที่ ตัวแปร หรือ นิพจน์ และคืนค่าของค่าคงที่ ตัวแปร หรือ นิพจน์ นั้นที่แปลงเป็นสตริง

EXAMPLE

`str(-3E3)`	ให้ค่าเป็นสตริง `'-3000.0'`
`str(2+7)`	ให้ค่าเป็นสตริง `'9'`
`str('2+4-1')`	ให้ค่าเป็นสตริง `'2+4-1'`

BOOL

ฟังก์ชัน bool แปลงค่าของพารามิเตอร์ที่รับเป็น `True` หรือ `False` โดยจะแปลงค่า 0 เป็น `False` และ ค่าอื่น ๆ ที่ไม่ใช่ 0 เป็น `True`

EXAMPLE

`bool(3<4)`	ให้ค่าเป็น `True`
`bool(False)`	ให้ค่าเป็น `False`
`bool(0.0)`	ให้ค่าเป็น `False`
`bool(2+7)`	ให้ค่าเป็น `True`
`bool('xx')`	ให้ค่าเป็น `True`

PRINT

ฟังก์ชัน print แสดงค่าทางหน้าจอ โดยให้ใส่ค่าที่ต้องการแสดงเป็นพารามิเตอร์

- ถ้าต้องการพิมพ์หลายค่า ให้ใส่ทุกค่าในวงเล็บและใส่ , คั่น
- เมื่อแสดงค่าทางหน้าจอ จะเว้นวรรค (คือใส่ blank) ระหว่างแต่ละค่า
- เมื่อพิมพ์ค่าสุดท้ายแล้วจะขึ้นบรรทัดใหม่

EXAMPLE

```
print(2,'x')        พิมพ์ 2 x    แล้วขึ้นบรรทัดใหม่รอ
print(2,x)          พิมพ์ 2 90   แล้วขึ้นบรรทัดใหม่รอ
```

- ค่าที่จะพิมพ์ เป็น ค่าคงที่ หรือ ตัวแปร หรือ นิพจน์ ก็ได้
- ค่าที่จะพิมพ์เป็น integer, float, string, หรือ tuple หรือเป็นชนิดอื่นที่จะพูดถึงต่อไป (list, dict) ก็ได้

EXAMPLE

```
print(x+5)                  พิมพ์ 95 ถ้าตัวแปร x เก็บค่า 90
print('temp=',(x+32)*5/9)  พิมพ์ temp= 67.77777777777777 ถ้าตัวแปร x เก็บค่า 90
```

INPUT

ฟังก์ชัน input อ่าน**สตริง**ที่ผู้ใช้พิมพ์เข้ามาทางคีย์บอร์ด โดย

- รับพารามิเตอร์เป็นข้อความ (prompt) ที่จะแสดงเมื่อรอผู้ใช้พิมพ์ข้อความเข้ามา
- แล้วรอรับสตริงจากผู้ใช้ ผู้ใช้พิมพ์อะไรเข้ามาก็ได้จนกด enter
- คืนค่าเป็นสตริงที่ผู้ใช้พิมพ์มา

EXAMPLE

```
name=input('Enter your name')
```

พิมพ์ `'Enter your name'` และรับสตริงจากผู้ใช้ แล้วเก็บไว้ในตัวแปร `name`

- ฟังก์ชัน input รับ**สตริง**ที่ผู้ใช้พิมพ์เข้ามาทางคีย์บอร์ด ถ้าต้องการใช้เป็น**จำนวน** ต้องใช้ฟังก์ชันเปลี่ยนชนิดของค่า `int, float`

EXAMPLE

```
age=int(input('Enter your age'))
```

คำสั่งนี้ทำงานโดย (1) แสดงข้อความ `'Enter your name'` ที่หน้าจอ และรับสตริงจากผู้ใช้ (2) รับสตริงที่ผู้ใช้พิมพ์ (3) ส่งสตริงนั้นเข้าไปในฟังก์ชัน `int` และเปลี่ยนเป็นจำนวนเต็ม (4) เก็บจำนวนเต็มนั้นในตัวแปร `age`

IN

ฟังก์ชัน in ทำงานกับสตริง เขียนเป็นแบบ `x in y` โดย

- `x in y` จะคืนค่า `True` ถ้า สตริง `x` ปรากฏในสตริง `y`
- `x in y` จะคืนค่า `False` ถ้า สตริง `x` ไม่ปรากฏในสตริง `y`

EXAMPLE

```
print('ea' in 'At least')        # print True
print('ea' in January')          # print False
```

SPLIT

ฟังก์ชัน split ทำงานกับสตริง โดยจะแบ่งสตริงหนึ่งเป็นสตริงย่อย ๆ ด้วยตัวแบ่งที่กำหนดเป็นพารามิเตอร์ของฟังก์ชัน แล้วส่งสตริงย่อยทั้งหมดคืนมา การเรียกใช้ฟังก์ชัน split ใช้ สตริง ตามด้วยจุด (`.`) ตามด้วยพารามิเตอร์ในวงเล็บ

EXAMPLE

```
name='Tom-Tan-Tun'
n1,n2,n3 = name.spilt('-')       # n1,n2 and n3  get 'Tom', 'Tan' and  'Tun'
```

เอา `'Tom-Tan-Tun'` แบ่งด้วย `'-'` ซึ่งจะได้สตริง 3 ค่า แล้วเอาค่าทั้ง 3 ค่าไปเก็บในทูเปิลของตัวแปร `n1,n2,n3` ทำให้ `n1` เก็บสตริง `'Tom'` และ `n2` เก็บสตริง `'Tan'` และ `n3` เก็บสตริง `'Tun'`

STRIP

ฟังก์ชัน strip ทำงานกับสตริง โดยจะตัดเว้นวรรค (blank) tab หรือ ขึ้นบรรทัดใหม่ (รวมเรียกว่า whitespace) ที่อยู่ด้านหน้าและด้านหลังของสตริง แต่จะไม่ตัด whitespace ด้านในสตริง

EXAMPLE

```
name=' \t  Tim    \t Tom    Tun   \n   ')
namec=name.strip()
```

จะได้ `namec` เก็บสตริง `'Tim    \t Tom    Tun'`

นิพจน์

นิพจน์ (expression) สร้างจากตัวถูกกระทำ (operands) และ ตัวกระทำ (operators) 3 กลุ่มดังนี้

<-------- ทำก่อนไปหลัง (precedence) --------			ความหมาย	ทำจาก (associativity)	ตัวอย่าง
	Arithmetic Operators	**	ยกกำลัง	ขวาไปซ้าย	x**y**z หมายถึง $x^{(y^z)}$
		+ -	บวก (unary) ลบ (unary)	ขวาไปซ้าย	--y หมายถึง -(-y))
		* / %	คูณ หาร เศษการหาร	ซ้ายไปขวา	x/y%2*z หมายถึง (((x/y)%2)*z)
		+ -	บวก (binary) ลบ (binary)	ซ้ายไปขวา	x-y+z หมายถึง (x-y)+z
	String operators	*	ซ้ำ	ซ้ายไปขวา	2*'A'*3 หมายถึง 'A' ต่อกัน 2 ครั้ง แล้วเอาทั้งหมดมาต่อกัน 3 ครั้ง คือ 'AAAAAA'
		+	ต่อกัน(concatenate)	ซ้ายไปขวา	'123'+' X' หมายถึง '123 X'
	Relational Operators	> >= < <= == !=	มากกว่า มากกว่าหรือเท่ากับ น้อยกว่า น้อยกว่าหรือเท่ากับ เท่ากับ (ระวัง ใช้ = 2 ตัวติดกัน) ไม่เท่ากับ	ซ้ายไปขวา	x<y<=z หมายถึง (x<y) and (y<=z)
	Logical Operators	not	ไม่	ขวาไปซ้าย	not not x==y หมายถึง not(not(x==y))
		and	และ	ซ้ายไปขวา	True and False and True หมายถึง ((True and False) and True)
		or	หรือ	ซ้ายไปขวา	True or False or True หมายถึง ((True or False) or True)

ลำดับการทำงานของตัวกระทำในนิพจน์ขึ้นกับ precedence และ associativity ตามที่แสดงในตารางข้างบน

