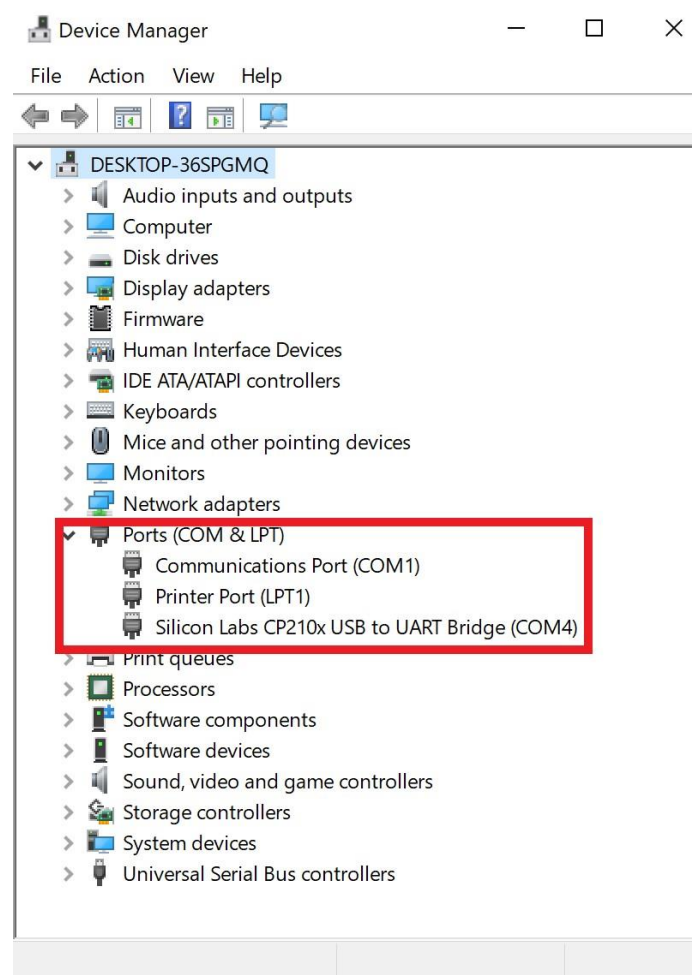
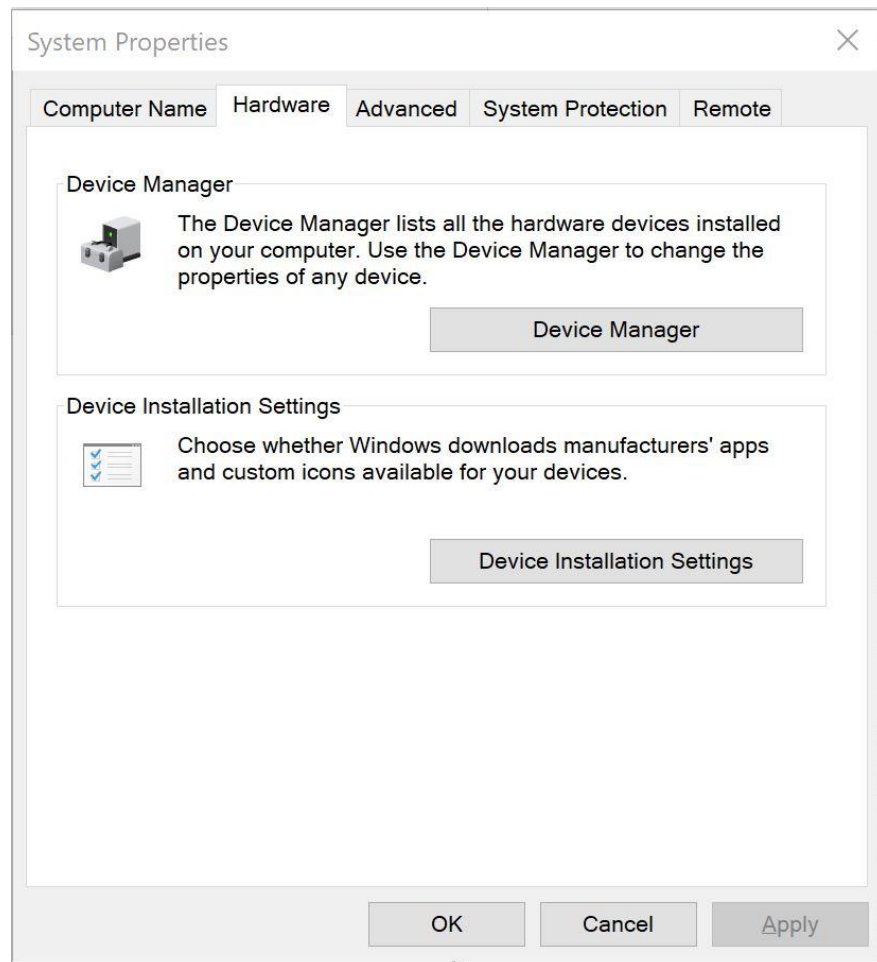


ไพธอน



ตรวจสอบหมายเลขคอมพอร์ต (com port)

