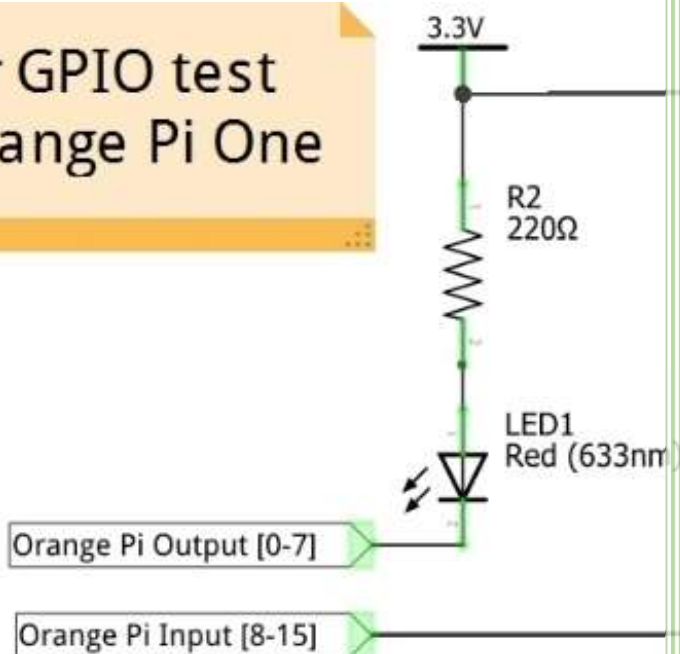


2016

การเขียนโปรแกรมพื้นฐาน 101

for GPIO test
Orange Pi One



เริ่มต้นกับ ภาษาไพทอน

กับ สวิตช์ และหลอดไฟ

10/21/2016

การเขียนโปรแกรมพื้นฐาน 101

Table of Contents

การเตรียมตัวเบื้องต้น	2
แนวคิดในการพัฒนาโปรแกรม	18
การพัฒนาโปรแกรมแบบสแตต-แมชชีน(State-Machine)	18
การพัฒนาโปรแกรมด้วยโฟลว์ชาร์ต	20
สรุปตัวอย่างคำสั่งต่างๆในภาษา Python	24
การประกาศตัวแปร ในรูปแบบต่างๆ	24
การแทรกคำอธิบาย(Comment)	24
ตัวแปรประเภทข้อความ (string)	24
ตัวแปรประเภทจำนวนเต็มและจำนวนเต็มขนาดยาว (Integer , Long)	26
ตัวแปรประเภทจำนวนทศนิยม (Float)	27
ตัวแปรเลขจำนวนเชิงซ้อน (Complex)	27
ตัวแปรประเภทชุดลำดับ(Lists)	28
ประเภทตัวแปรประเภท Tuples	29
ประเภทตัวแปรประเภท Dictionary	30
ตัวกระทำทางกรคำนวณ	31
ลำดับความสำคัญของตัวกระทำทางกรคำนวณ	37
คำสั่งสำคัญในการทำงาน	40
คำสั่งการวนซ้ำ, วนลูป (Loop)	40
คำสั่งควบคุมการวนซ้ำ	53
คำสั่งเงื่อนไข	59
เชิงอรรถ	64
อ้างอิง	65

การเขียนโปรแกรมพื้นฐาน 101

การเตรียมตัวเบื้องต้น

โปรแกรมแรกๆ ที่โปรแกรมเมอร์มักจะเริ่มต้นทำกันคือ การเขียนคำสั่งให้ แสดงผลคำว่า “Hello World” ลงบนคอนโซลแสดงผล สำหรับภาษา Python นั้น ใช้เพียงคำสั่งสั้นๆ ง่ายๆ ว่า

```
print “Hello World”
```

เพียงเท่านี้ เราก็จะสามารถแสดงผลคำว่า Hello World^[1] ได้แล้ว

แต่... จะพิมพ์ที่ไหนล่ะ? ยังไงล่ะ? พิมพ์แล้ว จะเห็นผลเลยหรือ? ต้องทำอะไรก่อนไหม?

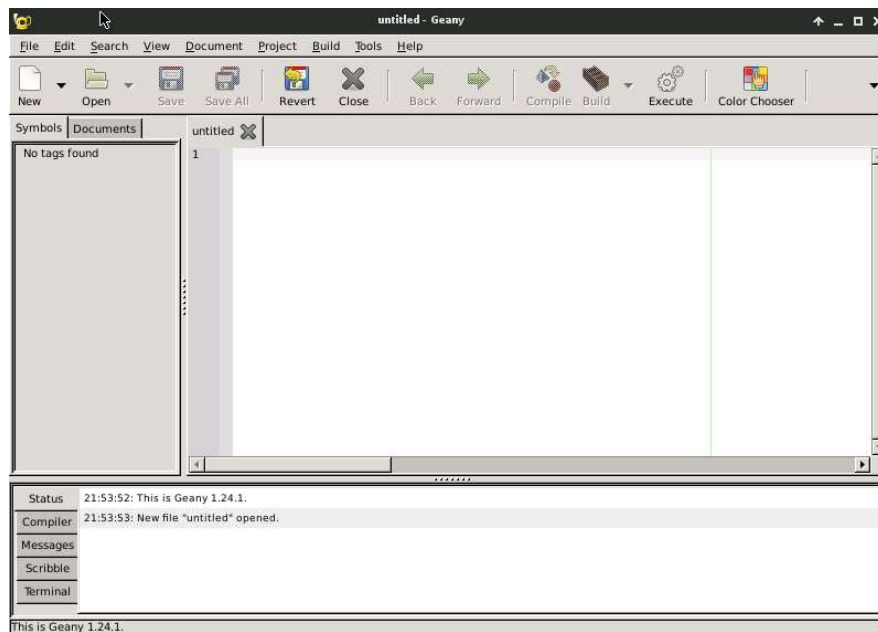
นี่แหละครับ จุดประสงค์ของคำว่า “Hello World” คือ ถ้าไม่สามารถทักทายกันได้ แสดงว่าเรายังใช้งานมัน ไม่ได้นั่นเอง

เรามาเริ่มกันเลย โดยเลือกที่เมนู *Application -> Development -> Geany*

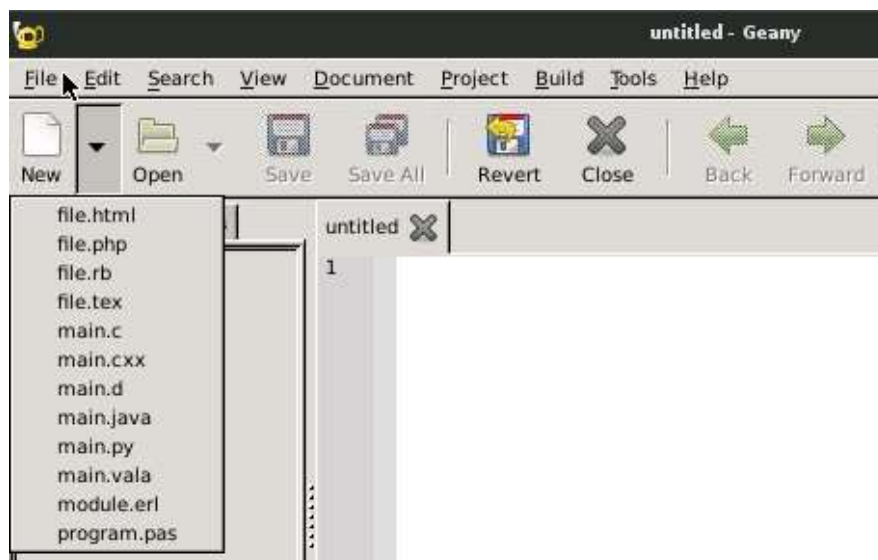


Geany คือ สิ่งที่เราเรียกว่า IDE (Integrated Development Environment) เป็นเครื่องมือ ที่ถูกสร้างขึ้นมา เพื่อให้เราสามารถพัฒนาโปรแกรมได้ง่ายมากขึ้น ยุ่งยากน้อยลง (แม้ว่าจะแคในบางครั้งก็ตาม) เมื่อเราเปิด ขึ้นมาก็จะพบหน้าต่างแบบนี้

การเขียนโปรแกรมพื้นฐาน 101

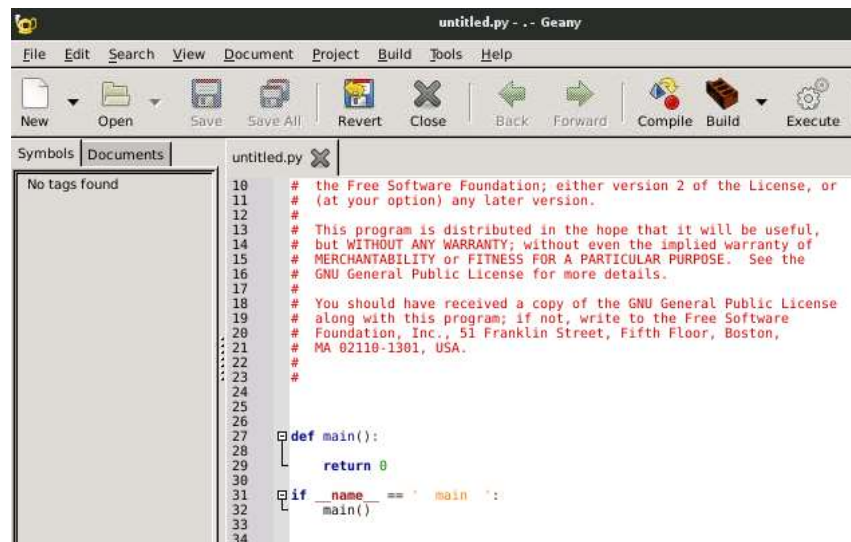


เมื่อเราจะสร้างโค้ด ก็เริ่มด้วยไปที่เมนู New -> main.py



จากนั้นเราก็จะได้ รูปแบบโค้ดมาตรฐานที่ Geany สร้างมา สำหรับให้เราเขียนโปรแกรมได้ง่ายขึ้น

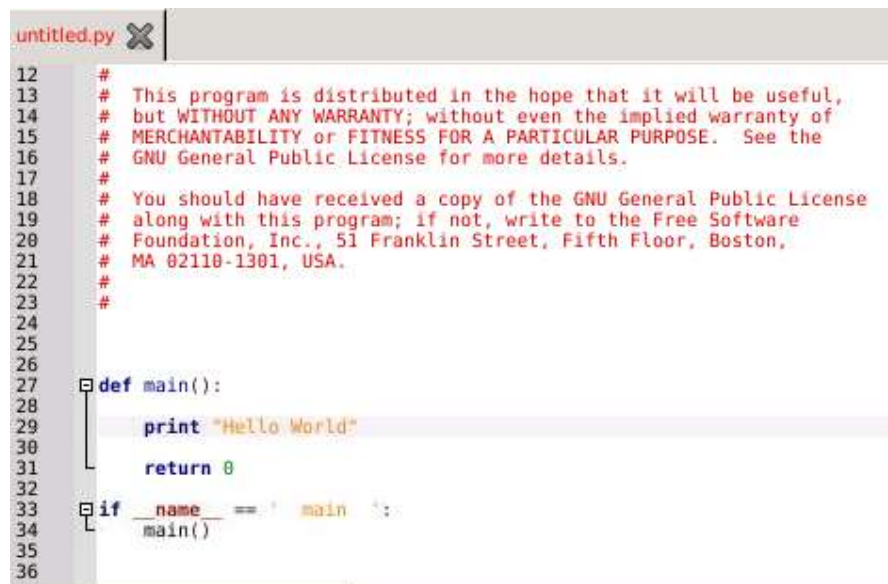
การเขียนโปรแกรมพื้นฐาน 101



```
10 # the Free Software Foundation; either version 2 of the License, or
11 # (at your option) any later version.
12 # This program is distributed in the hope that it will be useful,
13 # but WITHOUT ANY WARRANTY; without even the implied warranty of
14 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
15 # GNU General Public License for more details.
16 #
17 # You should have received a copy of the GNU General Public License
18 # along with this program; if not, write to the Free Software
19 # Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
20 # MA 02110-1301, USA.
21 #
22 #
23 #
24
25
26
27 def main():
28
29     return 0
30
31 if __name__ == '__main__':
32     main()
33
34
```

เมื่อเราได้โค้ดเริ่มต้นมาแล้ว และเป้าหมายอันยิ่งใหญ่ของเราคือ ทักทายกับคอมพิวเตอร์ด้วยคำว่า “Hello World”

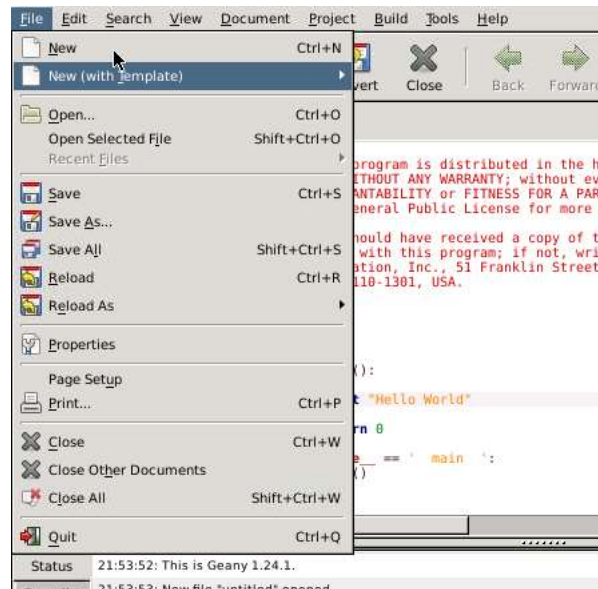
เราสามารถเพิ่มโค้ด “Hello World” ของเราได้ในบรรทัดหลัง `def main():` และก่อน `return 0` ดังตัวอย่าง