EXAMPLE

x=(-b+(b**2-4*a*c)**0.5)/(2*a) หมายถึง $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$

x=(-b+(b**2-4*a*c)**0.5)/2*a หมายถึง $x = \frac{-b+\sqrt{b^2-4ac}}{2}a$

x=-b+(b**2-4*a*c)**0.5/(2*a) หมายถึง $x = -b + \frac{\sqrt{b^2-4ac}}{2a}$

x=(-b+b**2-4*a*c**0.5)/(2*a) หมายถึง $x = \frac{-b+b^2-4a\sqrt{c}}{2a}$

COMMENTS

Comment เป็นส่วนที่ไม่ใช่โปรแกรม มักใช้อธิบายการทำงานของโปรแกรม

- ใช้ # แสดงจุดเริ่มต้นของ comment ที่อยู่ในบรรทัดเดียว
- ใช้ '''… ''' ครอบ comment หลายบรรทัด

EXAMPLE

```
a=int(input('Enter age'))        # comment starts at # to end of line
''' To make multiple-line comment,
    Make a string inside the triple quotes
'''
```

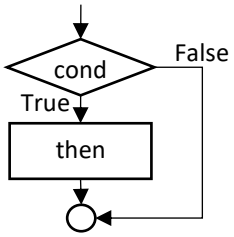
IF STATEMENTS

โครงสร้าง if ใช้สำหรับการทำงานแบบทางเลือก ภาษาไพธอนมีโครงสร้าง if ให้ใช้ได้ 3 แบบ คือ if-then, if-then-else และ if-elif

นอกจากนั้นยังสามารถใส่โครงสร้าง if ซ้อนในโครงสร้าง if ได้อีก เรียกว่า nested if

โครงสร้าง IF-THEN

```
if cond :
   do something    # then part
```



- ใช้บอกว่า ถ้าเงื่อนไข *cond* เป็นจริง ให้ทำอะไร (then part) แต่ถ้า *cond* เป็นเท็จ จะไม่ให้ทำอะไร
- สังเกตว่า สิ่งที่ให้ทำในโครงสร้าง if ต้องย่อหน้า (indent) เข้ามาหนึ่งระดับจากคำว่า if
- ถ้าสิ่งที่ต้องทำประกอบด้วยหลายคำสั่ง เราต้องย่อหน้าให้เท่ากันทุกคำสั่ง เราเรียกคำสั่งกลุ่มนี้ว่า บล็อค (block)

EXAMPLE

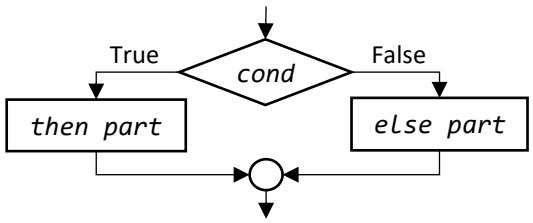
```
age=int(input('Enter your age : '))
if age>=18:
   print('You are already an adult.')         # one-line block
```

EXAMPLE

```
price = float(input('Enter price : '))
if price>=5000:
    discount = price*0.1                      # 1st line of block
    pay = price-discount
    print('Your price:', pay,'Baht.')
    print('Discount  :', discount,'Baht.')    # last line of block
print('Full price: ', price,'Baht.')
```

โครงสร้าง IF-THEN-ELSE

```
if cond :
   do one thing          # then part
else:
   do another thing      # else part
```



- ใช้บอกว่า ถ้าเงื่อนไข *cond* เป็นจริง ให้ทำอะไร (then part) แต่ถ้า *cond* เป็นเท็จ จะให้ทำอะไร (else part)
- ทั้ง then part และ else part ต้องเป็น block ด้วย

EXAMPLE

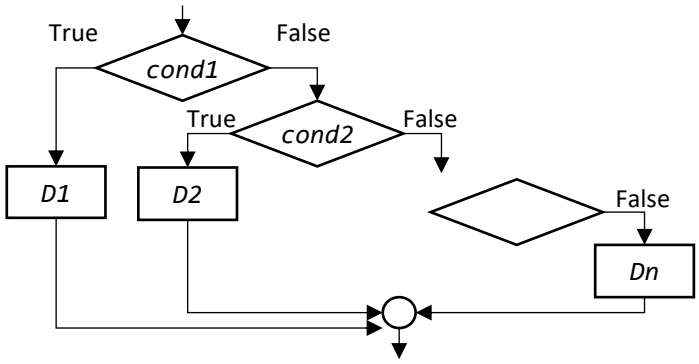
```
age=int(input('Enter your age : '))
if age>=18:
   print('You are already an adult.')
else:
   print('You are not an adult.')
```

EXAMPLE

```
price = float(input('Enter price : '))
if price<5000:
    discount = price*0.1                        # 1st line of block
    pay = price-discount
    print('Your price  :', pay,'Baht.')
    print('10% Discount:', discount,'Baht.')    # last line of block
else:
    discount = 500                              # 1st line of block
    pay = price-discount
    print('Your price  :', pay,'Baht.')
    print('Max Discount:', discount,'Baht.')    # last line of block
print('Full price: ', price,'Baht.')
```

โครงสร้าง IF-ELIF

```
if cond1 :
   do one thing # D1
elif cond2 :
   do another thing # D2
…
else:
   do the last thing # Dn
```



- ใช้บอกว่า
  - ถ้าเงื่อนไข *cond1* เป็นจริง ให้ทำอะไร (*do one thing*)
  - ถ้า *cond1* เป็นเท็จ แต่ *cond2* เป็นจริง ให้ทำอะไร (*do another thing*) …
  - ถ้า *cond1, cond2, …* เป็นเท็จหมด จะให้ทำอะไร (*do the last thing*)
- ในโครงสร้าง if-elif จะเลือกทำงาน *D1*, *D2*, …, *Dn* เพียงอย่างเดียว
- ออกจากโครงสร้างนี้ที่ที่ท้ายโครงสร้างที่เดียวสำหรับทุกกรณี
- ในกรณีสุดท้าย else ที่ไม่มี if ต้องไม่มีเงื่อนไข

EXAMPLE

```
age=int(input('Enter your age : '))
if age>=60:
   print('You are a senior.')
elif age>=35:       # get to this line if age>=60 is false
   print('You are an adult.')
elif age>=18:       # get to this line if age>=60 and age>=35 are false
   print('You are a young adult.')
elif age>=12:       # get to this line if age>=60 and age>=35 and age>=18 are false
   print('You are a teen.')
else:               # get to this line if all previous conditions are false
   print('You are a child.')     ###### NO condition in ELSE ######
```

NESTED IF

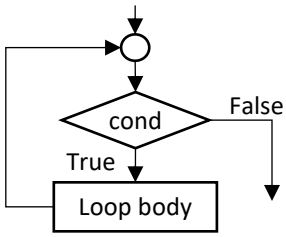
เป็นการใช้โครงสร้าง if ซ้อนกัน นั่นคือ ใน if part และ then part ก็เป็นโครงสร้าง if ด้วย และจะย่อหน้าเพิ่มในแต่ละระดับด้วย

EXAMPLE

```
age=int(input('Enter your age : '))
if age>=35:         # C1
    if age>=60:     # C2
        # get to this line if C1 and C2 are True (age>=60)
        print('You are a senior.')
    else:
        # get to this line if C1 is True and C2 is False (35<=age<60)
        print('You are an adult.')
else:
    if age>=18:     # C3
        # get to this line if C1 is False and C3 is True (18<=age<35)
        print('You are an adult.')
    else:
        if age>=12:   # C4
            # get to this line if C1 and C3 are False and C4 is True (12<=age<18)
            print('You are a teen.')
        else:
            # get to this line if C1 and C3 and C4 are False (age<12)
            print('You are a child.')
```

WHILE LOOPS

```
while cond :
    do something     # called loop body
```



เป็นโครงสร้างโปรแกรมที่ให้ทำ *Loop body* ซ้ำๆ โดยจะทดสอบเงื่อนไข *cond* ก่อน

- ถ้า *cond* เป็นจริง จะทำ *Loop body* แล้ววนไปเช็คเงื่อนไขซ้ำอีก
- ถ้า *cond* เป็นเท็จ จะเลิกวนซ้ำและไปทำคำสั่งหลัง *Loop body*

ถ้าเงื่อนไขเป็นเท็จตั้งแต่ครั้งแรก จะไม่เข้าไปทำงานในลูปเลย

การวนซ้ำ N รอบ

รูปแบบง่ายๆ ของการวนซ้ำแบบหนึ่ง คือ การวนซ้ำ n รอบ ดังนี้

PATTERN

```
i=0                 # set counter to 0
while i<n:          # loop while the counter does not exceed n
    # do something
    i=i+1           # increase the counter
```