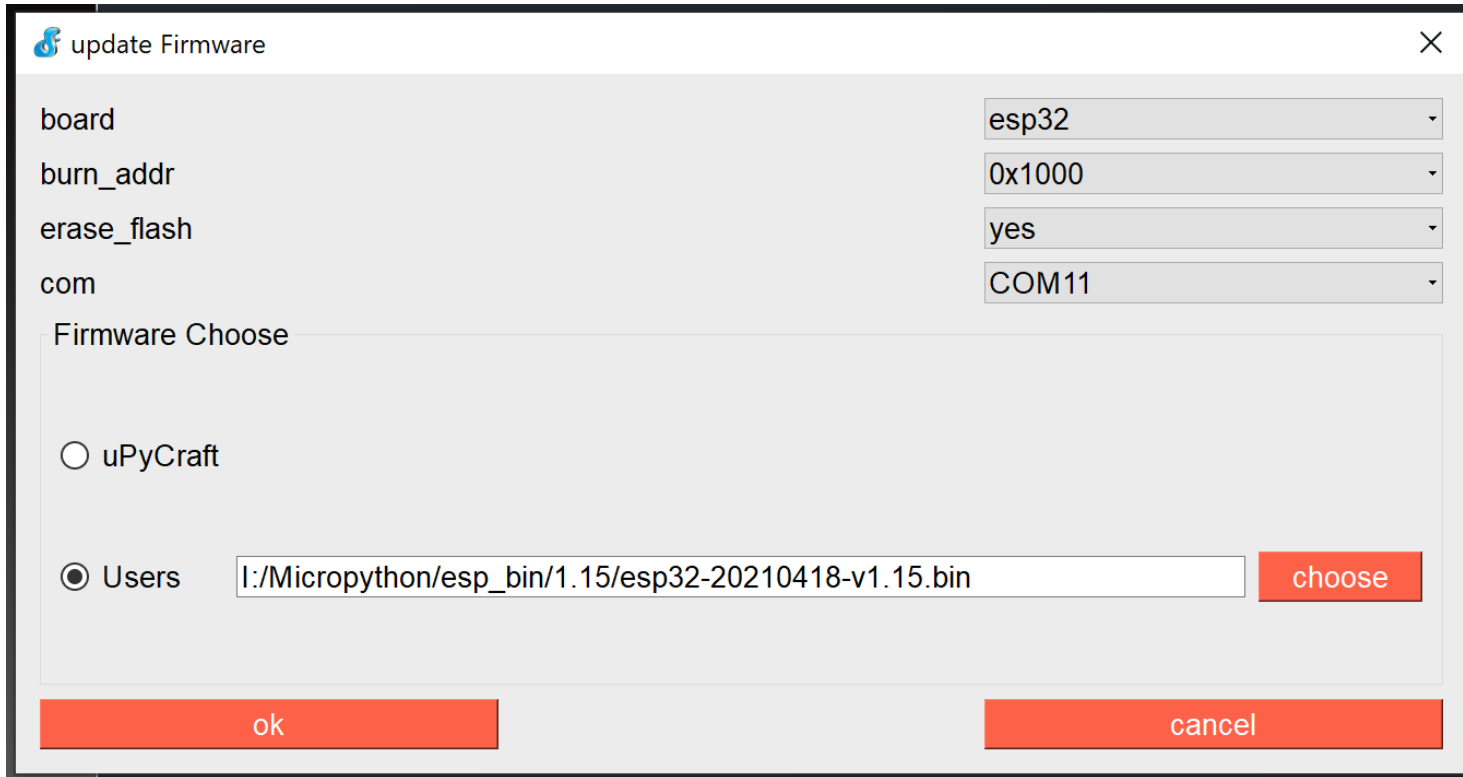


หากไม่พบ หมายเลข com port
ให้ติดตั้งไดรเวอร์ก่อน

- โปรแกรม uPyCraft IDE สำหรับ Windows



การติดตั้ง Micropython ผ่าน editor



update Firmware

board: esp32

burn_addr: 0x1000

erase_flash: yes

com: COM11

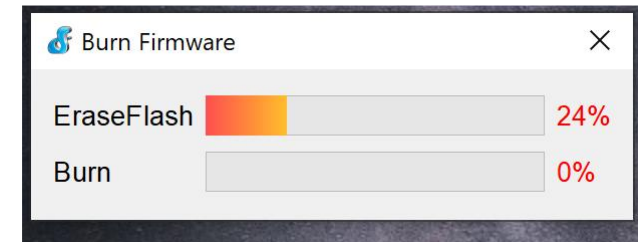
Firmware Choose

☐ uPyCraft

☒ Users: I:/Micropython/esp_bin/1.15/esp32-20210418-v1.15.bin

choose

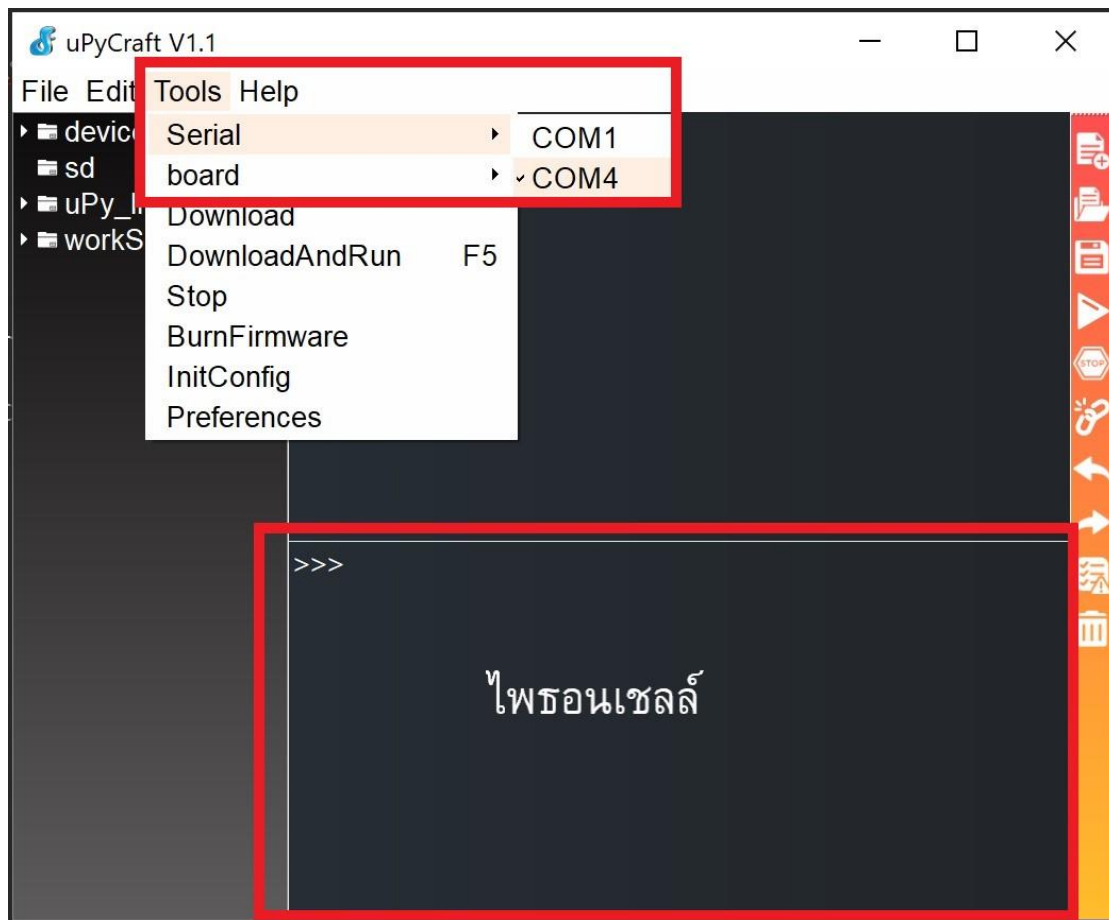
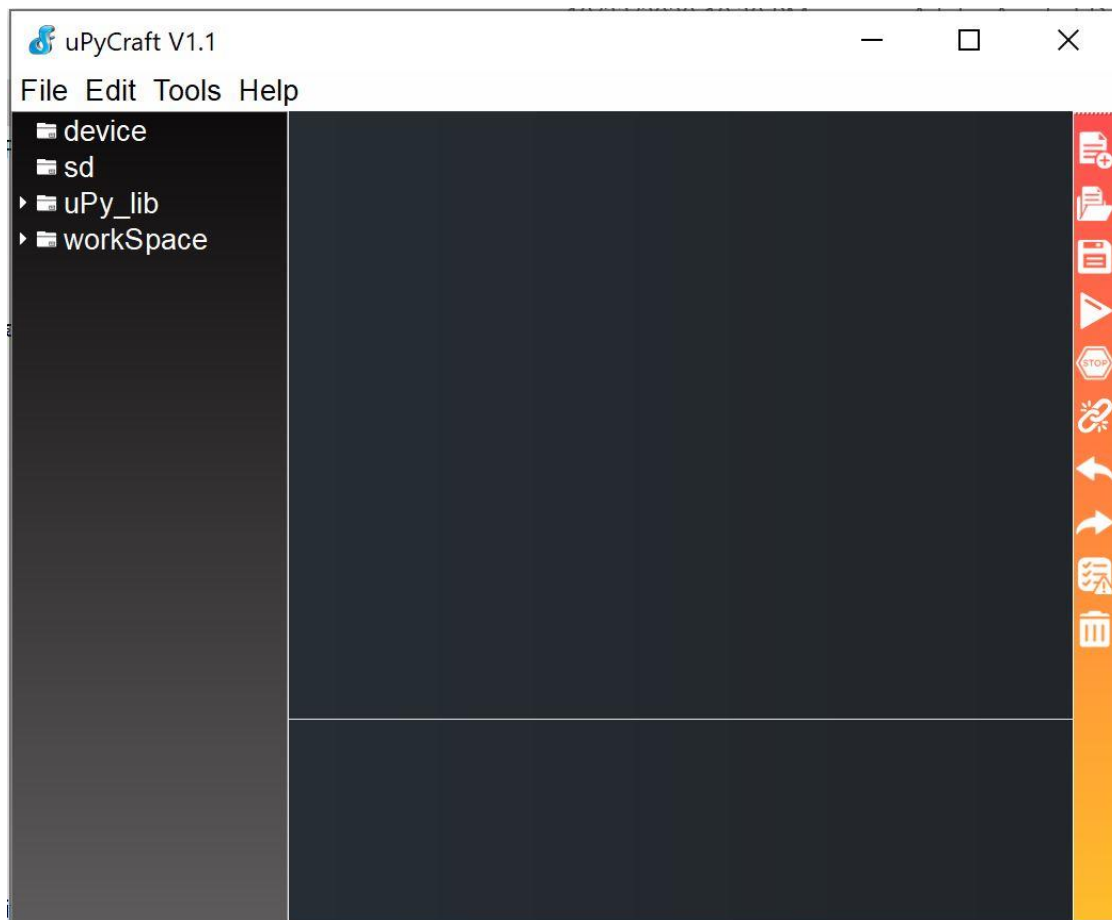
ok cancel



Burn Firmware

EraseFlash: 24%

Burn: 0%



เริ่มต้นใช้งาน

คุณสมบัติเด่น

- การแปลภาษาของภาษาไพธอนเป็นแบบอินเทอร์พรีเตอร์ประมวลผลไปทีละบรรทัด
- ไวยากรณ์อ่านง่าย โดยใช้การย่อหน้าแทน ทำให้สามารถอ่านโปรแกรมที่เขียนได้ง่าย
- ภาษาไพธอน และชุดของไลบรารีสนับสนุนการประมวลผลทางด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ได้อย่างมีประสิทธิภาพ
- ไพธอนมีไลบรารีที่สนับสนุนงานด้านการสร้างภาพกราฟฟิก และการประมวลผลภาพ (Image processing) มากมาย
- ไพธอนเตรียมไลบรารีสำหรับสนับสนุนการเขียนโปรแกรมทางด้านปัญญาประดิษฐ์
- ไพธอนมีความสามารถในการจัดการหน่วยความจำอัตโนมัติ (Garbage collection)

ภาษาไพธอน และ ภาษาซี

```
1 | print("Hello, World!")
```

ภาษาไพธอน

```
1 | #include <stdio.h>
2 | int main()
3 | {
4 |     // printf() displays the string inside quotation
5 |     printf("Hello, World!");
6 |     return 0;
7 | }
```