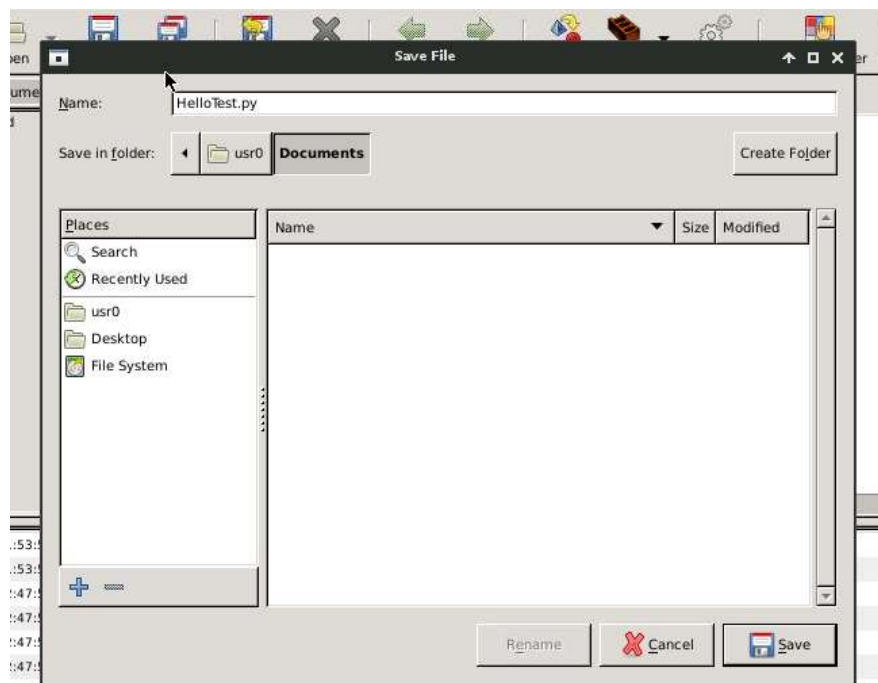
```
12 #
13 # This program is distributed in the hope that it will be useful,
14 # but WITHOUT ANY WARRANTY; without even the implied warranty of
15 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16 # GNU General Public License for more details.
17 #
18 # You should have received a copy of the GNU General Public License
19 # along with this program; if not, write to the Free Software
20 # Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
21 # MA 02110-1301, USA.
22 #
23 #
24 #
25
26
27 def main():
28     print "Hello World"
29
30     return 0
31
32 if __name__ == '__main__':
33     main()
34
35
36
```

การเขียนโปรแกรมพื้นฐาน 101

แล้วก็ทำการเซฟไฟล์ของเรา โดยไปที่เมนู File -> Save As



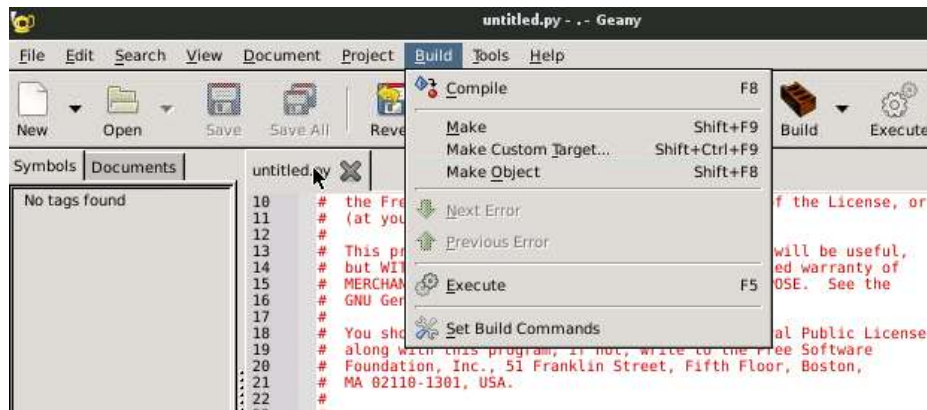
จากนั้นก็เปลี่ยนชื่อไฟล์ และเลือกที่เก็บไฟล์ได้ในหน้านี้ แล้วกด Save



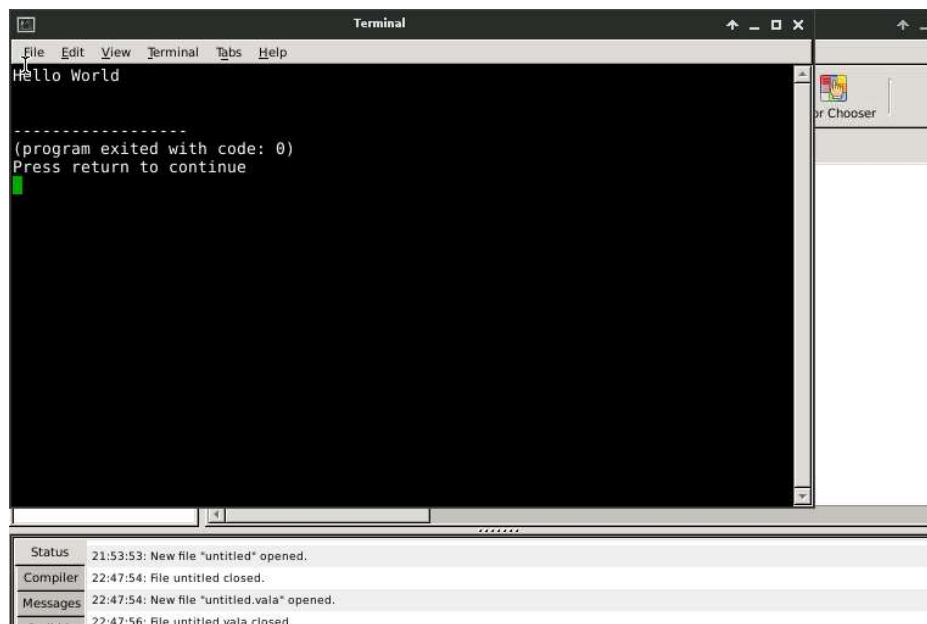
การเขียนโปรแกรมพื้นฐาน 101

เอาล่ะเมื่อทำทุกอย่างเรียบร้อยแล้ว เราจะมาลองรันโปรแกรมของเรากัน
โดยวิธีรันโปรแกรมนั้น มีได้ 3 ทางคือ

1. ใช้ทางลัด โดยกด F5
2. เข้าไปที่เมนู Build -> Execute (F5)
3. กดปุ่ม Execute ที่เป็นสัญลักษณ์เฟือง บนเมนู



แล้วก็จะได้พบกับ “Hello World” อย่างที่เราต้องการแล้ว



การเขียนโปรแกรมพื้นฐาน 101

การใช้งาน GPIO ด้วยไลบรารี WiringPi

โดยเริ่มต้นต้องตรวจสอบ Pin I/O ของไลบรารีว่า มีขาไหน ใช้ทำอะไรบ้าง ด้วยการใช้คำสั่ง `gpio readall` ใน Terminal

1. เปิด Xfce Terminal โดยเริ่มที่ Application Menu -> System -> Xfce Terminal



2. พิมพ์คำสั่ง `gpio readall` จะมีรายละเอียดขึ้นดังภาพ

การเขียนโปรแกรมพื้นฐาน 101

```
usr0@orangepione:~/Public/WiringPi-Python-OP/WiringPi$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name     | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     | 3.3v     |      |   | 1 | 2 |     | 5v       |     |     | |
|  2  |  8  | SDA.0    | ALT5 | 0 | 3 | 4 |     | 5V       |     |     |
|  3  |  9  | SCL.0    | ALT5 | 0 | 5 | 6 |     | 0v       |     |     |
|  4  |  7  | GPIO.7   | ALT3 | 0 | 7 | 8 | 0 | ALT3 | TxD3     | 15 | 14 |
|     |     | 0v       |      |   | 9 | 10 | 0 | ALT3 | RxD3     | 16 | 15 |
| 17  |  0  | RxD2     | ALT3 | 0 | 11 | 12 | 0 | ALT3 | GPIO.1   |  1 | 18 |
| 27  |  2  | TxD2     | ALT3 | 0 | 13 | 14 |   |     | 0v       |   |   |
| 22  |  3  | CTS2     | ALT3 | 0 | 15 | 16 | 0 | ALT3 | GPIO.4   |  4 | 23 |
|     |     | 3.3v     |      |   | 17 | 18 | 0 | ALT3 | GPIO.5   |  5 | 24 |
| 10  | 12  | MOSI     | ALT4 | 0 | 19 | 20 |   |     | 0v       |   |   |
|  9  | 13  | MISO     | ALT4 | 0 | 21 | 22 | 0 | ALT3 | RTS2     |  6 | 25 |
| 11  | 14  | SCLK     | ALT4 | 0 | 23 | 24 | 0 | ALT4 | CE0      | 10 |  8 |
|     |     | 0v       |      |   | 25 | 26 | 0 | ALT3 | GPIO.11  | 11 |  7 |
|  0  | 30  | SDA.1    | ALT4 | 0 | 27 | 28 | 0 | ALT4 | SCL.1    | 31 |  1 |
|  5  | 21  | GPIO.21  | ALT3 | 0 | 29 | 30 |   |     | 0v       |   |   |
|  6  | 22  | GPIO.22  | ALT3 | 0 | 31 | 32 | 0 | ALT3 | RTS1     | 26 | 12 |
| 13  | 23  | GPIO.23  | ALT3 | 0 | 33 | 34 |   |     | 0v       |   |   |
| 19  | 24  | GPIO.24  | ALT3 | 0 | 35 | 36 | 0 | ALT3 | CTS1     | 27 | 16 |
| 26  | 25  | GPIO.25  | ALT3 | 0 | 37 | 38 | 0 | ALT3 | TxD1     | 28 | 20 |
|     |     | 0v       |      |   | 39 | 40 | 0 | ALT3 | RxD1     | 29 | 21 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name     | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
usr0@orangepione:~/Public/WiringPi-Python-OP/WiringPi$
```

ในส่วนที่ WiringPi ใช้งานคือขาในช่อง wPi

เช่นหากเราเลือกใช้ขา pin 1 จะหมายถึง pin 1 บนตาราง wPi ซึ่งจะตรงกับขาที่ 12 ของ Physical – Name
GPIO.1 – BCM 18

```
usr0@orangepione:~/Public/WiringPi-Python-OP/WiringPi$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name     | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     | 3.3v     |      |   | 1 | 2 |     | 5v       |     |     | |
|  2  |  8  | SDA.0    | ALT5 | 0 | 3 | 4 |     | 5V       |     |     |
|  3  |  9  | SCL.0    | ALT5 | 0 | 5 | 6 |     | 0v       |     |     |
|  4  |  7  | GPIO.7   | ALT3 | 0 | 7 | 8 | 0 | ALT3 | TxD3     | 15 | 14 |
|     |     | 0v       |      |   | 9 | 10 | 0 | ALT3 | RxD3     | 16 | 15 |
| 17  |  0  | RxD2     | ALT3 | 0 | 11 | 12 | 0 | ALT3 | GPIO.1   |  1 | 18 |
| 27  |  2  | TxD2     | ALT3 | 0 | 13 | 14 |   |     | 0v       |   |   |
| 22  |  3  | CTS2     | ALT3 | 0 | 15 | 16 | 0 | ALT3 | GPIO.4   |  4 | 23 |
|     |     | 3.3v     |      |   | 17 | 18 | 0 | ALT3 | GPIO.5   |  5 | 24 |
| 10  | 12  | MOSI     | ALT4 | 0 | 19 | 20 |   |     | 0v       |   |   |
|  9  | 13  | MISO     | ALT4 | 0 | 21 | 22 | 0 | ALT3 | RTS2     |  6 | 25 |
| 11  | 14  | SCLK     | ALT4 | 0 | 23 | 24 | 0 | ALT4 | CE0      | 10 |  8 |
|     |     | 0v       |      |   | 25 | 26 | 0 | ALT3 | GPIO.11  | 11 |  7 |
|  0  | 30  | SDA.1    | ALT4 | 0 | 27 | 28 | 0 | ALT4 | SCL.1    | 31 |  1 |
|  5  | 21  | GPIO.21  | ALT3 | 0 | 29 | 30 |   |     | 0v       |   |   |
|  6  | 22  | GPIO.22  | ALT3 | 0 | 31 | 32 | 0 | ALT3 | RTS1     | 26 | 12 |
| 13  | 23  | GPIO.23  | ALT3 | 0 | 33 | 34 |   |     | 0v       |   |   |
| 19  | 24  | GPIO.24  | ALT3 | 0 | 35 | 36 | 0 | ALT3 | CTS1     | 27 | 16 |
| 26  | 25  | GPIO.25  | ALT3 | 0 | 37 | 38 | 0 | ALT3 | TxD1     | 28 | 20 |
|     |     | 0v       |      |   | 39 | 40 | 0 | ALT3 | RxD1     | 29 | 21 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name     | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
usr0@orangepione:~/Public/WiringPi-Python-OP/WiringPi$
```

การเขียนโปรแกรมพื้นฐาน 101

เมื่อเรามาดูบนบอร์ดเสริมของเรา เมื่อเทียบกันก็จะเห็นว่า ขาที่ 12 นั้น จะตรงกับขา Name 1 – WiringPi 1
– BCM GPIO 18 – GPIO 18

นอกเหนือไปจากขา I/O แล้ว ยังมีขาที่ใช้งานสำหรับเป็นแหล่งพลังงาน หรือไฟเลี้ยงวงจร อีก 3 ชนิดคือ