EXAMPLE: พิมพ์สูตรคูณแม่ 13

```
i=1
k=13
while i<=12:        # loop 12 times because i starts at 1 and use <=12
    print(i,'x',k,'=',i*k)
    i=i+1
```

EXAMPLE: พิมพ์ M TO N

```
i=m
while i<=n:         # loop n-m+1 times
    print(i)
    i=i+1
```

EXAMPLE: วนรับชื่อ n รอบ

```
i=0
while i<n:          # loop n times
    name=input('Enter a name :')
    print(i,'.',name)
    i=i+1
```

การวนซ้ำทั่วไป

While loop ยังใช้ในรูปแบบทั่วไป คือ วนซ้ำเมื่อเงื่อนไขเป็นจริง

วนรับค่าเมื่อค่านั้นทำให้เงื่อนไขเป็นจริง

EXAMPLE

```
x=int(input('Enter a number:'))         # get the first input
while x>0:                              # test the input on condition
    print(x)                            # print input from the previous round
    x=int(input('Enter a number:'))     # get the next input
```

- รับค่าครั้งหนึ่งก่อนเข้าลูป (ต้องรับค่าก่อนเข้าลูป เพื่อให้สามารถทดสอบเงื่อนไขก่อนเข้าลูปครั้งแรกได้)
- เงื่อนไขที่หัวลูป คือ ทดสอบว่าค่าที่รับมาตรงเงื่อนไข
- ค่าที่พิมพ์ต้องตรงตามเงื่อนไข เพราะผ่านการตรวจสอบที่หัวลูปก่อนพิมพ์

วนรับค่าจนได้ค่าที่ตรงตามเงื่อนไข

EXAMPLE

```
x=int(input('Enter a number:'))         # get the first input
while x<=0:                             # valid input is a positive integer
    x=int(input('Enter a number:'))     # get the next input
print('input is',x)              # when exit from the loop, the last input is valid
```

- รับค่าครั้งหนึ่งก่อนเข้าลูป
- เงื่อนไขที่หัวลูป คือ ทดสอบว่าค่าที่รับมาไม่ตรงตามเงื่อนไข
- เมื่อออกจากลูปมา ค่าสุดท้ายที่รับมาต้องตรงตามเงื่อนไข

การคำนวณแบบเก็บสะสม (ACCUMULATE) ค่าที่รับมา

PATTERN

```
# initialize accumulator variable
i=m          # set first value
while i<n:   # check last value
    # accumulate in the variable
    i=i+1    # next value
```

EXAMPLE: SUMMATION M TO N

```
total=0             # initialize accumulator variable
i=m                 # set first value
while i<=n:         # the last value is n
    total=total+i   # accumulate in the variable
    i=i+1           # next value
print(total)
```

EXAMPLE: LOOP AND READ INTEGERS UNTIL GET 0, AND SUM ALL INPUTS.

```
total=0                          # initialize accumulator variable
x=int(input('Enter a number:'))  # get the first input
while x!=0:                      # stop when get 0 as input
    total=total+x                # accumulate in the variable
    x=int(input('Enter a number:')) # next input
print(total)
```

EXAMPLE: FACTORIAL N

สามารถมองการหา factorial เป็นการเก็บสะสมค่าได้ แต่เป็นการสะสมค่าโดยการคูณแทนการบวก ดังนั้น ค่าเริ่มต้นของตัวสะสมค่าเป็น 1

```
f=1          # initialize accumulator variable
i=1          # set first value
while i<=n:  # the last value is n
    f=f*i    # accumulate in the variable
    i=i+1    # next input
print(f)
```

EXAMPLE: LOOP AND READ INTEGERS UNTIL GET 0, AND COUNT ALL INPUTS.

สามารถมองการนับจำนวนข้อมูลที่รับเป็นการเก็บสะสมค่าได้ โดยการบวก 1 ดังนั้น ค่าเริ่มต้นของตัวสะสมค่าเป็น 0

```
all=0                                # initialize accumulator variable
x=int(input('Enter a number:'))      # get the first input
while x!=0:                          # stop when get 0 as input
    all=all+1                        # accumulate in the variable
    x=int(input('Enter a number:'))  # get the next input
```

การทดสอบค่าทีละค่า

EXAMPLE: GET N INTEGERS AND PRINT ONLY EVEN NUMBERS.

```
i=0                                  # set counter to 0
n=int(input('Enter n : '))           # input n (to loop n times)
while i<n:                           # loop while counter does not exceed n
    x=int(input('Enter a number:'))  # get input
    if x%2==0:                       # test input to print
        print(x)
    i=i+1                            # increment the counter
```

EXAMPLE: GET INTEGERS WHILE INPUTS ARE POSITIVE INTEGERS, AND PRINT ONLY EVEN NUMBERS.

```
x=int(input('Enter a number:'))        # get the first input
while x>0:                             # loop while the input is positive
    if x%2==0:                         # test input to print
        print(x)
    x=int(input('Enter a number:'))    # get the next input
```

การทดสอบว่ามีค่าที่ตรงตามเงื่อนไขอย่างน้อยหนึ่งค่า / ทุกค่า / ไม่มีแม้แต่ค่าเดียว

EXAMPLE: GET N INTEGERS AND CHECK IF **NONE** IS EVEN.

```
i=0
flag=True                  # set flag to True (none of the values is even)
n=int(input('Enter n : '))
while i<n:
    x=int(input('Enter a number:'))
    if x%2==0:             # find an even number
       flag=False          # change the flag to False (one of the values is even)
    i=i+1
if flag:  print('None is even.')
```

EXAMPLE: GET N INTEGERS AND CHECK IF **SOME** (AT LEAST ONE) OF THEM ARE EVEN.

```
i=0
flag=False                 # set flag to False (NOT some of the values is even)
n=int(input('Enter n : '))
while i<n:
    x=int(input('Enter a number:'))
    if x%2==0:             # find an even number
       flag=True           # change the flag to Ture (sone of the values is even)
    i=i+1
if flag:  print('Some are even.')
```

EXAMPLE: GET N INTEGERS AND CHECK IF **ALL** ARE EVEN.

```
i=0
flag=True                  # set flag to True (all of the values is even)
n=int(input('Enter n : '))
while i<n:
    x=int(input('Enter a number:'))
    if x%2!=0:             # find NOT an even number
       flag=False          # set flag to False (NOT all of the values is even)
    i=i+1
if flag:  print('All are even.')
```

คำสั่ง BREAK/ CONTINUE

break ใช้เพื่อให้กระโดด**ออก**จากลูป

continue ใช้เพื่อให้กระโดดไปที่**ต้นลูป**ใหม่ (ไม่ต้องทำคำสั่งที่อยู่หลังจากนั้นในลูป)

EXAMPLE: CHECKING PRIME NUMBER.

```
i=2
prime=True
n=int(input('Enter n : '))
while i<n:
    if n%i==0:            # check if i divides n
        prime=False       # know that n is not a prime
        break             # no need to check further, exit from the loop
    i=i+1
if prime:  print(n,' is prime.')
else: print(n,' is not prime.')
```

EXAMPLE: FIND PRIME FACTORS.

```
i=2                 # first value to check if it is a factor
n=int(input('Enter n : '))
while n>=i:
   if n%i!=0:       # check if i divides n
      i=i+1         # check the next integer as a factor
      continue      # Jump to the start of loop / no need to change n
   print(i)         # here: i divides n (i is a factor of n)
   n=n/i            # start again to find factors of n/i
   i=2              # reset the first value to factor
```

ลูปซ้อนลูป (NESTED LOOP)

EXAMPLE: FIND PRIME FACTOR, AND PRINT EACH FACTOR ONLY ONCE.

```
i=2
n=int(input('Enter n : '))
while n>=i:
   if n%i!=0:
      i=i+1
      continue
   print(i)
   while n%i==0:    # divide until n is not divisible by i
      n=n/i
   i=2
```

EXAMPLE: พิมพ์สูตรคูณ แม่ 2 - 10

```
i=2
while i<=10:
   j=1
   while j<=12:     # print multiplication table for i
      print(i,'x',j,'=',i*j)
      j=j+1
   i=i+1
```