ภาษาซี

ไวยากรณ์สำคัญ

- การตั้งชื่อตัวแปร ตัวใหญ่และตัวเล็กถือว่าเป็นคนละตัวแปร เช่น abc, aBC และ ABC เป็นต้น
- ไพธอนใช้การเยื้องย่อหน้า การใช้ช่องว่าง (space) และ แท็บ tabs ไม่เหมือนกัน อย่าผสมเลือกอย่างใดอย่างหนึ่ง
- ไพธอนใช้เครื่องหมาย Colon : แสดงขอบเขตของข้อมูล
- เมื่อจำเป็นต้องเขียนคำสั่ง ที่มีความยาวมากๆ ไม่หมดใน 1 บรรทัด ให้ใช้เครื่องหมาย \ ตามด้วย enter
- ไพธอนใช้เครื่องหมาย ' (single quote), " (double quote) ในการแสดงค่าของสตริง แต่เครื่องหมาย """ (triple quote) สามารถใช้เชื่อมต่อสตริงแบบหลาย ๆ บรรทัดได้

ไวยากรณ์สำคัญ

- การตั้งชื่อตัวแปรภาษาไพธอน ตัวใหญ่และตัวเล็กถือว่าเป็นคนละตัวแปร เช่น abc, aBC และ ABC เป็นต้น การตั้งชื่อตัวแปรขอให้เป็นชื่อที่เข้าใจได้เมื่อกลับมาอ่านอีกครั้ง และห้ามใช้คีย์เวิร์ด (keyword) เป็นตัวแปร

FALSE	Class	Finally	Is	return
None	Continue	For	Lambda	try
TRUE	Def	From	nonlocal	while
And	Del	Global	Not	with
As	Elif	If	Or	yield
Assert	Else	Import	Pass	
Break	Except	In	Raise	

- ในภาษาซีมีการใช้เครื่องหมายปีกกาเปิดและปิด ({, }) เพื่อแสดงถึงขอบเขตของฟังก์ชันหรือโปรแกรม ดังนี้

```
float compute()  
{  
    float x, y;  
    return x*y;  
}
```

- ไพธอนใช้เครื่องหมาย colon (:)

```
def on_relay(relay):  
    relay.value(0)  
def off_relay(relay):  
    relay.value(1)
```

- ไพธอนใช้การเยื้องย่อหน้า เพื่อแสดงขอบเขต ต้องระวังการเยื้องหน้าให้ดีอาจใช้ช่องว่าง (space) หรือ แท็บ (tabs) อย่างไม่อย่างหนึ่งเพียงอย่างเดียว

การทำงานพื้นฐาน

ประเภท	โอเปอเรเตอร์	การใช้งาน	ตัวอย่าง
คณิตศาสตร์	+	บวก	$x+10$
	-	ลบ	$x-10$
	*	คูณ	$x*10$
	/	หาร	$x/10$
	%	เศษ	$x\%1$
เปรียบเทียบ	==	เท่ากับ	$A == B$
	!=	ไม่เท่ากับ	$A != B$
	>	มากกว่า	$A > B$
	<	น้อยกว่า	$A < B$
	>=	มากกว่าหรือเท่ากับ	$A >= B$
	<=	น้อยกว่าหรือเท่ากับ	$A <= B$
ตรรกศาสตร์	and	และ	$A \text{ and } B$
	or	หรือ	$A \text{ or } B$

ชนิดข้อมูล (Data types)

- **Integer** เลขจำนวนเต็ม (Integers)

```
>>> 1 + 2 + 3
```

```
6
```

```
>>> 1 + 100 + 25 + 3
```

```
129
```

```
>>>
```

- **ประเภท Float** ตัวเลขทศนิยม หรือจำนวนจริง (Floating-Point numbers)

```
>>> 3.6 + 5.0 + 25/4
```

```
14.85
```

- ประเภท Boolean

```
>>> x = 10
>>> y = 20
>>> x == y
False
>>> Z = True
>>> print(Z)
True
```

- ประเภท String ข้อมูลชนิดตัวอักษร

```
>>> myname = "CK"
>>> print(myname)
CK
```

- ประเภท **list** เก็บข้อมูลได้หลายจำนวนภายในตัวแปรเดียว

```
>>> member = ["chatchai", "wipada", "wisarut"]
>>> print(member)
[chatchai, wipada, wisarut]
>>> print(member[0])
chatchai
```

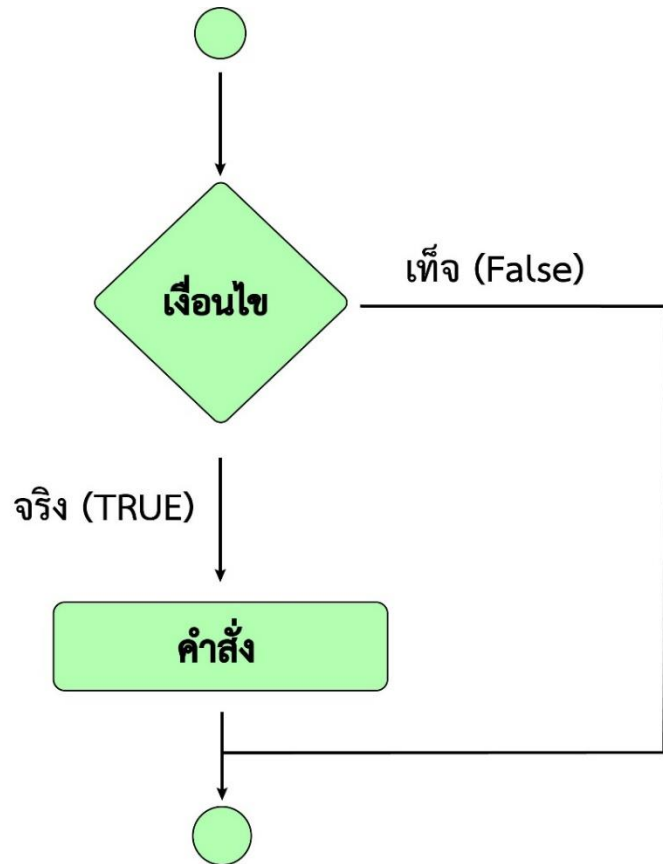
- ประเภท **Tuple** มีลักษณะคล้ายกับลิสต์สำหรับเก็บข้อมูลที่มีค่าคงที่และไม่สามารถแก้ไขค่าได้

```
>>> nick = ("ck", "june", "opp")
>>> print(nick)
(ck, june, opp)
>>> print(nick[1])
june
```