1. -> 5v ที่ขา Physical 2 และ 4 ให้อายุไฟขนาดแรงดัน 5v (กระแสสูงสุดไม่เกิน 100mA)
2. -> 3.3v ที่ขา Physical 1 และ 17 ให้อายุไฟขนาดแรงดัน 3.3v (กระแสสูงสุดไม่เกิน 100mA)
3. -> 0v (GND) ที่ขา physical 6 , 9 , 14 , 20 , 25 , 30 , 34 และ 39 เป็นแรงดันไฟขนาด 0v (GND,กราวนด์)

Name	wiringPi Pin	BCM GPIO	BCM GPIO	wiringPi Pin	Name
GPIO Extension Board					
3.3V	-	-	3V3	5V0	5V
SDA	8	R1:0/R2:2	SDA1	5V0	5V
SCL	9	R1:1/R2:3	SCL1	GND	0V
GPIO7	7	4	GPIO4	TXD0	14
GND	-	-	GND	RXD0	15
GPIO0	0	17	GPIO17	GPIO18	18
GPIO2	2	R1:21/R2:27	GPIO27	GND	-
GPIO3	3	22	GPIO22	GPIO23	23
3.3v	-	-	3V3	GPIO24	24
MOSI	12	10	SPIMOSI	GND	-
MISO	13	9	SPIMISO	GPIO25	25
SCLK	14	11	SPISCLK	SPICE0	8
0V	-	-	GND	SPICE1	7
ID_SDA	30	0	ID_SD	ID_SC	1
GPIO21	21	5	GPIO5	GND	-
GPIO22	22	6	GPIO6	GPIO12	12
GPIO23	23	13	GPIO13	GND	-
GPIO24	24	19	GPIO19	GPIO16	16
GPIO25	25	26	GPIO26	GPIO20	20
GND	-	-	GND	GPIO21	21

การเขียนโปรแกรมพื้นฐาน 101



ตัวอย่างการใช้งาน I/O ของไลบรารี wiringpi กับบอร์ดขยาย

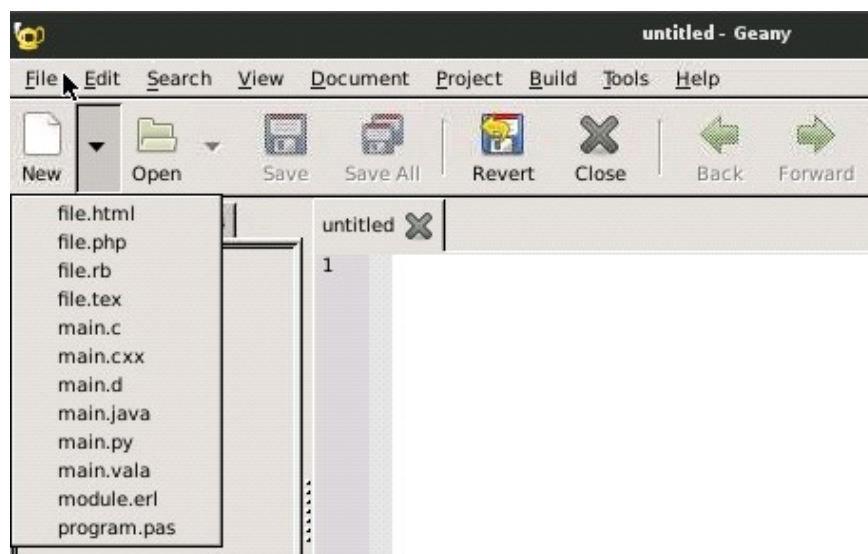
ในตัวอย่างนี้ เราจะใช้ wPi 2 เป็นขารับค่าอินพุต (input) จากสวิตช์และ wPi 4 เป็นขาควบคุมเอาต์พุต (output) ไปยังหลอดไฟ LED

การเขียนโปรแกรมพื้นฐาน 101

ก่อนอื่นเราต้องเปิดโปรแกรม Geany ที่ใช้สำหรับเขียนโปรแกรมขึ้นมาก่อน โดยเข้าไปที่ Application Menu -> Development -> Geany



จากนั้นก็สร้างไฟล์ โปรแกรมขึ้นมาโดยไปที่รูปลูกศรชี้ลงข้างๆ เมนู New



การเขียนโปรแกรมพื้นฐาน 101

อันดับแรกของการเขียนโปรแกรม เราประกาศตัวแปร เพื่อใช้แทนค่าที่ระบุการใช้งานของขา wPi เสียก่อน ดังนี้

```
IO_INPUT = 0 # ระบุว่าใช้ IO_INPUT แทนการใช้งานขาเพื่อรับค่าอินพุต
```

```
IO_OUTPUT = 1 # ระบุว่าใช้ IO_OUTPUT แทนการใช้งานขาเพื่อควบคุมเอาต์พุต
```

ประกาศตัวแปรแทนสถานะของสวิตช์คือ

```
SW_PD = 0 # แทนสวิตช์ถูกกด
```

```
SW_RL = 1 # แทนสวิตช์ถูกปล่อย
```

ประกาศตัวแปรเพื่อแทนขา ที่เราใช้เชื่อมต่อกับสวิตช์

```
SW_PIN = 2 # ระบุว่าต่อสวิตช์เข้ากับขา wPi 2
```

จากนั้นเราก็ประกาศตัวแปรแทนสถานะของหลอดไฟที่เราจะควบคุม

```
LED_ON = 1 # แทนหลอดไฟติด
```

```
LED_OFF = 0 # แทนหลอดไฟดับ
```

ประกาศตัวแปรแทนขา ที่เราใช้เชื่อมต่อกับหลอดไฟ

```
LED_PIN = 4 # ระบุว่าเราใช้ขา wPi 4 ในการควบคุมหลอดไฟ
```

แล้วเราก็มานิยามให้โปรแกรมรู้จักกับสวิตช์ และหลอดไฟ รวมถึงกำหนดสถานะเริ่มต้นให้กับทั้งสวิตช์ และหลอดไฟด้วย

```
Sw = SW_RL # สร้างสวิตช์ขึ้นในโปรแกรม และกำหนดสถานะเริ่มต้นให้เป็นการปล่อยสวิตช์
```

```
Led = LED_OFF # สร้างหลอดไฟ LED ขึ้นในโปรแกรม และกำหนดสถานะเริ่มต้นให้เป็น สถานะดับไฟ
```

```
import wiringpi # เรียกใช้งานคำสั่งต่างๆของ wiringpi
```

ใน def main():

```
เพิ่มคำสั่ง wiringpi.wiringPiSetup() # เพื่อเปิดใช้งาน GPIO ผ่าน wiringpi
```


การเขียนโปรแกรมพื้นฐาน 101

ในบรรทัดต่อมาเพิ่มคำสั่ง `print "Pin 2 to SW"` # เพื่อแจ้งให้ผู้ใช้ได้ทราบว่าเราเลือกที่ขา wPi 2 ในการใช้งานกับสวิตช์

และเพิ่มคำสั่ง `wiringpi.pinMode(SW_PIN,IO_INPUT)` # เพื่อตั้งค่าให้ขา wPi 2 ทำหน้าที่เป็นขารับค่าอินพุตในบรรทัดต่อมา

จากนั้นจึงมาตั้งค่า LED ด้วย

คำสั่ง `print "Pin 4 to LED"` # เพื่อบอกผู้ใช้ว่า เราเลือกขา wPi 4 เป็นขาควบคุม LED

และคำสั่ง `wiringpi.pinMode(LED_PIN,IO_OUTPUT)` # เพื่อตั้งค่าให้ขา wPi 4 เป็นขาควบคุมเอาต์พุต

หลังจากตั้งค่าของระบบการทำงานเสร็จแล้ว เราจึงเริ่มต้นเขียนโค้ดเพื่อควบคุมการทำงานของระบบอีกที

ในการควบคุมการทำงานของระบบนั้น เราอยากจะให้ระบบทำงานไปเรื่อยๆ ดังนั้นเราต้องสร้างลูป ให้ระบบทำงานเป็นรอบๆ ไปเรื่อยๆ ด้วยคำสั่ง

while 1 : #สร้างลูปที่วนไปเรื่อยๆ

เพิ่มคำสั่งเพื่ออ่านค่าจากสวิตช์ที่เรากำหนดไว้ที่ขา wPi 2 ด้วยคำสั่ง `sw = wiringpi.digitalRead(SW_PIN)`

SW_PIN คือตัวแปรที่ทำหน้าที่แทนขา wPi 2 ตามที่เราได้กำหนดไว้ตอนต้น

เพิ่มคำสั่งตรวจสอบค่าที่ได้รับจากสวิตช์ว่ามีการกดอยู่หรือไม่ `if sw == SW_PD :`

และเพิ่มคำสั่งการทำงาน สำหรับเงื่อนไขที่มีการกดสวิตช์อยู่ด้วย การเปลี่ยนสถานะให้ หลอดไฟ LED ติด ด้วย

คำสั่ง `led = LED_ON`

ถ้ามีการกดอยู่จะทำให้เงื่อนไขนี้เป็นจริง โปรแกรมก็จะทำคำสั่งในเงื่อนไขนี้ คือการเปลี่ยนสถานะ LED โดย

`led = LED_ON`

และเช่นกันจำเป็นจะต้องเพิ่มคำสั่งสำหรับกรณีที่มีการปล่อยสวิตช์เช่นกัน `if sw == SW_RL :`

และตามด้วยคำสั่งการทำงาน สำหรับเงื่อนไขที่มีการปล่อยการกดปุ่มสวิตช์ด้วยการเปลี่ยนสถานะให้ หลอดไฟ

LED ดับลง ด้วยคำสั่ง `led = LED_OFF`

ดังนั้นหากมีการปล่อยการกดสวิตช์ จะทำให้เงื่อนไขนี้เป็นจริง และโปรแกรมจะทำการเปลี่ยนสถานะ LED โดย

`led = LED_OFF`

การเขียนโปรแกรมพื้นฐาน 101

หลังจากจัดการเรื่องการกำหนดสถานะของ LED ให้เป็นไปตามเงื่อนไขต่างๆ ของสวิตช์แล้ว

โปรแกรมจะเป็นที่จะต้องสั่งให้หลอดไฟ LED จริงๆ มีสถานะเป็นไปตาม สถานะ LED ในโปรแกรมนี โดยใช้คำสั่ง `wiringpi.digitalWrite(LED_PIN,led)` โดย LED_PIN คือขาที่เราใช้เชื่อมต่อกับ LED ซึ่งเราได้กำหนดไว้ตอนต้น และ led คือสถานะของหลอดไฟ LED ในโปรแกรมของเรา เพื่อให้หลอดไฟ LED ของจริงถูกควบคุมตามสถานะ LED ในโปรแกรม

แล้วจึงจบด้วย `return 0` ก็จะจบโปรแกรม

หลังจากเราเขียนโปรแกรมเสร็จแล้ว เราจะได้หน้าตาแบบนี้

```
IO_INPUT = 0
IO_OUTPUT = 1

SW_PD = 0
SW_RL = 1
SW_PIN = 2

LED_ON = 1
LED_OFF = 0
LED_PIN = 4

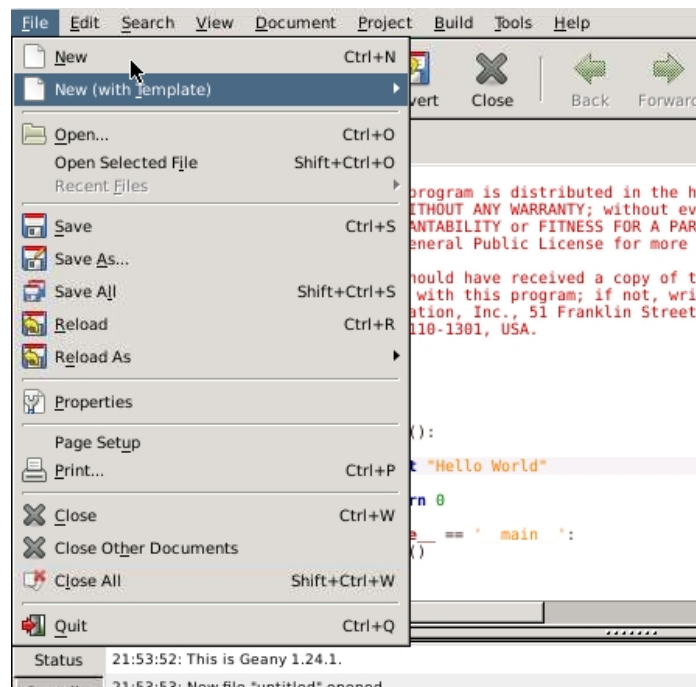
sw = SW_RL
led = LED_OFF

import wiringpi

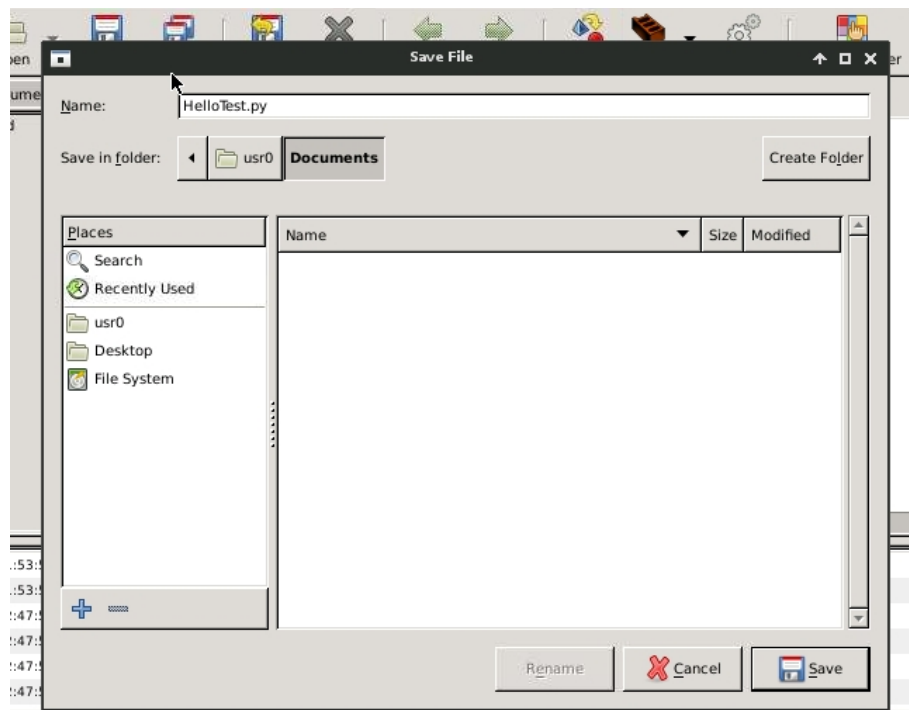
def main():
    wiringpi.wiringPiSetup()
    print "Pin 2 to SW"
    wiringpi.pinMode(SW_PIN, IO_INPUT)
    print "Pin 4 to LED"
    wiringpi.pinMode(LED_PIN, IO_OUTPUT)
    while 1 :
        sw = wiringpi.digitalRead(SW_PIN)
        if sw == SW_PD :
            led = LED_ON
        if sw == SW_RL :
            led = LED_OFF
        wiringpi.digitalWrite(LED_PIN,led)
    return 0
```