RANGE และสตริง

RANGE และสตริง ที่เป็น ITERATOR

- range(start, stop, step) เป็น iterator ที่สร้างชุดของจำนวนเต็ม start, start+step, start+2*step, …ที่มีค่าไม่ถึง stop
- สตริงเป็น iteraror ที่เป็นชุดของ character ในสตริง เริ่มจากตัวแรกไปจนตัวสุดท้าย
- **for** *var* **in** *iterator* : เป็นการวนลูป ซึ่งในการวนแต่ละรอบ จะดึงค่าหนึ่งค่าในชุดของ iterator มาไว้ในตัวแปร *var* แล้วเอาค่าในตัวแปรมาใช้งานในรอบนั้น
- การใช้ for loop จะกำหนดค่าที่จะใช้ในแต่ละรอบด้วย iterator เมื่อเทียบกับการใช้ while loop ที่ต้องเขียนโปรแกรมให้คำนวณค่าที่จะใช้ในแต่ละรอบ

EXAMPLE: PRINT M TO N.

```
m=int(input('Enter m : '))
n=int(input('Enter n : '))
for i in range(m,n+1):     # range as an iterator
   print(i)                # print m, m+1, m+2, …,
```

EXAMPLE: GET 20 INTEGERS.

```
for i in range(20):        # range(20) makes 0,1,2,…,19
   x=int(input('Enter a number:'))
   print(x)                # i is used as a counter
```

EXAMPLE: COUNT ‘A’ IN YOUR NAME.

```
cnt=0
name=input('Enter your name : ')
for c in name:             # string as an iterator
   if c=='a':              # c is each character in the string
      cnt=cnt+1
print(cnt)
```

การวนซ้ำสะสมค่า

ใช้ตัวแปรที่สะสมค่าและ operator ที่สะสมค่า เหมือนที่ทำในบทที่แล้ว แต่เปลี่ยนการวนซ้ำแบบ while มาเป็นแบบ for

PATTERN

```
# initialize accumulator variable
for i in ...:       # specify the collection of values
    # accumulate in the variable
```

EXAMPLE: SUMMATION M TO N

```
total=0                    # initialize accumulator variable
for i in range(m, n+1):    # range(m, n+1) => m, m+1, m+2, …, n
    total=total+i          # accumulate i in the accumulator
print(total)
```

EXAMPLE: FACTORIAL N

```
f=1                        # initialize accumulator variable
for i in range(2,n+1):     # range(2, n+1) => 2, 3, 4, …, n
    f=f*i                  # accumulate i in the accumulator (by multiplication)
print(f)
```

EXAMPLE: CREATE A SENTENCE FROM WORDS GIVEN AS INPUT.

```
n=20
sentence=''                # initialize accumulator variable
for i in range(n):         # use for i in range(n) to loop n times
    word=input('Enter a word : ').strip()     # input a word
    sentence=sentence+word+' '   # accumulate sentence by concatenation
print(sentence)
```

การวนซ้ำตรวจสอบค่า

ใช้ if ตรวจสอบค่า เหมือนที่ทำในบทที่แล้ว แต่เปลี่ยนการวนซ้ำแบบ while มาเป็นแบบ for

PATTERN

```
for i in ...:       # specify the collection of values
    # test the value in i
```

EXAMPLE: FIND INTEGERS THAT DIVIDE N.

```
n=int(input('Enter n : '))
for i in range(2,n):       # range(2,n) => 2,3,…, n-1
    if n%i==0:
        print(n, 'is divisible by', i)
```

EXAMPLE: FIND DIGITS IN A SENTENCE.

```
sentence= input('Enter a sentence : '))
cnt=0
for c in sentence:          # the variable c is each character in sentence
    if '0'<=c<='9':         # can also use c.isdigit()
        cnt=cnt+1
print(cnt)
```

การวนซ้ำตรวจสอบว่า ทุกค่า / บางค่า / ไม่มีค่าใด ตรงตามเงื่อนไข

ตั้งค่าเริ่มต้น และเปลี่ยน flag เหมือนที่ทำในบทที่แล้ว แต่เปลี่ยนการวนซ้ำแบบ while มาเป็นแบบ for

PATTERN

```
# set flag
for i in ...:       # specify the collection of values
    if i ... :      # test the value in i for the condition
       # reset flag
       break        # can also exit from for loop when flag is reset
```

EXAMPLE: CHECKING PRIME

```
prime=True                  # set flag
n=int(input('Enter n : '))
for i in range(2, n):       # range(2,n) is the set of values to try to divide n
    if n%i==0:
        prime=False         # reset flag
        break               # can exit from the loop when the flag is already reset
if prime:
   print(n,'is prime.')
else:
   print(n,'is a composite.')
```

EXAMPLE: CHECKING PASSWORD CONTAIN AT LEAST ONE NUMBER (SOME)

```
has_digit=False             # set flag
pwd=input('Enter password : '))
for c in pwd:               # In each round, c is each character in the password
   if '0'<=c<='9':
      has_digit=True        # reset flag
      break
if has_digit:
   print(pwd,'is ok.')
else:
   print(pwd,'is invalid.')
```

EXAMPLE: CHECKING USER NAME CONTAIN ONLY CAPITAL LETTERS (ALL)

```
only_cap=True
name=input('Enter user name : '))
for c in name:            # In each round, c is each character in the name
   if not('A'<=c<='Z'):   # can also use c.isupper()
      only_cap=False
      break
if only_cap:
   print(name,' contains capital letters only.')
else:
   print(name,' contains non-capital letters.')
```

EXAMPLE: CHECKING USER NAME CONTAIN NO SPECIAL CHARACTER (NONE)

```
no_sp=True                # set flag
name=input('Enter user name : '))
for c in name:
   if c in '''!@#$%^&*()_+=-{}[]|\"':;?/>.<,''':
   # a in b is True if character a is in string b
      no_sp=False         # reset flag
      break
if no_sp:
   print(name,' contain no special character.')
else:
   print(name,' contain some special characters.')
```

ลิสต์และไฟล์

ค่าคงที่ชนิดลิสต์

- ลิสต์เป็น iterator ชนิดหนึ่ง
- ใช้ [ ] ครอบสมาชิกในลิสต์ โดยที่สมาชิกแต่ละตัวคั่นด้วยเครื่องหมาย , เช่น

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

- สมาชิกในลิสต์ไม่จำเป็นต้องเป็นชนิดเดียวกัน เช่น

```
[1,2,3,4,'no answer']
```

- สมาชิกในลิสต์เป็นลิสต์ก็ได้ เช่น

```
[1,[2,3],[4],[]]
```

- [ ] หมายถึง ลิสต์ว่าง (empty list)

การอ้างถึงสมาชิกแต่ละตัวในลิสต์

- สามารถอ้างถึงสมาชิกแต่ละตัวในลิสต์โดยการระบุตำแหน่ง โดยตำแหน่งแรก คือ 0 เช่น

```
days=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
print(days[1])          # print 'Monday'
days[0]='*Monday*'      # change the first element from 'Monday' to '*Monday*'
```

- ถ้าใช้ตำแหน่งเป็นเลขลบ จะเป็นการนับย้อนจากสมาชิกตัวสุดท้ายในลิสต์ โดยตำแหน่งสุดท้าย คือ -1 ตำแหน่งก่อนสุดท้าย คือ -2 ... เช่น

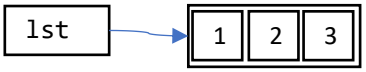
```
prime10=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
print(prime10[-1])            # print 29
y = prime10[-4]+prime10[6]    # get y = 17+17
```

- ถ้าอ้างถึงตำแหน่งที่เกินจากที่มีในลิสต์ จะเกิด error เช่น

```
prime10=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
print(prime10[10])            # error because positive index goes from 0 to 9
y = prime10[-11]+prime10[1]   # error because negative index goes from -1 to -10
```

การกำหนดค่าลิสต์ให้ตัวแปร

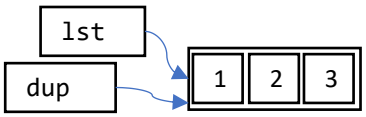
- ในการกำหนดค่าคงที่ลิสต์ให้ตัวแปร ประกอบด้วยการสร้างค่าคงที่ลิสต์ในหน่วยความจำ แล้วโยงชื่อตัวแปรกับที่เก็บลิสต์ในหน่วยความจำ เช่น




```
lst=[1,2,3]
```