- **ประเภท Dictionary** เก็บข้อมูลเป็นคู่ ระหว่าง คีย์(Key) กับข้อมูล (Value) โดยที่การกำหนดค่าคีย์ต้องไม่ซ้ำกัน

```
>>> nickname = {"chatchai": "ck", "wipada" : "june", "wisarut" : "opp"}
>>> print(nickname)
{wipada: june, wisarut: opp, chatchai: ck}
>>> print(nickname["wipada"])
june
```

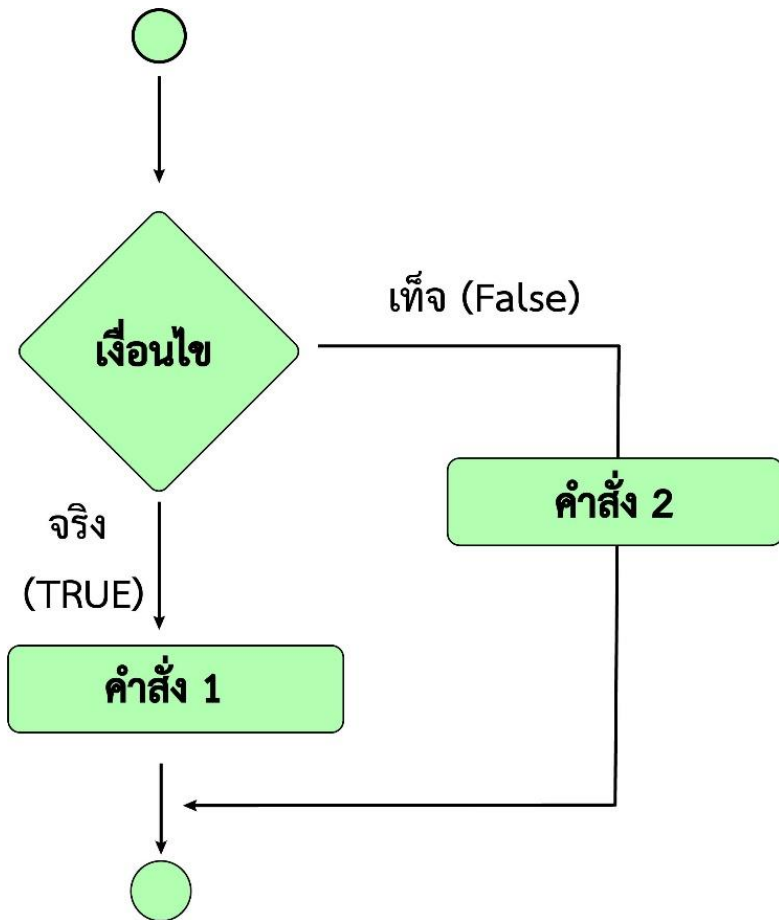
การทำงานแบบเงื่อนไข `if`, `if/else`, `if/elif/else`



```
if a == 0:  
    print ("zero!")
```


การทำงานแบบเงื่อนไข

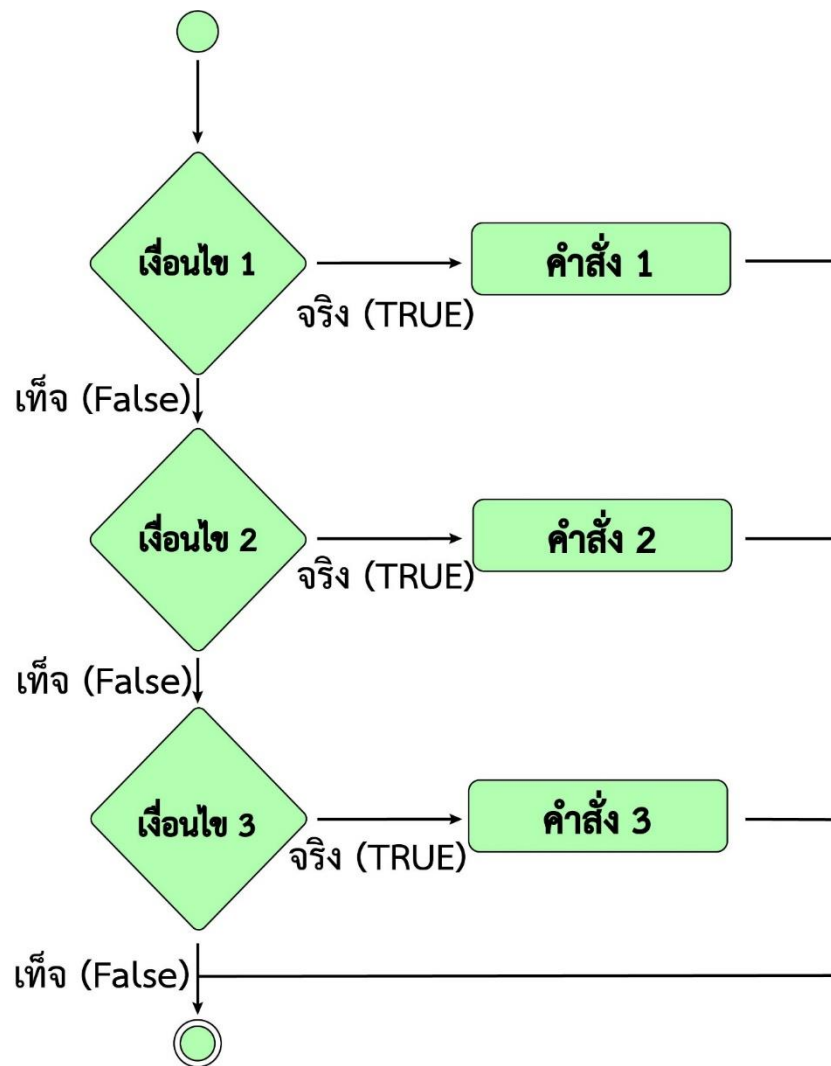
`if/else`, `if/elif/else`



```
if a == 0:  
    print ("zero!")  
else:  
    print ("non zero")
```