แล้วเราจึงทำการ Save ไฟล์โค้ดโปรแกรมของเรา โดยไปที่ File -> Save As

การเขียนโปรแกรมพื้นฐาน 101

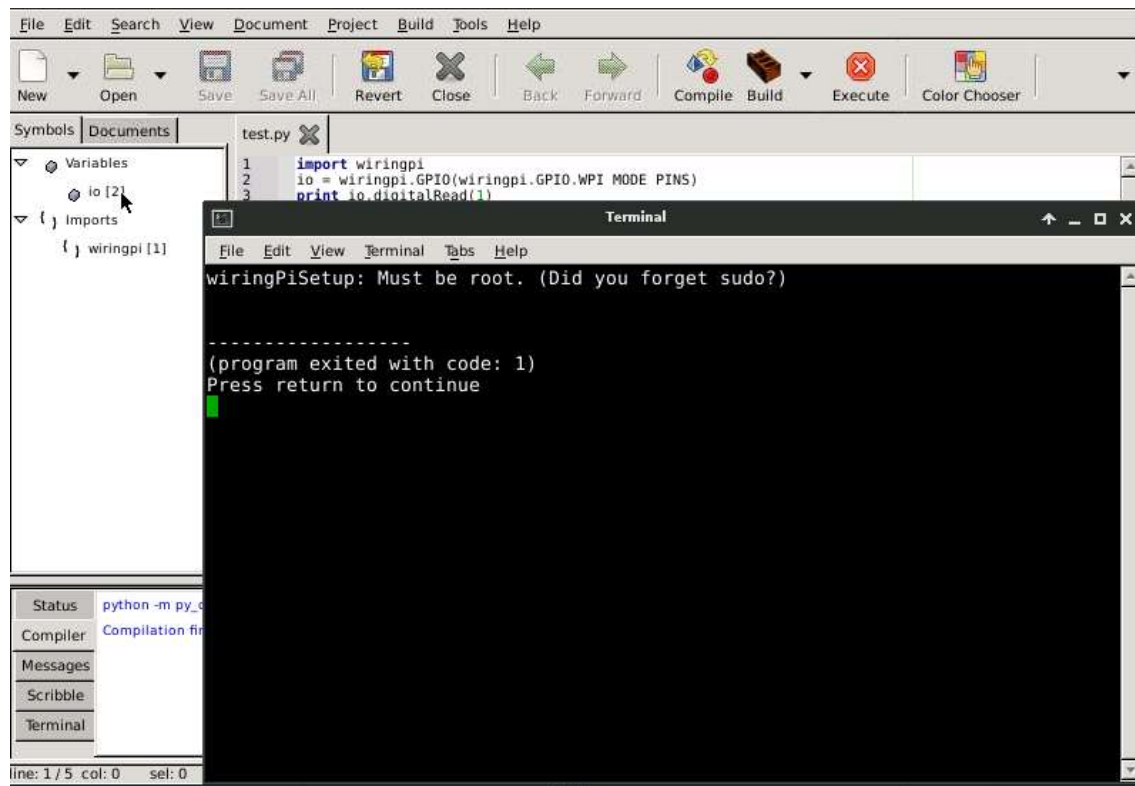


แล้วแก้ไขชื่อและนามสกุลในช่อง Name ซึ่งนามสกุลของไฟล์โปรแกรมของเรานั้นจะต้องเป็น .py เท่านั้น เช่น test.py เป็นต้น



การเขียนโปรแกรมพื้นฐาน 101

จากนั้นจึงทดลองรันโปรแกรมที่เราเขียน ถ้าหากมีข้อความแสดงขึ้นว่า “wiringPiSetup : Must be root.
(Did you forget sudo?)” ดังภาพ

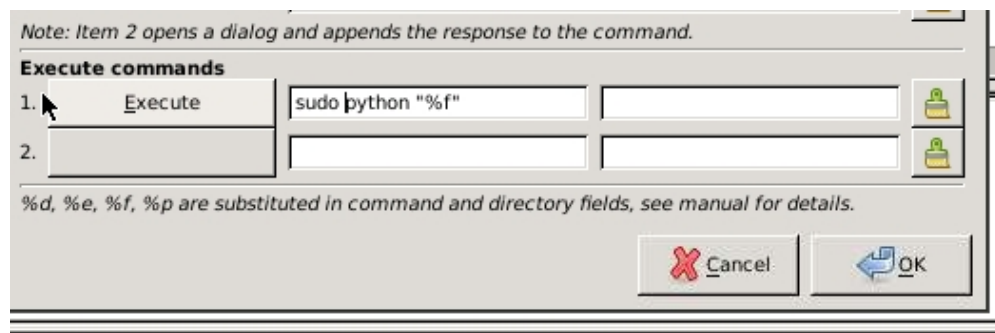


แสดงว่าโปรแกรมที่เราเขียนนั้น จำเป็นที่จะต้องทำงานในสถานะ root (สถานะผู้ดูแลระบบ Administrator)
ตัวนั้นเราจึงต้องกลับไปแก้ไขให้ โปรแกรมรันในฐานะผู้ดูแลระบบเสียก่อน โดยไปที่ Build->Set Build
Commands

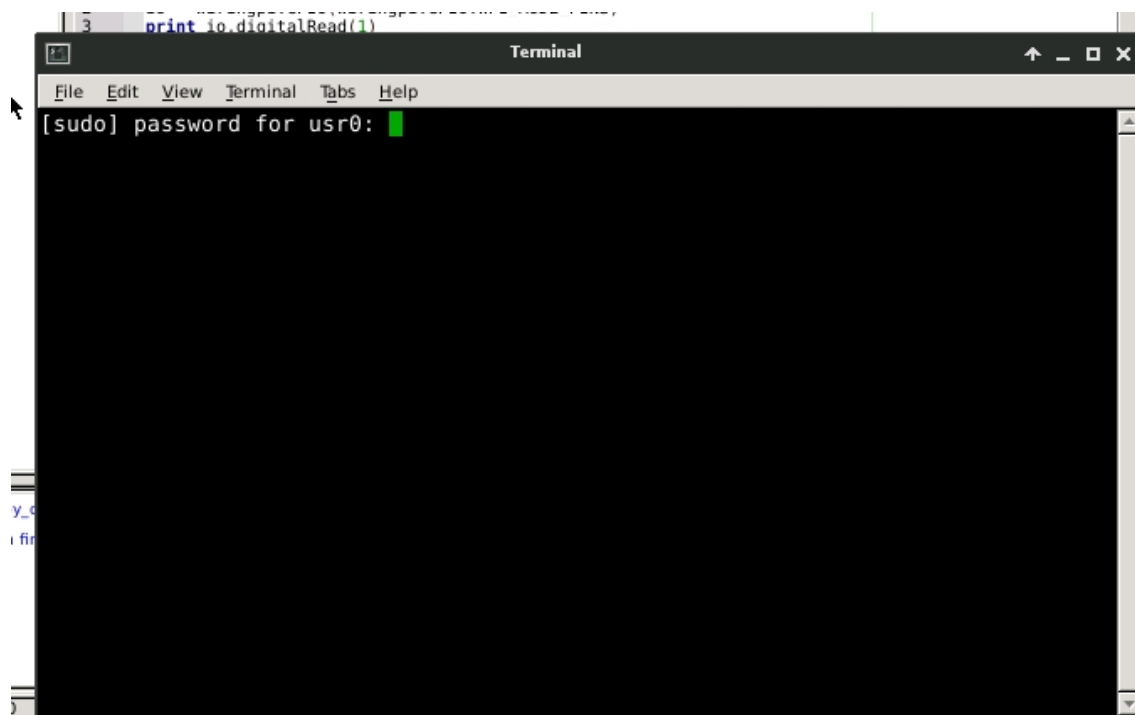


แล้วเพิ่ม sudo ลงในข้างหน้าของ python “%f” ในช่อง Execute แล้วลองรันดูอีกครั้ง

การเขียนโปรแกรมพื้นฐาน 101



ในครั้งนี้จะถามเพียงแค่ รหัสผ่าน (Password) เท่านั้น หากใส่รหัสผ่านถูกต้องโปรแกรมก็จะเริ่มทำงาน



หากไม่ต้องการใส่รหัสผ่าน ในช่องของ Execute นั้นให้ใส่ข้อความเพิ่มเข้าไปดังนี้

```
echo xxรหัสผ่านxx | sudo python -S \"%f\"
```

แนวคิดในการพัฒนาโปรแกรม

แนวคิดในการพัฒนาโปรแกรมนั้น มีการพัฒนามายาวนาน โดยอาศัยพื้นฐานจากการพัฒนากลไกของเครื่องจักรกล ที่มีการทำงานที่ละขั้นตอนต่อกันไป

ซึ่งในสมัยแรกของระบบคอมพิวเตอร์นั้น(ช่วงก่อนปี 1936 ที่อลัน ทัวริงนำเสนอแนวคิดเรื่อง คอมพิวเตอร์สมัยใหม่) มันไม่มีความสามารถในการคำนวณมากนัก และยังไม่มีความคิดในการจัดเก็บข้อมูลอย่างเป็นรูปแบบอีกด้วย ทำให้งานหลักของมันยังคงเป็นเพียงทำงานตามขั้นตอนที่ถูกกำหนดไว้แล้ว โดยอาศัยการออกแบบ และพัฒนาโปรแกรมด้วยแนวคิด สเตต-แมคชีน (State-Machine) ที่มีการพึ่งพาการคำนวณน้อย และไม่มีการจัดการหน่วยความจำ

แต่หลังจากปี 1949 ซึ่งเป็นปีที่ แนวคิดคอมพิวเตอร์สมัยใหม่ ถูกทำให้กลายเป็นเครื่องคอมพิวเตอร์จริงๆ ที่ชื่อ EDSAC โดย EDSAC นั้นมีโครงสร้างคล้ายคอมพิวเตอร์สมัยนี้มาก มีทั้งหน่วยประมวลผล และหน่วยความจำในเวลาเดียวกันเอง ที่แนวคิดการพัฒนาโปรแกรมโดยใช้โฟลว์ชาร์ต ถูกนำมาใช้อย่างจริงจัง เนื่องจากคอมพิวเตอร์ EDSAC ถูกพัฒนามาบรรทัดประสงค์ให้คอมพิวเตอร์สามารถทำงาน และคำนวณอย่างซับซ้อนได้ ดังนั้นแล้วการออกแบบโปรแกรมด้วยโฟลว์ชาร์ตนั้นจึงเหมาะสมกับคอมพิวเตอร์สมัยใหม่มากกว่า

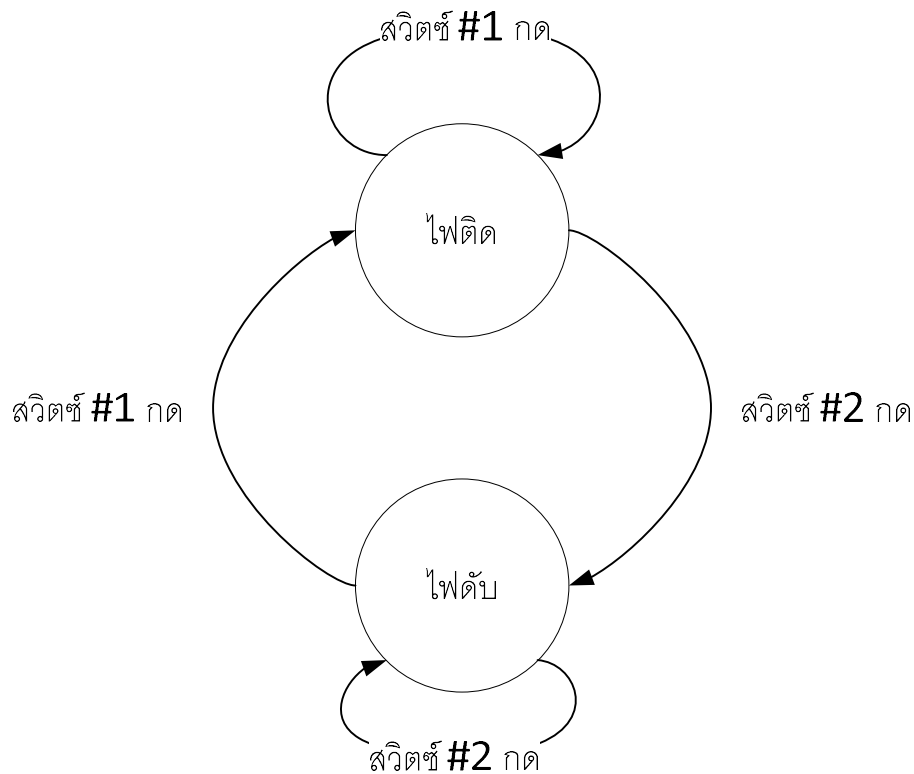
การพัฒนาโปรแกรมแบบสเตต-แมคชีน(State-Machine)