- ถ้าใช้การกำหนดค่าให้ตัวแปร (assignment) เอาตัวแปรที่เก็บลิสต์ไปเก็บในอีกตัวแปรหนึ่ง เช่น

```
lst=[1,2,3]
dup=lst   # dup and lst share the same list
```



ตัวแปรทั้ง 2 ตัวจะอ้างถึงลิสต์เดียวกัน (ไม่มีการสร้างลิสต์ใหม่) ดังนั้น การเปลี่ยนค่าในลิสต์จะส่งผลกับทั้งสองตัวแปร เช่น

```
lst[1]=0              # change both lst and dup
print(dup[1])         # also get 0
```

- ถ้าต้องการให้สร้างลิสต์ใหม่ก่อนที่จะเอาไปเก็บในอีกตัวแปร เราสามารถใช้ฟังก์ชัน list เพื่อเปลี่ยนลิสต์เป็นลิสต์ ซึ่งจะทำให้มีการสร้างลิสต์ใหม่ขึ้นมา ก่อนจะเอาไปเก็บในตัวแปรใหม่ เช่น

```
lst=[1,2,3]
anotherList=list(lst)  # anotherList and lst do not share the same list
lst[1]='*'             # change only lst, but does not change anotherList
print(lst[1])          # get '*'
print(anotherList[1])  # get 1
```

ฟังก์ชันสำหรับลิสต์

หาความยาวของลิสต์

- ฟังก์ชัน len ใช้หาจำนวนสมาชิกในลิสต์เพื่อการคำนวณหรือไม่ให้อ้างถึงสมาชิกเกินตำแหน่งที่มีในลิสต์

EXAMPLE:

```
prime=[2, 3, 5, 8, 11, 13, 17, 19, 23, 29]
print(len(prime))         # get 10
```

IN

ฟังก์ชัน in ทำงานกับลิสต์ เขียนเป็นแบบ `x in y` โดย

- `x in y` จะคืนค่า `True` ถ้า ค่า `x` เป็นสมาชิกของลิสต์ `y`
- `x in y` จะคืนค่า `False` ถ้า ค่า `x` ไม่เป็นสมาชิกของลิสต์ `y`

EXAMPLE

```
print(1 in [1, [2,3], [4,5,6])          # print True
print([2, 3] in [1, [2,3], [4,5,6])     # print True
print(2 in [1, [2,3], [4,5,6])          # print False
print([4,5] in [1, [2,3], [4,5,6])      # print False
```

เพิ่มสมาชิกในลิสต์

- ฟังก์ชัน append ใช้เพิ่มสมาชิกต่อท้ายลิสต์ แล้วคืนค่า None
- ฟังก์ชัน insert ใช้แทรกสมาชิกเพิ่มในลิสต์ที่ตำแหน่งที่กำหนด แล้วคืนค่า None

โดยมีรูปแบบการเรียกใช้ดังนี้

EXAMPLE:

```
prime = [2, 3, 5, 7, 11, 13, 17]
prime.append(23)          # prime is changed to [2, 3, 5, 7, 11, 13, 17, 23]
print(prime.append(23))   # None is printed because the function append returns None
prime.insert(7,19)        # prime is changed to [2, 3, 5, 7, 11, 13, 17, 19, 23]
npm = prime+[29]          # npm is [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
                          # prime is not changed
```

ต่อลิสต์ 2 ลิสต์

- ตัวกระทำ + นำลิสต์ 2 ลิสต์มาต่อกันเป็นลิสต์ใหม่ โดยไม่เปลี่ยนค่าลิสต์เดิม

EXAMPLE:

```
prime = [2, 3, 5, 7, 11, 13, 17]
npm = prime+[19, 23, 29]    # npm is [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
                            # prime is not changed
ls1=[1,3,5]
lst2=['a', 'b']
lst=lst1+lst2               # lst is [1, 3, 5, 'a', 'b']
```

การลบค่าในลิสต์

- ฟังก์ชัน del ลบค่าในลิสต์โดยระบุว่าชื่อของลิสต์และตำแหน่งของสมาชิกที่ต้องการลบ
- ฟังก์ชัน remove หาค่าที่ระบุในลิสต์ เริ่มจากตำแหน่งที่ 0 แล้วลบค่าแรกที่เจอในลิสต์ และคืนค่า None

EXAMPLE

```
prime=[2, 3, 5, 7, 11, 13, 17, 19, 11, 23, 29]
del(prime[2])       # delete element at position 2 from list
                    # prime is changed to [2, 3, 7, 11, 13, 17, 19, 11, 23, 29]
prime.remove(11)    # delete the first element value 11 from prime
                    # prime is changed to [2, 3, 7, 13, 17, 19, 11, 23, 29]
```

การเรียงค่าในลิสต์

- ฟังก์ชัน sort เรียงค่าในลิสต์ แล้วคืนค่า None (เรียงค่าตามการเปรียบเทียบด้วย operator <)

```
score = [6, 5, 7, 9, 2, 4, 8, 1]
score.sort()                # score is changed to [1, 2, 4, 5, 6, 7, 8, 9]

cptl = ['Paris', 'Bangkok', 'Soul', 'London', 'Tokyo']
cptl.sort()
# cptl is changed to ['Bangkok', 'London', 'Paris', 'Soul', 'Tokyo']

sLst = [[2,4], [6], [2,3], [1,2,3]]
sLst.sort()
# sLst is changed to [[1, 2, 3], [2, 3], [2, 4], [6]]
```

- ถ้าใช้ฟังก์ชัน sort เมื่อค่าในลิสต์ไม่สามารถเปรียบเทียบกันด้วย operator < จะเกิด error

```
cptl = [3, 'Bangkok', 132, 'London', 'Tokyo']
cptl.sort()     # error because it is not possible to compare int and string
```

- กำหนดว่าได้ว่าให้เรียงจาก มากไปน้อย โดยระบุ reverse=True เช่น

```
score = [2, 4, 8, 1, 6, 5, 7, 9]
score.sort(reverse=True)     # score is changed to [9, 8, 7, 6, 5, 4, 2, 1]

cptl = ['Paris', 'Bangkok', 'Soul', 'London', 'Tokyo']
cptl.sort(reverse=True)
# cptl is changed to ['Tokyo', 'Soul', 'Paris', 'London', 'Bangkok']
```

- ถ้าไม่ระบุพารามิเตอร์ reverse หรือระบุ reverse=False จะเรียงจากน้อยไปมาก เช่น

```
score = [6, 5, 7, 9, 2, 4, 8, 1]
score.sort(reverse=False)    # score is changed to [1, 2, 4, 5, 6, 7, 8, 9]

cptl = ['Paris', 'Bangkok', 'Soul', 'London', 'Tokyo']
cptl.sort(reverse=False)
# cptl is changed to ['Bangkok', 'London', 'Paris', 'Soul', 'Tokyo']
```

- ฟังก์ชัน reverse เรียงค่าในลิสต์ย้อนจากหลังไปหน้า และคืนค่า None เช่น prime.reverse() เช่น

```
prime=[2, 3, 5, 7, 11, 13, 17, 19, 11, 23]
prime.reverse()        # prime is changed to [23, 11, 19, 17, 13, 11, 7, 5, 3, 2]
```

การใช้ FOR LOOP กับลิสต์

เราสามารถดึงค่าทีละค่าจากลิสต์โดยใช้ for loop

PATTERN

```
for var in list:
   # do something with var
```

EXAMPLE

```
score = [4,6,8,2,3,5,8]
total=0
cnt=0
for s in score:
   if s>=5:
      total=total+s
      cnt=cnt+1
print('Average passing score =', total/cnt)
```

ลองเขียนโปรแกรมวนซ้ำเก็บสะสมค่า ตรวจสอบค่า ทดสอบว่ามีค่าที่ตรงตามเงื่อนไขอย่างน้อยหนึ่งค่า / ทุกค่า / ไม่มีแม้แต่ค่าเดียว ในทำนองเดียวกับการใช้ while loop

LIST SLICING

- การตัดบางส่วนของลิสต์ ทำได้ดังนี้

  `listName[start:stop:step]`

  สร้างลิสต์ใหม่จากลิสต์ `listName` โดยเลือกเอาค่าที่ตำแหน่ง `start, start+step, start+2*step,…` ไปเรื่อยๆ และตำแหน่งต้องน้อยกว่า (ไม่เท่ากับ) `stop`