การทำงานแบบเงื่อนไข

`if/else`, `if/elif/else`



```
if a == 0:
    print "zero!"
elif a < 0:
    print "negative!"
else:
    print "positive!"
```

การทำงานซ้ำหลายครั้ง

while loops

do something

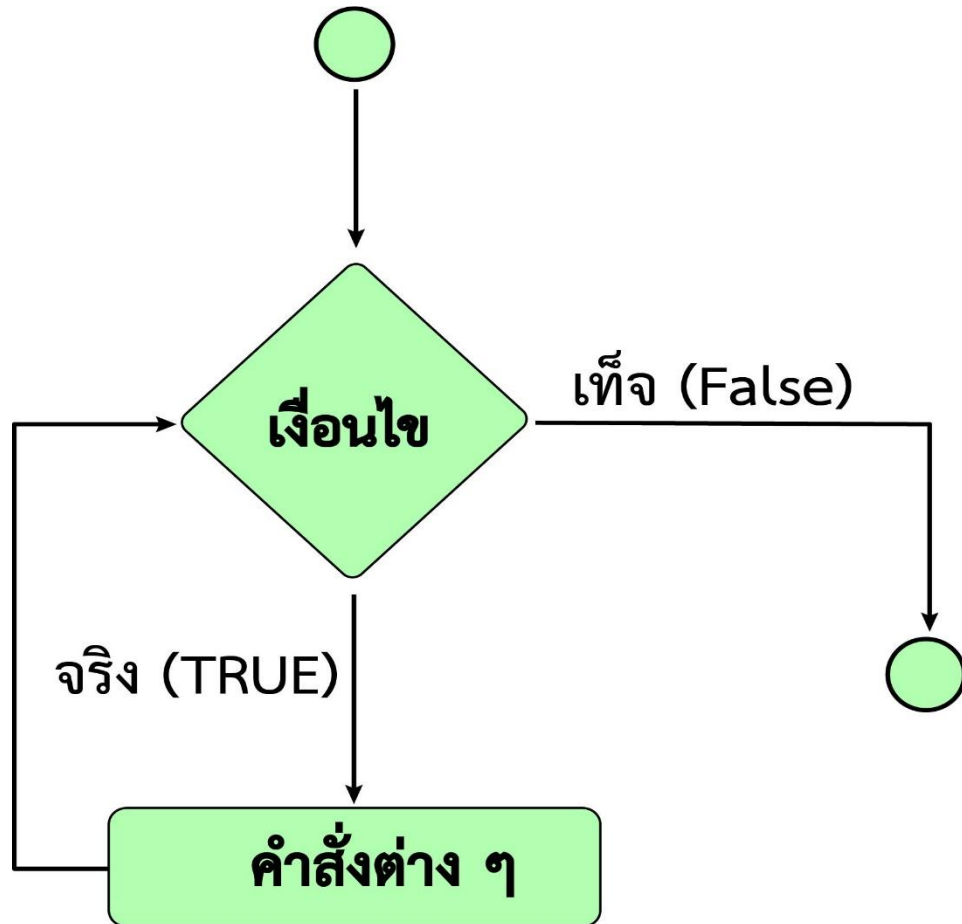
```
a = 10
```

```
while a > 0:
```

```
    print a
```

```
    a = a - 1
```

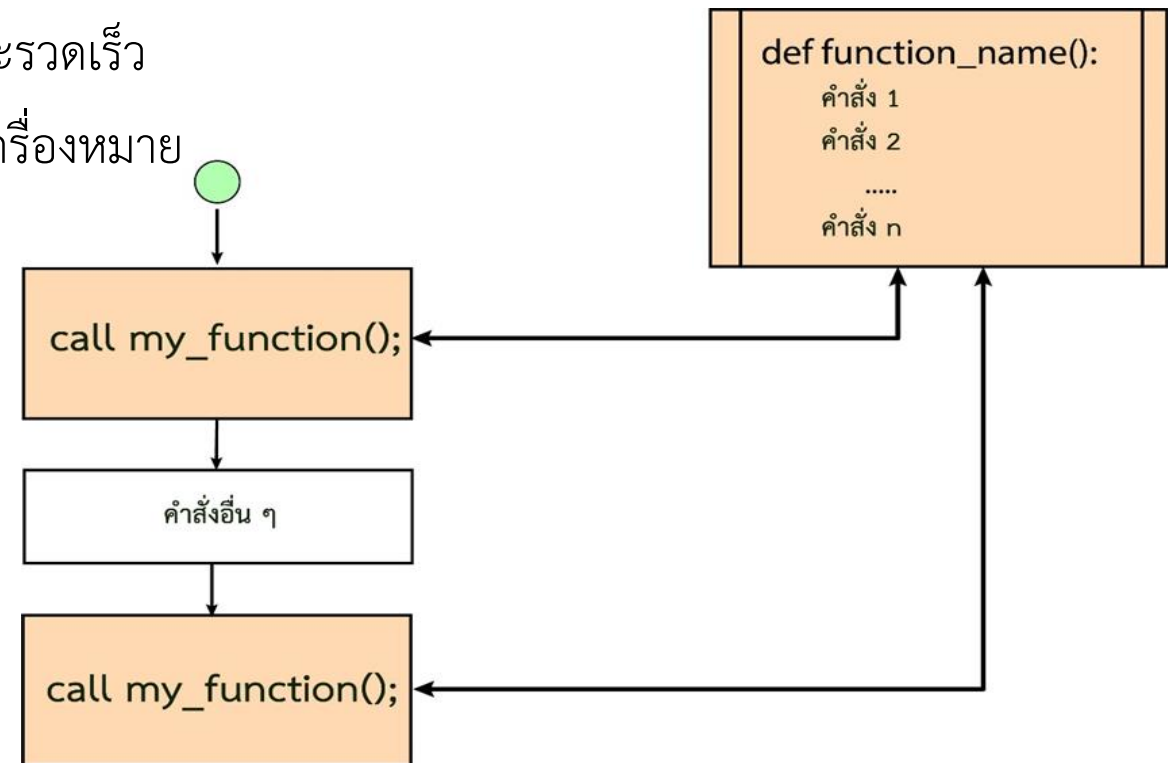
ค่าที่ได้ ??