สเตต-แมคชีนนั้น ไม่มีสัญลักษณ์ และอะไรที่ซับซ้อน แนวความคิดของมันคือ การทำงานตามสถานะต่างๆของโปรแกรม ดังนั้นโครงสร้างโปรแกรมจึงประกอบไปด้วยเพียง 2 สัญลักษณ์หลัก คือ “การทำงาน” (Action) และ “สถานะ” (State) เช่น

เราต้องการพัฒนาโปรแกรม ให้หลอดไฟ LED ติดดับ จากการควบคุมด้วยสวิตช์ 2 ตัว โดยให้ตัวหนึ่ง กดเพื่อให้ไฟติด และอีกหนึ่งตัว กด เพื่อให้ไฟดับ

เราจะพบว่า ระบบของเรามีการ”ทำงาน” 2 อย่างคือ “ไฟติด” และ”ไฟดับ” ซึ่งถูกควบคุมจากสถานะของการ ”กด” ของสวิตช์ 2 ตัว

การเขียนโปรแกรมพื้นฐาน 101



ตัวอย่างโค้ดจากสเตต-แมชชีนนี้

```
LED_ON = 1    #Action เปิด LED
LED_OFF = 0   #Action ปิด LED
SW_PSHD = 1   #State เมื่อกดสวิตช์
SW_RLSD = 0   #State เมื่อปล่อยสวิตช์
SW1_PIN = 8   #Pin สำหรับสวิตช์#1
SW2_PIN = 9   #Pin สำหรับสวิตช์#2
LED_PIN = 7   #Pin สำหรับ LED

import wiringpi

sw1 = 0       #ตัวแปร sw1 สำหรับ สวิตช์#1
sw2 = 0       #ตัวแปรsw2 สำหรับสวิตช์#2
led = 0       #ตัวแปรled สำหรับ LED

def main():
    wiringpi.wiringPiSetup()

    print "Simplify State-Machine"
    print ""Switch#1 had pressed for turn LED ON
          Switch#2 had pressed for turn LED OFF""
    print "Pin 7 to LED"
    wiringpi.pinMode(LED_PIN,1)
    print "Pin 8 to Swich#1"
    wiringpi.pinMode(SW1_PIN,0)
    print "Pin 9 to Swich#2"
```

การเขียนโปรแกรมพื้นฐาน 101

```
wiringpi.pinMode(SW2_PIN, 0)
while 1 :
    sw1 = wiringpi.digitalRead(SW1_PIN)    #อับเฉาสถานะของสวิตช์#1 ไปเก็บไว้ที่ ตัวแปร sw1
    sw2 = wiringpi.digitalRead(SW2_PIN)    #อับเฉาสถานะของสวิตช์#2 ไปเก็บไว้ที่ ตัวแปร sw2
    if sw1 == SW_PSHD :                    #ถ้า sw1 มีสถานะเท่ากับ SW_PSHD
        led = LED_ON                      #ไฟ LED ติด
    if sw2 == SW_PSHD :                    #ถ้า sw2 มีสถานะเท่ากับ SW_PSHD
        led = LED_OFF                     #ไฟ LED ดับ
    wiringpi.digitalWrite(LED_PIN, led)    #ให้ หลอดLED ทำงานตามสถานะของ led
return 0
```

การพัฒนาโปรแกรมด้วยโฟลว์ชาร์ต

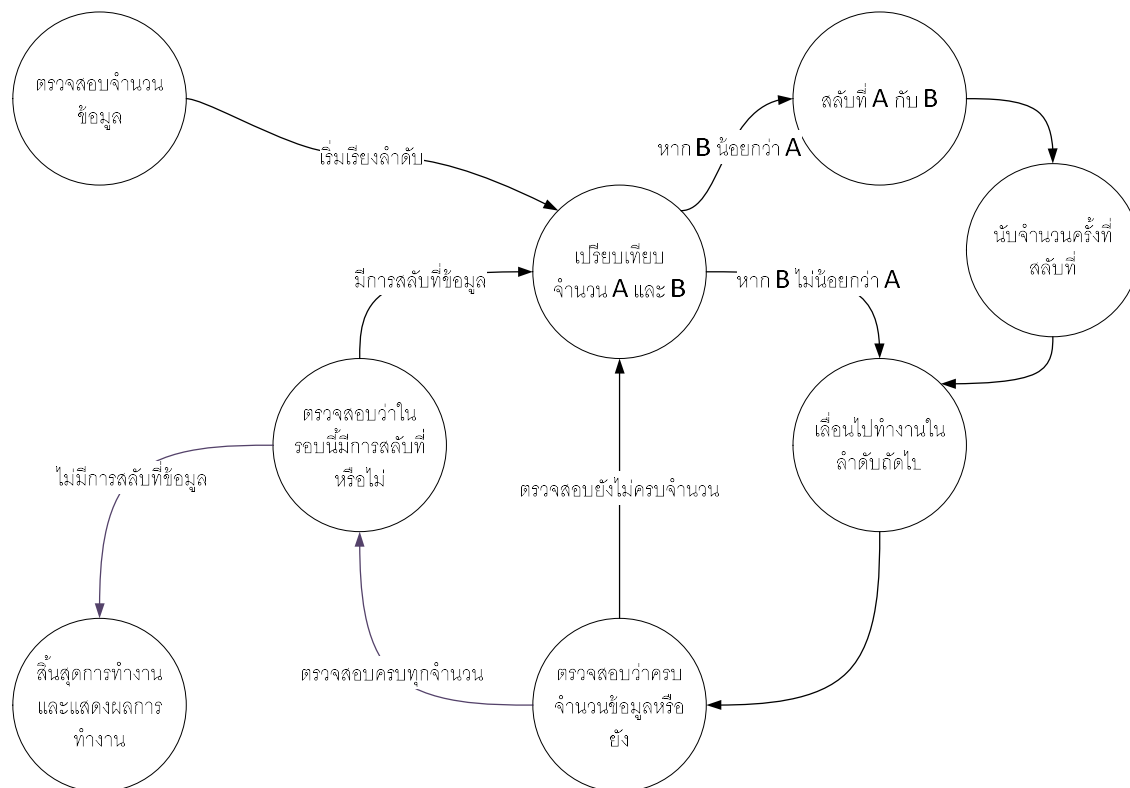
แนวความคิดในการใช้โฟลว์ชาร์ตในการพัฒนาโปรแกรมนั้น เกิดจากความต้องการใช้คอมพิวเตอร์เข้ามาช่วยในการทำงานวิจัย และศึกษา ในกระบวนการทางวิทยาศาสตร์ และวิศวกรรม เนื่องจากการพัฒนาโปรแกรมด้วยแนวคิดดั้งเดิมอย่าง สเตต-แมคชีนนั้น ไม่สามารถตอบสนองการพัฒนาโปรแกรมที่จำเป็นต้องมีการคำนวณทางคณิตศาสตร์ หรือมีกระบวนการสังเคราะห์ข้อมูลทางวิทยาศาสตร์ได้ เพราะความยากลำบากในการตรวจสอบกระบวนการทำงาน ซึ่งส่งผลโดยตรงต่อความน่าเชื่อถือของกระบวนการ เนื่องจากแนวคิดของสเตต-แมคชีนนั้น มีรูปแบบใช้งานอย่างจำกัด โดยมีเพียงแค่ สัญลักษณ์วงกลมที่สื่อแทนการทำงาน และเส้นลากที่สื่อแทนสถานะ ให้ใช้งาน ต่างจากแนวคิดของโฟลว์ชาร์ต ที่มีทางเลือกมากกว่า สัญลักษณ์ที่มากกว่า และมีความหมายที่ชัดเจนในแต่ละสัญลักษณ์ ทำให้สามารถทำความเข้าใจได้ง่ายกว่า เมื่อนำมาออกแบบโปรแกรมที่ซับซ้อน

การปรับเปลี่ยนโครงสร้างของคอมพิวเตอร์ และแนวคิดในการพัฒนาโปรแกรมนี้นำส่งผลอย่างมากต่อทิศทางการพัฒนาโปรแกรม เพราะทำให้เกิดโปรแกรมที่เรียกว่า อัลกอริทึม (Algorithm) ซึ่งเป็นชุดโปรแกรมที่ถูกออกแบบมาเพื่อให้ทำงานที่ซับซ้อนได้เอง เช่น การเรียงลำดับชุดตัวเลขจากน้อยไปมาก เป็นต้น แม้ว่าการใช้แนวคิดแบบ สเตต-แมคชีน จะสามารถทำงานได้เช่นเดียวกัน แต่การออกแบบนั้นกลับซับซ้อนมากเกินไปจนความจำเป็น และยังตรวจสอบได้ยากอีกด้วย

การเขียนโปรแกรมพื้นฐาน 101

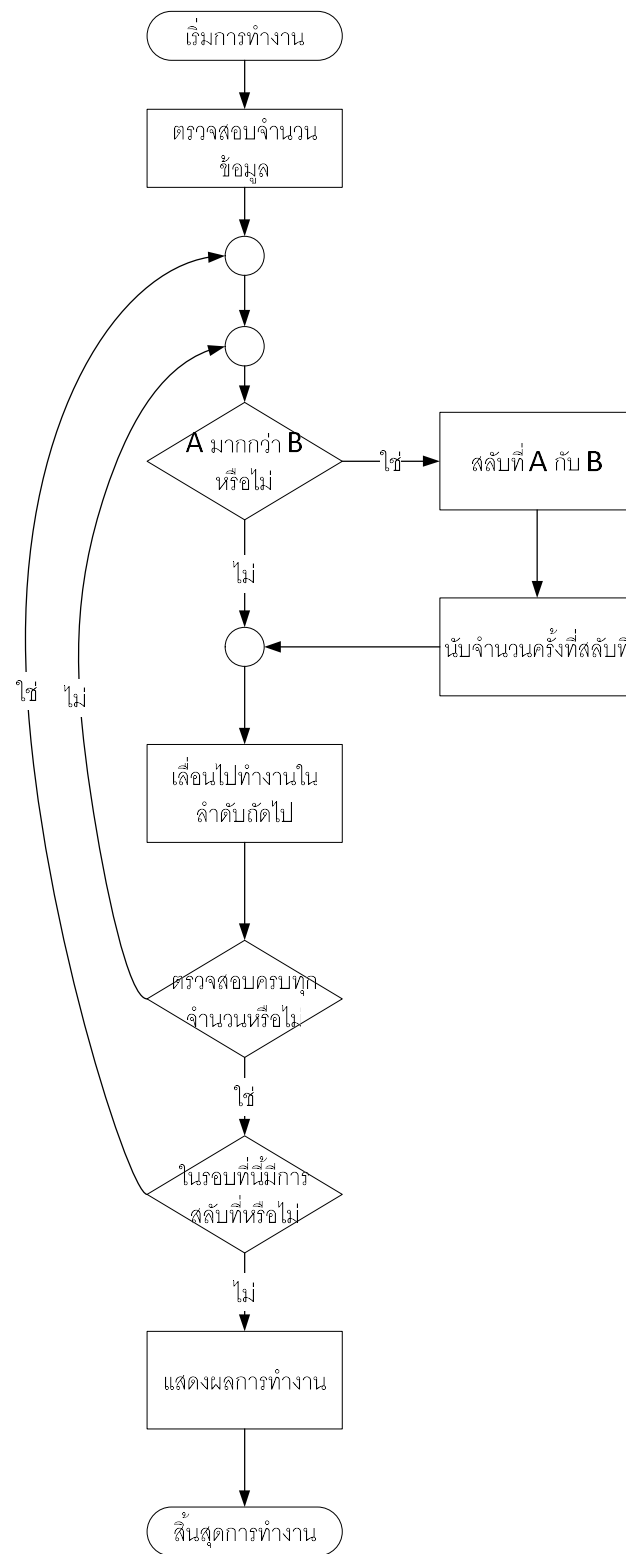
ตัวอย่างการออกแบบโปรแกรมเรียงลำดับตัวเลขจากน้อยไปมาก

สเตต-แมคชีน ในการเรียงลำดับตัวเลขจากน้อยไปมาก



การเขียนโปรแกรมพื้นฐาน 101

โฟลว์ชาร์ตในการเรียงลำดับตัวเลขจากน้อยไปมาก



การเขียนโปรแกรมพื้นฐาน 101

โค้ดตัวอย่างการเรียงลำดับเลขจำนวนจากน้อยไปมาก

```
numberTest = [5,2,3,4,2,7]
l = len(numberTest) #ตรวจสอบจำนวนชุดข้อมูล
s = 1 #สร้างตัวแปรไว้เก็บจำนวนการสลับที่

while s != 0 : #ทำงานจนกว่าจะไม่มีการสลับที่
    s = 0 #รีเซ็ตนับการสลับที่ใหม่
    for c in range (0 , l-1): #วนตรวจสอบข้อมูลจนชุดสุดท้าย
        a = numberTest[c] #เก็บค่าชุดปัจจุบันไว้ที่ a
        b = numberTest[c+1] #เก็บค่าชุดถัดไว้ที่ b
        if a > b : #เปรียบเทียบค่า A กับ B
            #สลับค่า A กับ B ถ้า A มากกว่า B
            numberTest[c] = b
            numberTest[c+1] = a
            s = s + 1 #นับจำนวนการสลับที่
        else :
            continue #ถ้า A ไม่มากกว่า B ให้ทำในข้อมูลชุด
            #ถัดไป
    else : #ถ้าไม่มีสลับที่แล้ว
        print "Mission Completed"
        print numberTest #แสดงข้อมูลที่มีการเรียงลำดับแล้ว
```