- สามารถระบุตำแหน่ง `start, stop` เป็นจำนวนบวกหรือลบก็ได้ เช่น `days[1:-1]`
- สามารถระบุ `step` เป็นจำนวนบวกหรือลบก็ได้ ถ้า `step` เป็นจำนวนลบจะเรียงค่าจากหลังไปหน้า
- ถ้าไม่ระบุ `step` เช่น `days[2:4]` หมายถึง `step=1`
- ถ้าไม่ระบุ `start` เช่น `days[:6:2]` หมายถึง `start=0`
- ถ้าไม่ระบุ `stop` เช่น `days[2::2]` หมายถึง `stop=len(days)`
- สามารถทำ slicing กับ string ได้เหมือนกับลิสต์

EXAMPLE

```
days=['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
weekdays=days[1:-1]       # ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
Rweekdays=days[-2:0:-1]   # ['Friday', 'Thursday', 'Wednesday', 'Tuesday', 'Monday']
odddays=days[1:6:2]       # ['Monday', 'Wednesday', 'Friday']
weekends=days[0:8:6]      # ['Sunday', 'Saturday']
```

การใช้ไฟล์

เมื่อต้องการอ่านหรือเขียนไฟล์ เราต้องเปิดไฟล์เพื่อให้ได้ file object ที่จะอ้างถึงเมื่อต้องการทำงานกับไฟล์นั้น และต้องปิดไฟล์เมื่อเลิกทำงานกับไฟล์นั้น

การเปิดและปิดไฟล์

ฟังก์ชัน open(*filename*, *mode*) ใช้เปิดไฟล์

- *filename* ต้องระบุชื่อไฟล์ที่บอก folder ที่เก็บไฟล์ด้วย ถ้าไม่บอก folder หมายความว่าไฟล์นั้นอยู่ใน default folder ซึ่งปกติจะเป็น folder ที่เก็บไฟล์โปรแกรม
- ถ้า *mode* = 'r' หมายถึงเปิดไฟล์เพื่ออ่านไฟล์
- ถ้า *mode* = 'w' หมายถึงเปิดไฟล์เพื่อเขียนไฟล์
- ฟังก์ชัน open คืนค่าเป็น file object ที่ใช้อ้างถึงเมื่อต้องการอ่านหรือเขียนไฟล์

ฟังก์ชัน close เพื่อปิดไฟล์

- ต้องอ้างถึง file object ที่ต้องการปิด เช่น `fObj.close()` เมื่อ `fObj` เป็น file object

PATTERN

```
f=open(...,'r')     # open to read
# read file
f.close()
```

```
f=open(...,'w')     # open to write
# write file
f.close()
```

EXAMPLE

```
# open file in drive c, in the folder Users/DELL to read
f=open('c:\\Users\\DELL\\test.txt','r')
# read file
f.close()           # close the file with file object f

# open file in default folder to write
sc=open('data.txt','w')   # if there is the file with this name, it will be deleted.
# write file
sc.close()          # close the file with file object sc
```

WITH OPEN … AS …

with open(*filename*, *mode*) as *fileObj* : ใช้เปิดและปิดไฟล์ โดย

- ส่วนที่อ้างถึง *fileObj* ต้องอยู่ใน block ภายใต้ with open
- ไม่จำเป็นต้องมีคำสั่งปิดไฟล์ เพราะจะปิดไฟล์ให้เองเมื่อออกจาก block

PATTERN

แบบที่ใช้ with open … as	แบบที่ใช้ open/close

`with open(...,'r') as f :`
`  # read file`
`  # must be a block`
`  # no need to close file`
	`f=open(...,'r')`
	`# read file`
	`f.close()`

`with open(...,'w') as f :`
`  # write file`
`  # must be a block`
`  # no need to close file`
	`f=open(...,'w')  # specify file name`
	`# write file`
	`f.close()`

EXAMPLE

```
with open('test.txt','r') as f:  # open file and the variable f stores file object
# read file

with open('data.txt','w') as sc: # open file and the variable sc stores file object
# write file
```

การอ่านไฟล์โดยใช้ FOR LOOP

เราสามารถอ่านข้อความจากไฟล์ทีละบรรทัดโดยใช้ for loop ค่าที่อ่านจากไฟล์มีชนิดเป็นสตริง และมีตัวอักขระขึ้นบรรทัดใหม่ ('\n') ปะท้ายอยู่ด้วย

PATTERN

`f=open(...,'r')`
`for line in f:  # type of line is string`
`  # do something with line`
`f.close()`
	`with open(...,'r')  as f :`
	`  # must indent here * block *`
	`  for line in f: # type of line is string`
	`  # do something with line`

EXAMPLE

`f=open('score.txt','r')`
`for line in f:`
`  print(line)`
`f.close()`
	`with open('score.txt','r') as f:`
	`  for line in f:`
	`  print(line)`

- ถ้าต้องการนำไปคำนวณ ต้องเปลี่ยนเป็น int หรือ float ก่อน

```
f=open('score.txt','r')
total=0
for line in f:
    total=total+int(line)
f.close()
print(total)
```

```
with open('score.txt','r') as f:
   total=0
   for line in f:
      total=total+int(line)
print(total)
```

- ถ้าสตริงในหนึ่งบรรทัดประกอบด้วยค่าหลายค่า ต้องเอาไปแยกเป็นส่วนๆ ด้วยฟังก์ชัน split

```
f=open('score.txt','r')
total=0
for line in f:
    name, score= line.split(',')
    total=total+int(score)
f.close()
print(total)
```

```
with open('score.txt','r') as f:
   total=0
   for line in f:
      name, score= line.split(',')
      total=total+int(line)
print(total)
```

การอ่านไฟล์หนึ่งบรรทัด

ฟังก์ชัน `readline()` เป็นคำสั่งที่อ่านสตริง 1 บรรทัด โดย `f.readline()` อ่านสตริงจากไฟล์ที่อ้างถึงด้วย file object `f` คืนค่าเป็นสตริงที่มีตัวอักขระขึ้นบรรทัดใหม่ ('\n') ปะท้ายอยู่ด้วย

```
with open('score.txt','r') as f:
   line=f.readline()
print(line)
print('-----')
```

ถ้าบรรทัดแรกของไฟล์เป็น

```
123
```

Output ที่พิมพ์บนหน้าจอเป็น

```
Line 1:123
Line 2:
Line 3:-----
```

การอ่านไฟล์แยกเป็นบรรทัดมาเก็บในลิสต์

ฟังก์ชัน `readlines()` เป็นคำสั่งที่อ่านสตริงจากไฟล์ทั้งไฟล์ แยกเป็นบรรทัด โดย `f.readlines()` อ่านสตริงจากไฟล์ที่อ้างถึงด้วย file object `f` คืนค่าเป็นลิสต์ของสตริง ซึ่งแต่ละสตริงมาจาก 1 บรรทัดในไฟล์ และมีตัวอักขระขึ้นบรรทัดใหม่ ('\n') ปะท้ายแต่ละสตริงด้วย

EXAMPLE

ถ้าไฟล์ score.txt เป็น

```
First line
Second line
Third line
```

f=open('score.txt','r') sc=f.readlines() f.close()	with open('score.txt','r') as f: sc=f.readlines()

โปรแกรมข้างบนนี้ให้ sc เป็น ['First line\n','Second line\n', 'Third line']

'Third line' ไม่มี \n เพราะไม่มีการเคาะขึ้นบรรทัดใหม่ท้ายบบรรทัดสุดท้าย

การเขียนไฟล์

f.write(data) เป็นคำสั่งที่เขียน data ต่อท้ายไฟล์ที่อ้างถึงด้วย file object f โดยที่ data ต้องเป็นสตริง

PATTERN

f=open(...,'w')
…
f.write(data)
…
f.close()
	with open(...,'w') as f:
	  …
	  f.write(data)
	  …

ถ้าต้องการให้มีการขึ้นบรรทัดใหม่หลังข้อความที่เขียนในไฟล์ ต้องสร้างสตริงที่ต่อท้ายด้วย '\n' แล้วจึงเขียนลงไฟล์

f=open('score.txt','w')
for sc in [9,5,6.5,9,4,8.5]:
  f.write(str(sc)+'\n')
f.close()
	with open('score.txt','w') as f:
	  for sc in [9,5,6.5,9,4,8.5]:
	  f.write(str(sc)+'\n')

EXAMPLE

```
#copy a file
with open('test.txt','r') as f1 , open('score.txt','w') as f2 :
    for s in f1:
        f2.write(s)

# another way to copy a file
f1=open('test.txt','r')
f2=open('score.txt','w')
   for s in f1:
      f2.write(s)
f1.close()
f2.close()
```