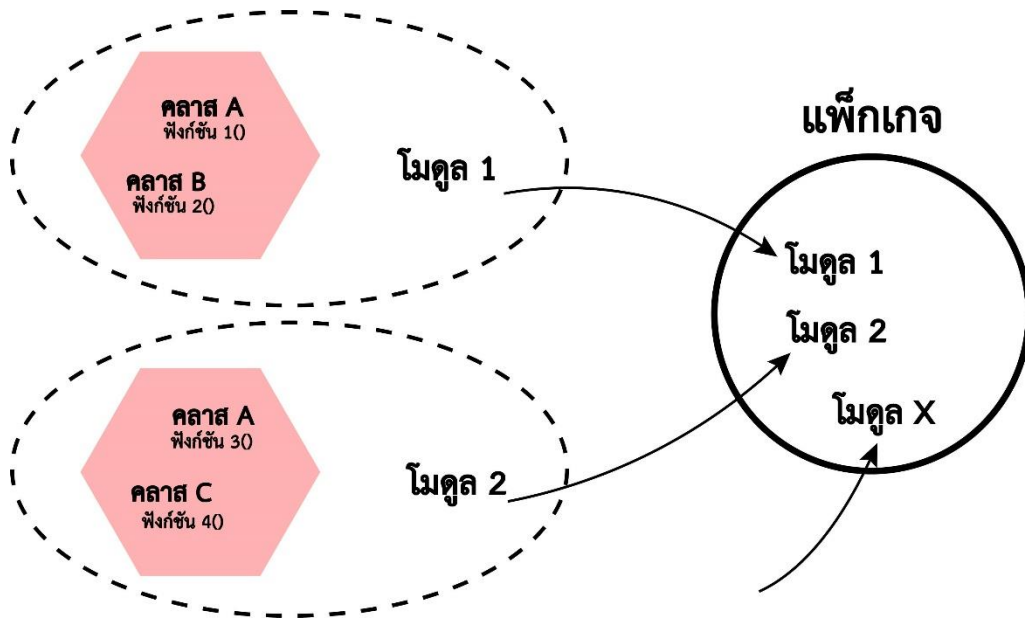
ฟังก์ชัน

- โปรแกรมย่อยหรืองานย่อยๆ (Sub-program) ภายในโปรแกรมขนาดใหญ่หรือ
- ประโยชน์ของฟังก์ชัน
 1. ช่วยลดคำสั่ง ที่ซ้ำซ้อนกันในโปรแกรม
 2. สามารถปรับปรุงและแก้ไขโปรแกรมได้อย่างรวดเร็ว
 3. ช่วยทำให้โปรแกรมมีความกะทัดรัด ทำให้เข้าใจง่ายและรวดเร็ว
- การประกาศฟังก์ชันใช้คำว่า def นำหน้าตามด้วยชื่อฟังก์ชันและเครื่องหมายวงเล็บ ():

```
def foo(x) :  
    y = 10 * x + 2  
    return y
```



โมดูล



- การใช้คำสั่ง `import` เรียกใช้งานฟังก์ชันและตัวแปรทั้งหมดในโมดูลเข้ามาทำงานในโปรแกรม สามารถเรียกใช้มากกว่าหนึ่งโมดูลพร้อมกัน

```
import math [, module2, module3 ..., moduleN]
```

```
print math.sqrt(2.0)
```

- การใช้คำสั่ง `from module import function` เรียกฟังก์ชันหรือตัวแปรเฉพาะที่ต้องการมาใช้งานเท่านั้น

```
from math import sqrt
```

```
print sqrt(2.0)
```

- การใช้คำสั่ง `from module import *` เป็นการเรียกฟังก์ชันทั้งหมดของโมดูลนั้น

```
from math import *
```

```
print sqrt(2.0)
```

- การใช้คำสั่ง `import moduleName as newName` เป็นการเรียกใช้โมดูลชื่อเดิมไปเป็นชื่อโมดูล

```
import pandas as pd
```

```
data_in = pd.read_csv('temperature.csv')
```