การเขียนโปรแกรมพื้นฐาน 101

สรุปตัวอย่างคำสั่งต่างๆในภาษา Python

การประกาศตัวแปร ในรูปแบบต่างๆ

การแทรกคำอธิบาย(Comment)

ในกระบวนการพัฒนาโปรแกรมเมื่อมีการเขียนโค้ดที่ค่อนข้างยาว การพัฒนาโปรแกรมขนาดใหญ่ ที่มีการทำงานร่วมกันหลายคน หรือการพัฒนาโปรแกรมที่ซับซ้อนนั้น การเขียนคำอธิบายไว้ระหว่างบรรทัดของการเขียนโปรแกรม เพื่อช่วยบอกว่าโค้ดบรรทัดนี้นั้น มีไว้ทำอะไร ผลที่ได้จากบรรทัดนี้คืออะไร ซึ่งจะทำให้เมื่อมีการกลับมาดูอีกครั้ง หรือมีการวิเคราะห์โปรแกรมอีกครั้งหนึ่งทำได้ง่าย และมีความถูกต้องขึ้นมาก

ในภาษา Python นั้น การแทรกคำอธิบายนั้นสามารถทำได้โดยใช้คำสั่ง '#' ดังตัวอย่าง

```
print "Hello World1"  
#print "Hello World2"
```

➤ Hello World1

```
print "Hello World1" #print "Hello World2"
```

➤ Hello World1

ตัวแปรประเภทข้อความ (string)

ใช้ (' ') (" ") และ (""" """) เป็นการประกาศตัวแปร เพื่อใช้แทนข้อความที่เรากำหนดไว้ เช่น ใช้แทนประโยค Hello World

```
x = "Hello World"  
print x
```

➤ Hello World

```
x = 'Hello World'  
print x
```

➤ Hello World

```
x = """Hello World"""  
print x
```

➤ Hello World

ข้อแตกต่างระหว่าง (' ') (" ") กับ (""" """) นั่นคือ การแสดงข้อความหลายบรรทัด ดังเช่นตัวอย่าง

การเขียนโปรแกรมพื้นฐาน 101

```
x = """H
e
l
l
o"""
```

```
print x
```

```
➤ H
  e
  l
  l
  o
```

แต่ หากใช้ (“ ”) หรือ (‘ ’)

```
x = 'H
e
l
l
o'
```

หรือ

```
x = "H
e
l
l
o"
```

```
print x
```

- ระบบจะแจ้งว่า Error : EOL while scanning string literal หมายถึงว่า เราไม่ได้ปิดประโยคของข้อความอย่างถูกต้อง เพราะ (‘ ’) และ (“ ”) ไม่สามารถรองรับการแสดงผลหลายบรรทัดจากคำสั่ง print เพียงครั้งเดียว

```
x = 'Hello World!'
print x           # แสดงอักษรทั้งหมดของ x
print x[0]        # แสดงอักษรของ x ตั้งแต่ตัวอักษรตัวแรก
print x[2:5]       # แสดงอักษรของ x ตั้งแต่ตัวอักษรที่ 3 ไปจนถึงตัวอักษรที่ 6
print x[2:]        # แสดงตัวอักษรของ x ตั้งแต่ลำดับ 3
print x * 2        # แสดงตัวอักษรทุกตัวของ x 2 ครั้ง
print x + "TEST"   # แสดงอักษรทั้งหมดของ x และต่อด้วย "TEST"
print x[5],x[3],x[1]+ "-" +x[6],x[4],x[2],x[0] #แสดงอักษรของ x เรียงลำดับ 5-3-1-6-4-2-0
```

- Hello World!
➤ H
➤ Llo
➤ llo World!
➤ Hello World!Hello World!

การเขียนโปรแกรมพื้นฐาน 101

- Hello World!TEST
- I e-W o l H

ตัวแปรประเภทจำนวนเต็มและจำนวนเต็มขนาดยาว (Integer , Long)

เป็นการประกาศตัวแปรที่ใช้สำหรับเก็บข้อมูลตัวเลขที่เป็นจำนวนเต็ม เช่น

```
x = 5 #กำหนดให้ x มีค่าเท่ากับจำนวนเต็ม 5
y = 4L #กำหนดให้ y มีค่าเท่ากับจำนวนเต็มขนาดยาว 4
print x+y
print x-y
print y-x
print x*y
print x/y
print y/x
print x%y
print y%x
```

- 9
- 1
- -1
- 20
- 1
- 0
- 1
- 4

ข้อแตกต่างระหว่างตัวแปรแบบจำนวนเต็ม กับตัวแปรจำนวนเต็มขนาดยาวคือ เลขจำนวนสูงสุดที่มันรองรับได้ โดยเราสามารถดูจำนวนที่ตัวแปรเลขจำนวนเต็มรองรับได้สูงสุดจากคำสั่ง sys.maxint ดังตัวอย่าง

```
sys.maxint
2147483647
```

และถ้าหากจำนวนที่เราใช้มีขนาดใหญ่กว่านี้ เช่น sys.maxint + 1 ระบบก็จะเปลี่ยนชนิดของตัวแปรไปเป็นจำนวนเต็มขนาดยาวโดยอัตโนมัติ เช่น

```
a = sys.maxint
print a , type(a)

a = a + 1
print a , type(a)

2147483647 <type 'int'>
2147483648 <type 'long'>
```

ดังนั้นแล้วเราจึงอาจจะต้องกังวลเรื่องขนาดของตัวแปรเลย เนื่องจากระบบจะจัดการให้เราทุกอย่าง

การเขียนโปรแกรมพื้นฐาน 101

ตัวแปรประเภทจำนวนทศนิยม (Float)

การประกาศตัวแปรที่ใช้สำหรับเก็บข้อมูลตัวเลขที่เป็นจำนวนทศนิยม เช่น

```
x = 5.0 # x เป็นตัวแปรประเภททศนิยม
y = 4 # y เป็นตัวแปรจำนวนเต็ม
print x+y
print x-y
print y-x
print x*y
print x/y
print y/x
print x%y
print y%x
print y , type(y)
y = x/y
print y , type(y)
```

- 9.0
- 1.0
- -1.0
- 20.0
- 1.25
- 0.8
- 1.0
- 4.0
- 4 <type 'int'>
- 1.25 <type 'float'>

ตัวแปรเลขจำนวนเชิงซ้อน (Complex)

การประกาศตัวแปรที่เป็นจำนวนเชิงซ้อน ในรูปแบบ a+bj (ซึ่ง j นั้นแทนจำนวนของ $\sqrt{-1}$ นั่นคือจำนวนที่เราเรียกว่า จำนวนจินตภาพ) a คือส่วนหนึ่งของจำนวนจริง และ b คือส่วนหนึ่งของจำนวนจินตภาพ

```
x = (5+1.25j) # x เป็นตัวแปรประเภทจำนวนเชิงซ้อน
y = 4 # y เป็นตัวแปรจำนวนเต็ม
```

```
print x , type(x)
print x.real,type(x.real)
print x.imag,type(x.imag)
print "x+y = ", x+y
print "x-y = ", x-y
print "y-x = ", y-x
print "x*y = ", x*y
print "x/y = ", x/y
print "y/x = ", y/x
print "x%y = ", x%y
print "y%x = ", y%x
```

การเขียนโปรแกรมพื้นฐาน 101

- $(5+1.25j)$ <type 'complex'>
- 5.0 <type 'float'>
- 1.25 <type 'float'>
- $x+y = (9+1.25j)$
- $x-y = (1+1.25j)$
- $y-x = (-1-1.25j)$
- $x*y = (20+5j)$
- $x/y = (1.25+0.3125j)$
- $y/x = (0.752941176471-0.188235294118j)$
- $x\%y = (1+1.25j)$
- $y\%x = (4+0j)$

ผลลัพธ์ที่ได้จากการหารเอาเศษ และการหารเอาจำนวนเต็ม อาจจะทำให้ผลที่ผิดพลาดได้ จำเป็นต้องระวังในการใช้คำสั่ง '%' และ '/'

ตัวแปรประเภทชุดลำดับ(Lists)

ตัวแปรแบบชุดลำดับคือ ตัวแปรที่สามารถเก็บข้อมูล หลายๆรูปแบบ และหลายๆข้อมูล เรียงกันเป็นลำดับ และมีกระบวนการที่สามารถทำให้เราเข้าถึงข้อมูลชุดนั้นๆได้ การประกาศตัวแปรแบบชุดลำดับสามารถทำได้ โดยการตั้งชื่อตัวแปร ในตัวอย่างนี้คือ x และเรากำหนดให้เท่ากับกลุ่มของค่าต่างๆ จำนวนหนึ่ง เช่น 'abcd' 786 2.23 'john' 70.2 ในที่นี้มีทั้งหมด 5 จำนวน เราจะสามารถกำหนดได้ดังนี้ $x = ['abcd', 786 , 2.23, 'john', 70.2]$

โดยทุกๆจำนวน จะต้องอยู่เรียงกันในกรอบสัญลักษณ์ “[]” และแต่ละค่านั้นจะแบ่งแยกกันด้วยสัญลักษณ์ “,” เช่น

```
x = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
y = [123, 'john']
```

x จะมีค่าตามตารางข้างล่างนี้

X[0]	X[1]	X[2]	X[3]	X[4]
abcd	786	2.23	john	70.2

y จะมีค่าตามตารางข้างล่างนี้

y[0]	y[1]
123	john

```
print x , y          # แสดงข้อมูลทั้งหมดของ x และ y
print x[0]           # แสดงเฉพาะข้อมูลชุดแรกของ x
```

การเขียนโปรแกรมพื้นฐาน 101

```
print x[1:3]          # แสดงข้อมูลของ x ตั้งแต่ชุดที่ 2 ไปจนถึงชุดที่ 3
print x[2:]           # แสดงข้อมูลของ x ตั้งแต่ชุดที่ 3
print y * 2           # แสดงข้อมูลทั้งหมดของ y 2 ครั้ง
print x + y           # แสดงชุดข้อมูลทั้งหมดของ x รวมกับ y
x = x + ['1000']       # เพิ่มข้อมูล '1000' เข้าไปในชุดข้อมูล x
print x               # แสดงข้อมูลทั้งหมดของ x ซึ่งจะมีข้อมูลใหม่ที่เพิ่มเข้ามาอยู่ด้วย
```

- ['abcd', 786, 2.23, 'john', 70.2] [123, 'john']
- abcd
- [786, 2.23]
- [2.23, 'john', 70.2]
- [123, 'john', 123, 'john']
- ['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
- ['abcd', 786, 2.23, 'john', 70.2, '1000']

ประเภทตัวแปรประเภท Tuples

Tuples เป็นตัวแปรลำดับขั้นอีกแบบหนึ่ง ซึ่งแตกต่างจากตัวแปรลำดับขั้นแบบ Lists ที่ได้กล่าวถึงในหัวข้อที่แล้ว ในส่วนที่ Tuples จะไม่อนุญาตให้มีการเขียนข้อมูลทับข้อมูลเดิม หรือเปลี่ยนแปลงข้อมูลจากภายนอกได้ นี่คือการแตกต่างที่สำคัญของตัวแปรประเภท Tuples

การประกาศตัวแปร Tuples สามารถทำได้ โดยการตั้งชื่อตัวแปร ในตัวอย่างนี้คือ x และเรากำหนดให้เท่ากับกลุ่มของค่าต่างๆ จำนวนหนึ่ง เช่น 'abcd' 786 2.23 'john' 70.2 ในที่นี้มีทั้งหมด 5 จำนวน เราจะสามารถกำหนดได้ดังนี้ x = ('abcd', 786 , 2.23, 'john', 70.2)

โดยทุกๆจำนวน จะต้องอยู่เรียงกันในกรอบสัญลักษณ์ “()” และแต่ละค่านั้นจะแบ่งแยกกันด้วยสัญลักษณ์ “,” เช่น

```
x = ( 'abcd', 786 , 2.23, 'john', 70.2 )
y = (123, 'john')
```

```
print x,y           # แสดงข้อมูลทั้งหมดของ x และ y
print x[0]          # แสดงข้อมูลชุดแรกของ x
print x[1:3]        # แสดงข้อมูลของ x ตั้งแต่ชุดที่ 2 ไปจนถึงชุดที่ 3
print x[2:]         # แสดงข้อมูลของ x ตั้งแต่ชุดที่ 3 ไปจนจบ
print y * 2         # แสดงข้อมูลของ y 2 ครั้ง
print x + y         # แสดงข้อมูล x รวมกับ y
```

- ('abcd', 786, 2.23, 'john', 70.2) (123, 'john')
- abcd
- (786, 2.23)
- (2.23, 'john', 70.2)
- (123, 'john', 123, 'john')