การเขียนลิสต์ของสตริงลงในไฟล์

ฟังก์ชัน `writelines()` เป็นคำสั่งที่เขียนสตริงที่อยู่ในลิสต์ลงในไฟล์ `f.writelines(manyLines)` เขียนสตริงที่อยู่ในลิสต์ `manyLines` ลงในไฟล์ที่อ้างถึงด้วย file object `f`

จากโปรแกรมข้างล่างนี้

```
with open('score.txt','w') as f:
   f.writelines(['first line\n', '\n\n','fourth line\n',)
```

จะได้ไฟล์ `score.txt` เป็น

```
first line


fourth line
```

LIST COMPREHENSION

List comprehension เป็นการสร้างลิสต์โดยเลือกค่าจาก iterator มาสร้างเป็นลิสต์ ดังนี้

```
[exp(i) for i in iterator if cond(i)]
```

เป็นการสร้างลิสต์ โดยเลือกสมาชิก `i` ใน `iterator` ที่ทำให้เงื่อนไข `cond(i)` เป็นจริง แล้วเอา `i` ไป คำนวณหา `exp(i)` แล้วเอาใส่ในลิสต์

EXAMPLE

```
score = [9, 4.5, 8.5, 10, 5, 7.5, 3, 9, 0]
pass = [i for i in sc if sc>=5]              # get [9, 8.5, 10, 5, 7.5, 9]
midPercent = [i*10 for i in sc if 4<sc<8]    # get [45.0, 50, 75.0]

evenStudent = [score[i] for i in range(0, len(score),2)]   # use range as iterator
                                                           # get [9, 8.5, 5, 3, 0]
```

EXAMPLE

```
f=open('temp.txt','r')
hiTempList=[float(line) for line in f if float(line)>36]
# get each line in temp.txt and convert to float
# if it is higher than 36 add in the list
f.close()
```

DICTIONARIES

ค่าคงที่ชนิด DICT

- Dict เป็น iterator ชนิดหนึ่ง
- ใช้ { } ครอบสมาชิกใน dict โดยที่สมาชิกเป็นคู่ของคีย์กับค่า (key-value pair) ที่เขียนในรูปแบบ

  key : value

  โดยแต่ละคู่ของคีย์กับค่า คั่นด้วยเครื่องหมาย , เช่น

  ```
  {'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':30, 'Dentistry':32}
  ```

- { } หมายถึง dict ว่าง (empty dict)
- คีย์ใน dict อาจเป็น int, float, string, tuple แต่คีย์เป็น list หรือ dict ไม่ได้
- ค่าใน dict อาจเป็น int, float, string, tuple, list หรือ dict ก็ได้
- คีย์ใน dict หนึ่งต้องไม่มีค่าซ้ำกัน
- เราไม่สามารถกำหนดลำดับของคู่ของคีย์กับค่าใน dict เองได้ เพราะตัวแปลภาษาจะกำหนดลำดับให้ตามวิธีที่กำหนด

การอ้างถึงค่าใน DICT

- เรา**ไม่**สามารถอ้างถึงคู่ของคีย์และค่าใน dict ด้วยตำแหน่ง
- เราสามารถอ้างถึงค่าใน dict ด้วยคีย์ เช่น

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
print(code['Arts'])          # print 22
```

- ถ้าอ้างถึงค่าใน dict ด้วยคีย์ที่ไม่อยู่ใน dict จะเกิด error เช่น

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
print(code['Fishery'])       # error because 'Fishery' is not a key in code
```

- เราสามารถ**เพิ่มคู่ของคีย์และค่าใน dict** ได้ดังนี้

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
code['Veterinary']=31        # Because 'Veterinary' is not a key in code,
                             # 'Veterinary':31 is added in code.
```

- ถ้าเพิ่มคู่ของคีย์และค่าใน dict เมื่อมีคีย์นั้นอยู่ใน dict แล้ว ค่าใหม่จะไปแทนค่าเก่า นั่นคือเรา**แก้ค่าที่คู่กับคีย์**นั้นใน dict เช่น

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
code['Medicine']=30          # Because 'Medicine' is a key in code, the value
                             # which is paired with 'Medicine' is changed to 30.
```

- ถ้าใช้การกำหนดค่าให้ตัวแปร (assignment) เอาตัวแปรที่เก็บ dict ไปเก็บในอีกตัวแปรหนึ่ง เช่น

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
dupCode=code     # dupCode and code share the same dict
```

ตัวแปรทั้ง 2 ตัวจะอ้างถึง dict เดียวกัน (ไม่มีการสร้าง dict ใหม่) ดังนั้น การเปลี่ยนค่าใน dict จะส่งผลกับทั้งสองตัวแปร เช่น

```
code['Medicine']=30                # change both code and dupCode
print(dupCode['Medicine'])         # also get 30
```

- ถ้าต้องการให้สร้าง dict ใหม่ก่อนที่จะเอาไปเก็บในอีกตัวแปร เราสามารถใช้ฟังก์ชัน dict เพื่อเปลี่ยน dict เป็น dict ซึ่งจะทำให้มีการสร้าง dict ใหม่ขึ้นมา ก่อนจะเอาไปเก็บในตัวแปรใหม่ เช่น

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
dupCode=dict(code)     # dupCode and code do not share the same dict

code['Medicine']=30    # change only code, but does not change dupCode
print(code['Medicine'])      # get 30
print(dupCode['Medicine'])   # get 20
```

ฟังก์ชันสำหรับ DICT

หาความยาวของ DICT

- ฟังก์ชัน len ใช้หาจำนวนสมาชิกใน dict

EXAMPLE:

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
print(len(code))          # get 5
```

หาคีย์ / ค่า / คู่ของคีย์และค่า ใน DICT

- ฟังก์ชัน keys ใช้หาคีย์ทั้งหมดของ dict แล้วคืนค่ามาเป็นลิสต์ของคีย์
- ฟังก์ชัน values ใช้หาค่าทั้งหมดของ dict แล้วคืนค่ามาเป็นลิสต์ของค่า
- ฟังก์ชัน items ใช้หาคู่ของคีย์และค่าทั้งหมดของ dict แล้วคืนค่ามาเป็นลิสต์ของคู่ของคีย์และค่า
- ฟังก์ชัน has_keys ใช้ตรวจสอบว่าค่านั้นเป็นหนึ่งในคีย์ของ dict หรือไม่ แล้วคืนค่ามาเป็น True/False

โดยมีรูปแบบการเรียกใช้ดังนี้

EXAMPLE:

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}

print(code.keys())
# get dict_keys(['Science', 'Engineering', 'Arts', 'Medicine', 'Dentistry'])

print(code.values())
# get dict_values([23, 21, 22, 20, 32])

print(code.items())
# get dict_items([('Science',23),('Engineering',21),('Arts',22),…('Dentistry',32)])

print(code.has_keys('Cooking'))        # get False
print(code.has_keys('Science'))        # get True
```

อาจตรวจสอบว่ามีค่าที่เราสนใจเป็นคีย์ในdict นั้นหรือไม่ ได้หลายวิธีดังตัวอย่างต่อไปนี้

EXAMPLE:

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}

if code.has_keys('Science'):
   print('Scince is in dict')
if 'Science' in code.keys():
   print('Scince is in dict')
if 'Science' in code:
   print('Scince is in dict')
```

เพิ่ม DICT หนึ่งในอีก DICT

- ฟังก์ชัน updates เอาคู่ของคีย์และค่าจาก dict หนึ่งไปใส่เพิ่มในอีก dict แล้วคืนค่า None
- ถ้าคีย์ซ้ำกัน ค่าเก่าจะถูกแทนด้วยค่าใหม่
- ถ้าคีย์ไม่ซ้ำกัน คู่ของคีย์และค่าใหม่จะถูกเพิ่มเข้าไปเลย

EXAMPLE:

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
code.update({'Medicine':30, 'Veterinary':31})
# code is changed to {'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':30,
'Dentistry':32, 'Veterinary':31}
```

การลบค่าใน DICT

- ฟังก์ชัน del ลบค่าใน dict โดยระบุว่าชื่อของ dict และ key ที่ต้องการลบ เช่น `del(code['Arts'])`