การแสดงผล

- การใช้งานรูปแบบ %

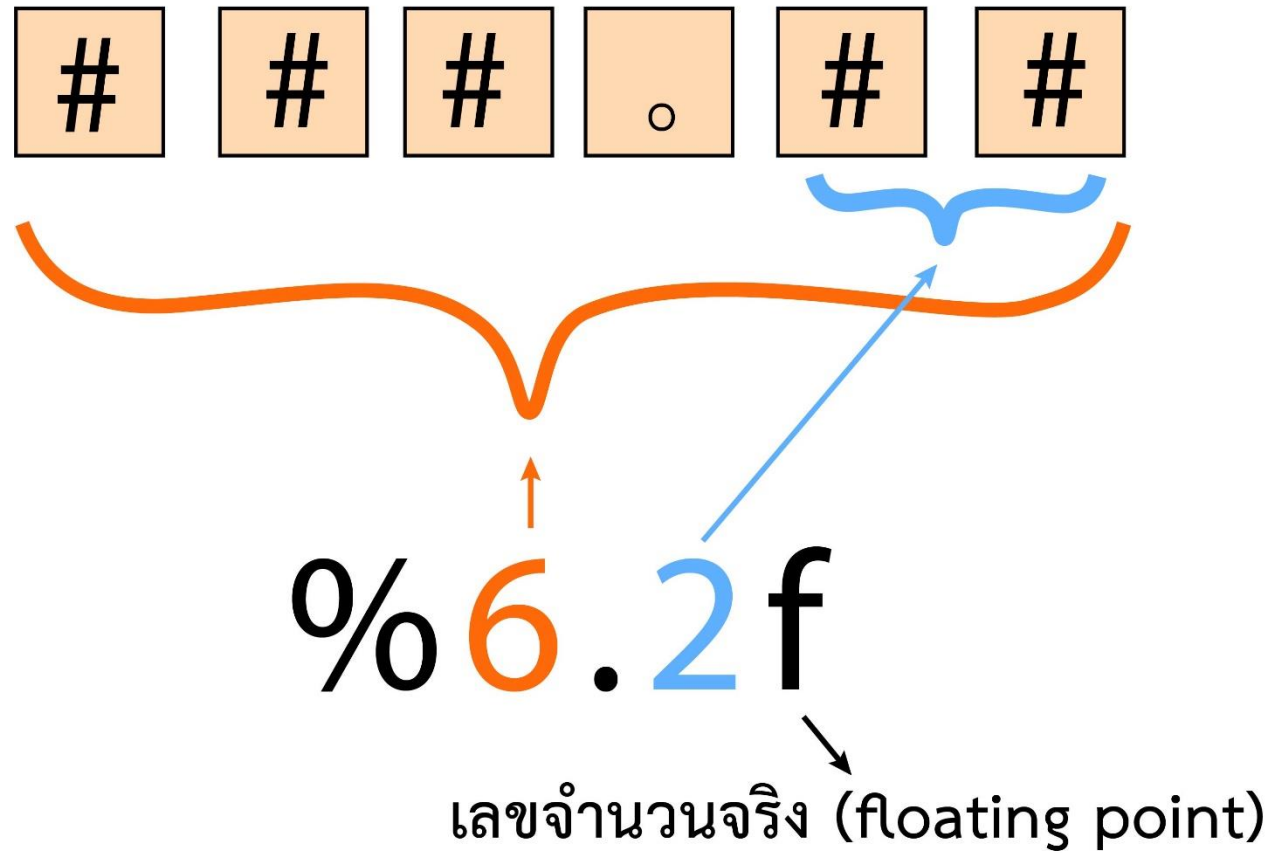
รูปแบบ	ประเภท	การใช้งาน
%d, %i	integer	แสดงจำนวนเต็ม
%u	unsigned int	เลขจำนวนเต็มแบบไม่มีเครื่องหมาย
%f	float	แสดงจำนวนทศนิยม
%c	character	ตัวอักษร
%s	string	แสดงผลแบบสตริง
%%	-	แสดงเครื่องหมาย %

ข้อควรระวัง

- ข้อควรระวังการเลือกชนิดที่ถูกต้อง

```
>>> x= 65
>>> print ("Temperature %d" %x)
Temperature 65
>>> print ("Temperature %c" %x)
Temperature A
```


จัดรูปแบบการแสดงผล



678.9

6	7	8	.	9	0
---	---	---	---	---	---

38

	3	8	.	0	0
--	---	---	---	---	---

54.321

	5	4	.	3	2
--	---	---	---	---	---

0.029

		0	.	0	3
--	--	---	---	---	---

1234.98

1	2	3	4	.	9	8
---	---	---	---	---	---	---

```
>>> Huminity = 75.56
>>> print("Huminity %5.1f, %5.2f, %5.3f" %(Huminity, Huminity, Huminity))
Huminity 75.6, 75.56, 75.560
>>> print("Huminity %8.1f, %8.2f, %8.3f" %(Huminity, Huminity, Huminity))
Huminity 75.6, 75.56, 75.560
```

- การใช้งานรูปแบบ .format()
- การแสดงการใช้ format เป็นการจัดรูปแบบสตริง

```
>>> Temperature = 25.0
>>> Huminity = 75.5
>>> print ("Temperature {}, Huminity {}".format(Temperature, Huminity) )
Temperature 25.0, Huminity 75.5
```

- กำหนดตัวแปรล่วงหน้า เพื่อความสะดวก

```
>>> DHT_Value = "Temperature {}, Huminity{}"
>>> print (DHT_Value.format(Temperature, Huminity) )
Temperature 25.0, Huminity75.5
```

```
>>> Temperature = 25.0
```

```
>>> Humidity = 75.5
```

```
>>> print ("Temperature {}, Humidity {}".format(Temperature, Humidity) )
```

```
Temperature 25.0, Humidity 75.5
```

```
>>> print ("Temperature {0}, Humidity {1}".format(Temperature, Humidity) )
```

```
Temperature 25.0, Humidity 75.5
```

```
>>> print ("Temperature {1}, Humidity {0}".format(Temperature, Humidity) )
```

```
Temperature 75.5, Humidity 25.0
```

```
>>> print ("Temperature {0:5.2f}, Humidity {0:5.2f}".format(Temperature, Humidity) )
```

```
Temperature 25.00, Humidity 25.00
```

เกิดอะไรขึ้นครับ ? ค่าที่ควรแสดง?