การเขียนโปรแกรมพื้นฐาน 101

➤ ('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

แต่หากเราจะเปลี่ยนค่าใน ตัวแปร x เราจะมีข้อผิดพลาด Error ให้เราทราบ โดยบอกว่า TypeError: 'tuple' object does not support item assignment ซึ่งหมายถึง Tuples นั้นไม่รองรับการกำหนดค่าซ้ำหรือแก้ไขข้อมูลใหม่ เช่น

```
x = ('abcd', 786, 2.23, 'john', 70.2) #กำหนดค่า x
y = ['abcd', 786, 2.23, 'john', 70.2] #กำหนดค่า y
x[1] = 1000 #เปลี่ยนค่า x ชุดที่ 2 และคำสั่งในบรรทัดนี้ จะเกิดปัญหา TypeError: 'tuple' object does
not support item assignment
y[1] = 1000 #คำสั่งนี้จะเปลี่ยนค่าของ y ชุดที่ 2 ให้เท่ากับ 1000 โดยไม่มีการแจ้งข้อผิดพลาด เนื่องจาก y เป็นตัวแปรประเภท
Lists ซึ่งยินยอมให้มีการเขียนข้อมูลทับข้อมูลเดิมได้
```

ประเภทตัวแปรประเภท Dictionary

ตัวแปรประเภทนี้ ชุดข้อมูลจะถูกเก็บในลักษณะคู่อันดับแบบ คีย์ กับข้อมูล ที่ไม่เรียงลำดับ ซึ่งต่างจากตัวแปรชุดแบบ Lists และ Tuples ที่มีลำดับข้อมูลเรียงกัน ดังนั้นแล้วเราจะไม่สามารถเข้าถึงข้อมูลด้วยวิธีเดียวกันกับที่ใช้ใน Tuples หรือ Lists ได้ เราจะต้องเข้าถึงข้อมูลโดยผ่าน คีย์ เท่านั้น

การประกาศตัวแปรประเภทดิกชันนารีนี้ เราสามารถทำได้โดย โดยการตั้งชื่อตัวแปร ในตัวอย่างนี้คือ x และเรากำหนดให้เท่ากับกลุ่มของค่าต่างๆ ที่ประกอบไปด้วยคีย์ และข้อมูล จำนวนหนึ่ง ในรูปแบบ คีย์ : ข้อมูล เช่น 'key': 'data' , 'name': 'abcd' , 1 : 786 , 1.5 : 2.23 , 2 : 'john', 'test': 70.2 ในที่นี่มีทั้งหมด 6 จำนวน เราจะสามารถกำหนดได้ดังนี้ x = {'key': 'data' , 'name': 'abcd' , 1 : 786 , 1.5 : 2.23 , 2 : 'john', 'test': 70.2}

โดยทุกๆจำนวน จะต้องอยู่เรียงกันในกรอบสัญลักษณ์ “{ }” มีสัญลักษณ์ : คั่นแยกระหว่าง คีย์ กับข้อมูล และแต่ละค่านั้นจะแบ่งแยกกันด้วยสัญลักษณ์ “,” เช่น

```
a = {}
x = {'key': 'data' , 'name' : 'abcd' , 1 : 786 , 1.5 : 2.23 , 2 :
'john', 'test': 70.2}
y = ('y1', 'y2', 1, 2, 3)
```

```
print x          # Prints complete dictionary
print x.keys()   # Prints all the keys
print x.values() # Prints all the values

➤ {1.5: 2.23, 1: 786, 2: 'john', 'name': 'abcd', 'key': 'data', 'test': 70.2}
➤ [1.5, 1, 2, 'name', 'key', 'test']
➤ [2.23, 786, 'john', 'abcd', 'data', 70.2]
```

```
x['t'] = 0.5      #เพิ่ม 't' เป็นคีย์ใหม่ โดยให้มีข้อมูลเป็น 5
print x          # Prints complete dictionary
print x.keys()   # Prints all the keys
```

การเขียนโปรแกรมพื้นฐาน 101

print x.values() # Prints all the values

- {1.5: 2.23, 1: 786, 2: 'john', 'name': 'abcd', 't': 0.5, 'key': 'data', 'test': 70.2}
- [1.5, 1, 2, 'name', 't', 'key', 'test']
- [2.23, 786, 'john', 'abcd', 0.5, 'data', 70.2]

x['test'] = 25 # เปลี่ยนข้อมูลของคีย์ 'test' ให้เท่ากับ 25

print x # Prints complete dictionary

print x.keys() # Prints all the keys

print x.values() # Prints all the values

- {1.5: 2.23, 1: 786, 2: 'john', 'name': 'abcd', 't': 0.5, 'key': 'data', 'test': 25}
- [1.5, 1, 2, 'name', 't', 'key', 'test']
- [2.23, 786, 'john', 'abcd', 0.5, 'data', 25]

a[y] = "Hello" # เพิ่มคีย์ ชื่อ y แบบ Tuples โดยให้มีค่าเท่ากับ "Hello"

print a # Prints complete dictionary

print a.keys() # Prints all the keys

print a.values() # Prints all the values

- {'y1', 'y2', 1, 2, 3}: 'Hello'}
- [['y1', 'y2', 1, 2, 3]]
- ['Hello']

a["H"] = y # เพิ่มคีย์ "H" ให้มีค่าเท่ากับ Tuples ของ y

print a # Prints complete dictionary

print a.keys() # Prints all the keys

print a.values() # Prints all the values

- {'H': ('y1', 'y2', 1, 2, 3), ('y1', 'y2', 1, 2, 3): 'Hello'}
- ['H', ('y1', 'y2', 1, 2, 3)]
- [['y1', 'y2', 1, 2, 3], 'Hello']

ตัวกระทำทางการคำนวณ

เป็นเครื่องหมายสำหรับใช้ทำงานทางการคำนวณ หรือเปรียบเทียบระหว่างข้อมูล เพื่อใช้ในการตัดสินใจ ในกระบวนการทำงาน

เครื่องหมาย	ความหมาย	ตัวอย่าง
+	การบวก – นำสองจำนวนมาบวกกัน	a = 10 b = 15 c = a + b c = 25

การเขียนโปรแกรมพื้นฐาน 101

-	การลบ – นำจำนวนแรกลบด้วยจำนวนที่สอง	$a = 10$ $b = 15$ $c = a - b$ $c = -5$
*	การคูณ – นำสองจำนวนคูณกัน	$a = 10$ $b = 15$ $c = a * b$ $c = 150$
/	การหาร – จำนวนแรกหารด้วยจำนวนที่สอง	$a = 20$ $b = 15$ $c = a / b$ $c = 1$ $a = 20.0$ $b = 15$ $c = a / b$ $c = 1.33..$
%	การหารเอาเศษ – คำตอบที่ได้จากการหาร คือเศษเหลือจากการหารนั้น	$a = 20$ $b = 15$ $c = a \% b$ $c = 5$
**	ยกกำลัง – เลขจำนวนแรก ยกกำลังด้วยเลขจำนวนที่สอง	$a = 20$ $b = 15$ $c = a ** b$ $c = 3.2768e+19$ หรือเท่ากับ 3.2768×10^{19}
//	การหารเอาเลขจำนวนเต็ม – เมื่อให้การหารแบบนี้ คำตอบที่ได้จะเป็นเลขจำนวนเต็มเท่านั้น	$a = 20$ $b = 15$ $c = a // b$ $c = 1$

การเขียนโปรแกรมพื้นฐาน 101

		a = 20.0 b = 15 c = a // b c = 1.0
==	เปรียบเทียบค่าทั้งสองจำนวนว่าเท่ากันหรือไม่ – คำตอบที่ได้มีเพียง True หากเท่ากัน และ False หากไม่เท่า	a = 20.0 b = 15 (a == b) is not true.
!=	เปรียบเทียบค่าทั้งสองจำนวนว่าเท่ากันหรือไม่ – คำตอบที่ได้มีเพียง True หากไม่เท่ากัน และ False หากเท่ากัน	a = 20.0 b = 15 (a != b) is true.
<>	เปรียบเทียบค่าทั้งสองจำนวนว่ามากกว่า หรือน้อยกว่า หรือไม่ – คำตอบที่ได้มีเพียง True หากมากกว่า หรือน้อยกว่ากัน และ False หากเท่ากัน จะเหมือน !=	a = 20.0 b = 15 (a <> b) is true.
>	เครื่องหมายมากกว่า – ตรวจสอบจำนวนทางซ้ายว่ามากกว่าทางขวาหรือไม่ จะให้คำตอบเป็น True เมื่อทางซ้ายมากกว่าทางขวา	a = 20.0 b = 15 (a > b) is true. a = 20.0 b = 15 (b > a) is false.
<	เครื่องหมายน้อยกว่า – ตรวจสอบจำนวนทางซ้ายว่าน้อยกว่าทางขวาหรือไม่ จะให้คำตอบเป็น True เมื่อทางซ้ายน้อยกว่าทางขวา	a = 20.0 b = 15 (a < b) is false. a = 20.0 b = 15 (b < a) is true.
>=	เครื่องหมายเท่ากับ หรือมากกว่า – ตรวจสอบจำนวนทางซ้ายว่าเท่ากับ หรือมากกว่าทางขวาหรือไม่ จะให้คำตอบเป็น True เมื่อทางซ้ายเท่ากับ หรือมากกว่าทางขวา	a = 20.0 b = 15

การเขียนโปรแกรมพื้นฐาน 101

		<p>(a >= b) is true.</p> <p>a = 15.0 b = 15</p> <p>(b >= a) is true.</p>
<=	เครื่องหมายเท่ากับ หรือน้อยกว่า – ตรวจสอบจำนวนทางซ้ายว่าเท่ากับ หรือน้อยกว่าทางขวา หรือไม่ จะให้คำตอบเป็น True เมื่อทางซ้ายเท่ากับ หรือน้อยกว่าทางขวา	<p>a = 20.0 b = 15</p> <p>(a <= b) is false.</p> <p>a = 15.0 b = 15</p> <p>(b <= a) is true.</p>
=	เครื่องหมายกำหนดค่า – ใช้สำหรับ กำหนดค่าทางซ้าย ให้เท่ากับค่าทางขวา	<p>c = a + b</p> <p>กำหนดให้ c เท่ากับ a + b</p>
+=	เครื่องหมายเพิ่มค่า – ใช้สำหรับ เพิ่มค่าทางซ้าย ด้วยค่าทางขวา	<p>c += a</p> <p>จะหมายถึง c = c + a</p>
-=	เครื่องหมายลดค่า – ใช้สำหรับ ลดค่าทางซ้าย ด้วยค่าทางขวา	<p>c -= a</p> <p>จะหมายถึง c = c - a</p>
*=	เครื่องหมายคูณค่า – ใช้สำหรับ คูณค่าทางซ้าย ด้วยค่าทางขวา	<p>c *= a</p> <p>จะหมายถึง c = c * a</p>
/=	เครื่องหมายหารค่า – ใช้สำหรับ หารค่าทางซ้าย ด้วยค่าทางขวา	<p>c /= a</p> <p>จะหมายถึง c = c / a</p>
%=	เครื่องหมายหารค่าแบบเอาเศษ – ใช้สำหรับ หารค่าแบบเอาเศษค่าทางซ้าย ด้วยค่าทางขวา	<p>c %= a</p> <p>จะหมายถึง c = c % a</p>
**=	เครื่องหมายยกกำลัง – ใช้สำหรับ ยกกำลังค่าทางซ้าย ด้วยค่าทางขวา	<p>c **= a</p>

การเขียนโปรแกรมพื้นฐาน 101

		จะหมายถึง $c = c ** a$
//=	เครื่องหมายหารเอาจำนวนเต็ม- ใช้สำหรับ หารเอาจำนวนเต็มค่าทางซ้าย ด้วยค่าทางขวา	$c //= a$ จะหมายถึง $c = c // a$
&	เครื่องหมาย “และ” หมายถึงเอาค่าในแบบเลขฐานสองของทั้งสองฝั่งมา “และ” กัน	$a = 20$ $b = 15$ $c = a \& b$ $a = 20$ 0b10100 $b = 15$ 0b1111 $c = 4$ 0b100
	เครื่องหมาย “หรือ” หมายถึงเอาค่าในแบบเลขฐานสองของทั้งสองฝั่งมา “หรือ” กัน	$a = 20$ $b = 15$ $c = a b$ $a = 20$ 0b10100 $b = 15$ 0b1111 $c = 31$ 0b11111
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	$a = 20$ $b = 15$ $c = a \wedge b$ $a = 20$ 0b10100 $b = 15$ 0b1111 $c = 27$ 0b11011
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	$a = 20$ $c = \sim a$ $a = 20$ 0b10100 $c = -21$ -0b10101
<<	เครื่องหมายเลื่อนบิตไปทางซ้าย – โดยจะเลื่อนบิตข้อมูลไปทางซ้ายตามจำนวนที่กำหนดทางขวา	$a = 20$ $c = a << 2$ $a = 20$ 0b10100 $c = 80$ 0b1010000
>>	เครื่องหมายเลื่อนบิตไปทางขวา – โดยจะเลื่อนบิตข้อมูลไปทางขวาตามจำนวนที่กำหนดทางขวา	$a = 20$ $c = a >> 2$ $a = 20$ 0b10100