EXAMPLE

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
del(code['Arts'])
# code is changed to {'Science':23, 'Engineering':21, 'Medicine':30, 'Dentistry':32}
```

FOR IN DICT

เนื่องจาก dict เป็น iterator ชนิดหนึ่ง เราสามารถใช้ for เพื่อเข้าถึงค่าใน dict ได้

EXAMPLE

```
code={'Science':23, 'Engineering':21, 'Arts':22, 'Medicine':20, 'Dentistry':32}
facStr=''
for fac in code:
   facStr=facStr+fac+' '
print(facStr)              # get 'Science Engineering Medicine Dentistry '

for fac in code:
   print(code[fac])        # get each faculty code e.g. 23, 21, 22, …
```

การเก็บตารางไว้ใน DICT เพื่ออ่านค่ามาใช้

เราสามารถเก็บตารางของค่า เช่น ตารางธาตุ ตารางราคาสินค้า เพื่ออ่านค่าที่คู่อยู่กับคีย์ เช่น ตารางธาตุ มีชื่อของธาตุเป็นคีย์ที่คู่กับเลขอะตอม ตารางราคาสินค้า มีรหัสสินค้าเป็นคีย์ที่คู่กับราคาสินค้า

PATTERN

```
tab = {...}                     # keep a table in a dict
tabKey = ...                    # tabKey is the key of what we need
if tabKey in tab:               # check if tabKey is a key in tab
       tabVal = tab[tabKey]     # read the value
       ...                      # do something with the value
else:
       ...                      # do something else when the key is not in tab
```

EXAMPLE: CHECK THE FACULTY FROM STUDENT ID

```
code={23:'Science', 21:'Engineering', 22:'Arts', 20:'Medicine', 32:'Dentistry'}
sid = input('Enter student ID: ')      # get student ID
fac = sid[-2:]                         # get faculty code from the last 2 digits
if fac in code                         # check if the faculty code is a key in code
   print(sid, 'is in',code[fac])       # get faculty name from code
else:                                  # fac is not a key in code
   print('No information for ', sid)
```

การเพิ่มและแก้ค่าใน DICT

เราสามารถเพิ่มคู่ของคีย์และค่า หรือแก้ค่าที่คู่กับคีย์ที่มีอยู่ใน dict แล้วได้

PATTERN

```
tab = {...}                       # keep a table in a dict
tabKey = ...                      # tabKey is the key of what we need
if tabKey in tab:                 # check if tabKey is a key in tab
       tabVal = tab[tabKey]       # read the value
       ...                        # do something with the value
else:
       ...                        # do something else when the key is not in tab
```

EXAMPLE: ADD OR CHANGE THE NUMBER OF CREDITS FOR THE COURSE

```
courseCredit = { 2301170:3, 2310172:1, 2301117:4, 2301118:4 }
if input('Want to add or change? (Y/N)') == 'Y':
   crs, crdt = input('Enter course ID and credit:').split()
   crs = int(crs)
   crdt = int(crdt)
   if crs in courseCredit:              # check if crs is already in the dict
      if courseCredit[crs]==crdt:       # check if the credit in dict = new value
         print('Already in the dictionary')
      else:
         print('Change the number of credit')
         courseCredit[crs] = crdt
   else:
      print('Store new course information')
      courseCredit[crs] = crdt
else:
   print('Bye')
```

EXAMPLE: CREATE A CALENDAR AS DICT

```
dayList  = ['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday']
dayOrder = {'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,
'Saturday':6}
daysInMonth = {'January':31, 'February':28, 'March':31}
calendar = {'January':{1: 'Wednesday'}, 'February':{1: 'Saturday'}, 'March':{1:
'Sunday'}}
for m in calendar:                      # create the calendar for each month
   mcal = calendar[m]                   # get calendar with day 1 only
   fstDay = mcal[1]
   dayPos = dayOrder[fstDay]
   for day in range(2,daysInMonth[m]+1):
      dayPos = (dayPos+1)%7
      mcal[day] = dayList[dayPos]
```

DICT COMPREHENSION

Dict comprehension เป็นการสร้าง dict โดยเลือกค่าจาก iterator มาสร้างเป็น dict (คล้ายกับ list comprehension) ดังนี้

```
[exp1(i):exp2(i) for i in iterator if cond(i)]
```

เป็นการสร้าง dict โดยเลือกสมาชิก `i` ใน `iterator` ที่ทำให้เงื่อนไข `cond(i)` เป็นจริง แล้วเอา i ไปคำนวณหา `exp1(i)` และ `exp2(i)` แล้วสร้างเป็นคู่คีย์กับค่าและใส่ใน dict

EXAMPLE

```
score = [['tim',9], ['tom',4.5], ['tum',8.5], ['tam',10]]
dSc1 = [ns[0]:ns[1] for ns in score if sc>=5]    # get {'tim':9, 'tum':8.5, 'tam':10}
dSc2 = [name:sc for name,sc in score if sc>=5]   # same as above

dayList  = ['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday']
# create dict {'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,...,'Saturday':6}
dayOrder = {dayList[i]:i for i in range(len(dayList))}

sq = [i:i*i for i in range(100)]       # use range as iterator
```

COMMON USES FOR SOME ITERABLE OBJECTS

Iterable objects เป็นชุดของค่า ได้แก่ strings range list file dict ภาษาไพธอนมีโครงสร้างที่ใช้ได้กับ iterable objects หลายชนิด เช่น for loop, ฟังก์ชัน in, not in

FOR LOOP

PATTERN

```
for var in iterable:
   # do something with var
```

ตัวแปรของ for loop ได้ค่าจาก iterable object ทีละค่า

- เมื่อ iterable object เป็น string ตัวแปรจะได้ค่าเป็นอักขระ (character) แต่ละตัวในสตริง
- เมื่อ iterable object เป็น range ตัวแปรจะได้ค่าเป็นจำนวนเต็มแต่ละค่าใน range
- เมื่อ iterable object เป็น list ตัวแปรจะได้ค่าเป็นสมาชิกแต่ละตัวในลิสต์
- เมื่อ iterable object เป็น file ตัวแปรจะได้สตริงแต่ละบรรทัดในไฟล์
- เมื่อ iterable object เป็น dict ตัวแปรจะได้คีย์แต่ละค่าใน dict

EXAMPLE

```
# for in string
cnt=0
for ch in 'this is a test ':     # ch is a character from 'this is a test '
   if ch==' ':
      cnt = cnt+1
print('There are', cnt,'blanks.')

# for in range
total=0
for s in range(10,101):   # s is a integer from 10 to 100
   total=total+s
print('sum of 10 to 100 =', total)

# for in list
score = [4,6,8,2,3,5,8]
total=0
for s in score:           # s is an element in the list score
   total=total+s
print('Total score =', total)
```

```
# for in file
with open('score.txt') as f:
   total=0
   for s in f:            # s is a string with \n at the end
      total=total+int(s)
print('Total score =', total)

# for in dict
course = {2301170:3, 2301172:1, 2301117:4}
total=0
for s in course:    # s is a key in the dict course
   total=total+course[s]
print('Total credit =', total)
```

IN, NOT IN

PATTERN

```
if exp in iterable:
   # do something

if exp not in iterable:
   # do something
```

in, not in ใช้ตรวจสอบว่า ค่าที่ได้จาก *exp* เป็นค่าหนึ่งใน *iterable*

- เมื่อ iterable object เป็น string จะตรวจสอบว่าค่าที่ได้จาก *exp* เป็นสตริงย่อยในสตริงนั้น
- เมื่อ iterable object เป็น range จะตรวจสอบว่าค่าที่ได้จาก *exp* เป็นค่าใน range
- เมื่อ iterable object เป็น list จะตรวจสอบว่าค่าที่ได้จาก *exp* เป็นเป็นสมาชิกตัวหนึ่งในลิสต์
- เมื่อ iterable object เป็น dict ตัวแปรจะได้คีย์แต่ละค่าใน dict

EXAMPLE

```
# in string
sub = ' is'
if sub in 'this is a test ':
   print(sub,'is in this is a test ')

# in range
s=int(input('Number to check:'))
if s in range(10,101):
   print(s,'is between 9 to 101')
```

2301172

```
# in list
course = [2301170,2301172,2301117]
crs = int(input('course:'))
if crs in course:
   print(crs,'in the list')

# in dict
courseCrdt = {2301170:3, 2301172:1, 2301117:4}
crs = int(input('course:'))
if crs in courseCrdt:
   print(crs,'in the dict')
```