การเขียนโปรแกรมพื้นฐาน 101

		c = 5 0b101
and	<p>เปรียบเทียบ “และ” ทางตรรกศาสตร์</p> <p>คำตอบที่ได้จะมีเพียง True หากทั้งสองข้างมีค่าเป็น True</p> <p>และ False หากมีมากกว่า 1 ค่าที่เป็น False</p>	<p>a = 0</p> <p>b = 1</p> <p>(a and b) is false.</p> <p>a = 1</p> <p>b = 1</p> <p>(a and b) is true.</p>
or	<p>เปรียบเทียบ “หรือ” ทางตรรกศาสตร์</p> <p>คำตอบที่ได้จะมีเพียง False หากทั้งสองข้างมีค่าเป็น False</p> <p>และ True หากมีมากกว่า 1 ค่าที่เป็น True</p>	<p>a = 0</p> <p>b = 1</p> <p>(a or b) is true.</p> <p>a = 0</p> <p>b = 0</p> <p>(a or b) is false.</p>
not	<p>เปรียบเทียบ “ไม่” ทางตรรกศาสตร์</p> <p>คำตอบที่ได้จะมีเพียง False หากค่าเป็น True</p> <p>และ True หากค่าเป็น False</p>	<p>a = 0</p> <p>b = 1</p> <p>not(a and b) is true.</p> <p>a = 1</p> <p>b = 1</p> <p>not(a and b) is false.</p>
in	จะให้คำตอบเป็น true หากค่าทางขวานั้นเป็นสมาชิกของชุดข้อมูลทางซ้าย	<p>a = 20.0</p> <p>b = [0,2,20,3]</p> <p>c = a in b</p> <p>C = true</p> <p>a = 20.0</p> <p>b = [0,2,30,3]</p> <p>c = a in b</p> <p>c = false</p>
not in	จะให้คำตอบเป็น false หากค่าทางขวานั้นเป็นสมาชิกของชุดข้อมูลทางซ้าย	<p>a = 20.0</p> <p>b = [0,2,20,3]</p> <p>c = a not in b</p> <p>C = false</p> <p>a = 20.0</p> <p>b = [0,2,30,3]</p> <p>c = a not in b</p>

การเขียนโปรแกรมพื้นฐาน 101

		c = true
is	จะให้คำตอบเป็น true หากค่าทางขวานั้นเป็นค่าเดียวกันทางซ้าย	a = 20.0 b = [15, 20.0] c = a is b[1] C = true a = 20.0 b = [15, 20] c = a is b[1] c = false
is not	จะให้คำตอบเป็น true หากค่าทางขวานั้นไม่เป็นค่าเดียวกันทางซ้าย	a = 20.0 b = [15, 20.0] c = a is not b[1] C = false a = 20.0 b = [15, 20] c = a is not b[1] c = true

ลำดับความสำคัญของตัวกระทำทางกรคำนวณ

เครื่องหมาย	ความหมาย	ตัวอย่าง
***	การยกกำลัง	a = 3 b = 5 c = b+a**2 c = 14
~, '+', '-'	~ – การกลับบิตข้อมูล + – เครื่องหมายของจำนวนบวก - – เครื่องหมายของจำนวนลบ	a = 3 b = ~a c = -a d = +a e = a + ~a f = a + -a g = a + +a a = 3, 0b11 b = -4, -0b100 c = -3, -0b11 d = 3, 0b11 e = -1, -0b1 f = 0, 0b0 g = 6, 0b110

การเขียนโปรแกรมพื้นฐาน 101

'*', '/', '%', '/'	การคูณ และหาร ในแบบต่างๆ	<pre> a = 3 b = 5 c = b+a*2 c = 11 a = 3 b = 5 c = b+a/2 c = 6 </pre>
'+', '-'	การบวกและลบ	<pre> a = 3 b = 5 c = b<<4+a c = 640 0b1010000000 a = 3 b = 5 c = b>>4-a c = 2 0b10 </pre>
'>>', '<<'	การเลื่อนบิตข้อมูล	<pre> a = 2 b = 14 c = b&13>>a c = 2 0b10 a = 2 b = 14 c = b&13<<a c = 4 0b100 </pre>
'&'	การ “และ” บิต ข้อมูล	<pre> a = 2 b = 14 c = 13 d = 18 e = a ^ b c & d f = a b ^ c & d g = a b & c ^ d e = 12 0b1100 f = 14 0b1110 g = 30 0b11110 </pre>
' ', '^'	การ “หรือ” และการ “หรือ แบบพิเศษ”	<pre> a = 2 b = 14 c = 13 d = 18 e = a ^ b c & d > a b ^ c & d f = a ^ b c & d < a b ^ c & d e = False f = True </pre>

การเขียนโปรแกรมพื้นฐาน 101

'<>', '==', '!='	การเปรียบเทียบที่เท่ากัน หรือไม่เท่ากัน	<pre> a = 2 b = 14 c = 13 d = 18 e = b > c == c f = b < c == c g = b <= b == b e = True f = False g = True </pre>
'<=', '<', '>', '>='	การเปรียบเทียบที่มากกว่า และ น้อยกว่า	<pre> a = 2 b = 14 c = 13 d = 18 e = 14 > a b ^ c & d f = 14 < a b ^ c & d g = 14 >= a b ^ c & d e = False f = False g = True </pre>
'=', '%=', '/=', '//=', '-=', '+=', ' =', '&=', '>>=', '<<=', '*=', '**='	การกำหนดค่า	<pre> a = 2 b = 14 c = 10 c += b & 13 >> a c = 12 0b1100 a = 2 b = 14 c = 10 c >>= b & 13 >> a c = 2 0b10 </pre>
'is', 'is not'	การตรวจสอบตัวแปร	<pre> a = 2 b = 14 c = a is b & 13 >> a d = b is b & 13 >> a e = a is not b & 13 >> a f = b is not b & 13 >> a c = True d = False e = False f = True </pre>

การเขียนโปรแกรมพื้นฐาน 101

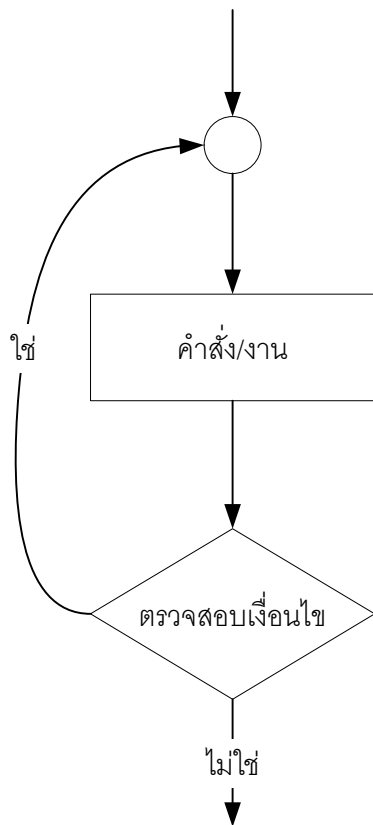
'in' , 'not in'	การตรวจสอบสมาชิกข้อมูล	<pre> a = 2 b = 14 c = ('123','abc',456,a) d = ['456','def',123] e = a in c f = b in d + [b] g = b not in d + [b] e = True f = True g = False </pre>
'not' , 'or' , 'and'	การตรวจสอบเงื่อนไข	<pre> a = 2 b = 14 c = ('123','abc',456,a) d = ['456','def',123] e = not a in c f = b in d + [b] and a in c g = not b in d + [b] and a in c h = b not in d + [b] or a in c i = not b not in d + [b] or a in c j = b not in d + [b] or not a in c e = False f = True g = False h = True i = True j = False </pre>

คำสั่งสำคัญในการทำงาน

คำสั่งการวนซ้ำ, วนลูป (Loop)

เป็นคำสั่งที่ใช้เพื่อให้โปรแกรมมีการทำงานซ้ำในช่วงคำสั่งการทำงานเดิมตามเงื่อนไขที่เราได้กำหนดไว้ ที่เรียกว่าการทำงานแบบวนลูป เช่น การสั่งให้โปรแกรมบวกเลขไปเรื่อยๆ ตราบใดที่ค่านั้นยังน้อยกว่า 100 เป็นต้น โดยเงื่อนไขในการวนลูปนั้น เงื่อนไขนั้นจะต้องเป็น จริง – ถูกต้อง – ใช่ เสมอ และการวนลูปนั้นจะสิ้นสุดลงเมื่อ เงื่อนไขนั้นมีผลเป็นเท็จ – ผิด – ไม่ใช่ ซึ่งคำสั่งนั้น มี 2 คำสั่งคือ while และ for

การเขียนโปรแกรมพื้นฐาน 101



While – while <เงื่อนไข> :

 คำสั่ง 1

 คำสั่ง 2

 คำสั่ง 3

 คำสั่ง 4

คำสั่ง while จะทำให้โปรแกรมทำงานวนลูปอยู่ในช่วงคำสั่งที่ 1 – คำสั่งที่ 3 ตราบใดที่ เงื่อนไขของ คำสั่ง while ยังคงเป็นจริง และหากเงื่อนไขของคำสั่ง while เป็นเท็จ โปรแกรมจึงจะมาทำคำสั่งที่ 4 ต่อไป

ตัวอย่างการใช้คำสั่ง while

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

การเขียนโปรแกรมพื้นฐาน 101

ผลการทำงาน

The count is: 0

The count is: 1

The count is: 2

The count is: 3

The count is: 4

The count is: 5

The count is: 6

The count is: 7

The count is: 8

Good bye!

ตัวอย่างนี้ จะเห็นว่าเงื่อนไขของการวนลูปคือ count จะต้อง “น้อยกว่า” 9 หาก count น้อยกว่า 9 แล้ว โปรแกรมจะแสดงข้อความว่า “The count is : ” ตามด้วยค่าของ count และมีการเพิ่มค่าของ count ทีละ 1 ในทุกๆรอบที่มีการวนมาทำงาน

เมื่อ count นั้นมีการเพิ่มค่าจนเท่ากับ 9 แล้ว จะทำให้เงื่อนไขของลูป while เป็นเท็จ ที่ระบุว่า count ต้อง น้อยกว่า 9 จึงจะมีการวนลูปในช่วงของคำสั่งชุดนี้ เมื่อการวนลูปสิ้นสุดลง โปรแกรมก็จะออกจากลูปนั้น และ มาทำงานในคำสั่ง print "Good bye!" ที่อยู่นอกวงลูปต่อไป

ลูปที่ไม่มีที่สิ้นสุด – Infinity Loop คือลูปที่เงื่อนไขนั้น ไม่มีทางเป็นเท็จได้ ดังนั้นโปรแกรมจะทำงานวนในลูป นั้นอยู่ตลอด โดยไม่มีการออกมาทำงานนอกลูป ซึ่งรูปแบบนี้บ่อยครั้งที่มีถูกออกแบบเพื่อให้โปรแกรมทำงาน อยู่ตลอดเวลาที่มีการทำงาน แต่บ่อยครั้งที่การเกิดรูปแบบไม่มีที่สิ้นสุดนี้ เกิดจากความผิดพลาดในการ ออกแบบเงื่อนไขในการทำงาน ทำให้โปรแกรมค้างอยู่ในลูปนั้น

ตัวอย่าง ลูปที่ไม่มีที่สิ้นสุด – Infinity Loop

```
var = 1
while var == 1 : # This constructs an infinite loop
    num = input("Enter a number :")
    print "You entered: ", num

print "Good bye!"
```

การเขียนโปรแกรมพื้นฐาน 101

ผลการทำงาน

```
Enter a number :1
You entered: 1
Enter a number :2
You entered: 2
Enter a number :5
You entered: 5
Enter a number :8
You entered: 8
Enter a number :4
You entered: 4
Enter a number :
```

จะเห็นว่าโปรแกรมนี้นับรับค่าที่เราป้อนเข้าไปอยู่ตลอดเวลา เนื่องจากว่าเงื่อนไข `var == 1` นั้น เป็นจริงเสมอ เพราะคำสั่งในลูปนั้น ไม่ได้เข้าไปเปลี่ยนแปลงค่าใดๆของ `var` ทำให้เงื่อนไข `var == 1` ไม่มีการเปลี่ยนแปลงไปด้วยเช่นกัน ดังนั้นมันจึงทำงานวนอยู่แบบนี้ตลอดไป

คำสั่งการวนลูปแบบ `while` มีการรองรับคำสั่ง `else` เพื่อให้ทำงานบนเงื่อนไขที่เป็นเท็จ โดยเฉพาะ เช่น

```
count = 0
while count < 5:
    print count, " is less than 5"
    count = count + 1
else :
    count = count - 1
    print count, " is not less than 5"
print "end for ",count
```

หาก `count` เริ่มต้นที่ 0 ผลที่ได้ก็จะเป็น

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
4 is not less than 5
end for 0
```

หาก `count` เริ่มต้นที่ 6 ผลที่ได้ก็จะเป็น

```
count = 6
while count < 5:
    print count, " is less than 5"
    count = count + 1
else :
```

การเขียนโปรแกรมพื้นฐาน 101

```
count = count - 1
print count, " is not less than 5"
print "end for ",count
```

ผลที่ได้

```
5 is not less than 5
end for 5
```

For – for <คำชี้แจง> :

คำสั่งที่ 1

คำสั่งที่ 2

คำสั่งที่ 3....

คำสั่งที่ 4

คำสั่งวนรูปแบบ for นี้จะแตกต่างจากคำสั่ง while ตรงที่ คำสั่ง for จะใช้ “คำชี้แจง” ในการกำหนดกรอบการวนรูปแบบ ดังนั้นเราจะไม่สามารถใช้ เงื่อนไขแบบ วนตราใดที่ค่ามันยังน้อยกว่า 9 ได้ แต่การวนรูปแบบการใช้ for นั้น จำนวนรอบของการวนจะถูกกำหนดอยู่ใน “คำชี้แจง” นั้นเลย เช่น วนนับตั้งแต่ 0 ไปจนถึง 9 ซึ่งจะมีระยะจำกัดอย่างชัดเจน โดยที่เราไม่สามารถแก้ไขคำชี้แจงนี้ได้ จากภายในลูปนั้น

ตัวอย่าง

```
for letter in 'Python':      # First Example
    print 'Current Letter :', letter

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:         # Second Example
    print 'Current fruit :', fruit

for count in range(10,20) :   # Second Example
    print 'Current count :', count

print "Good bye!"
```

ผลที่ได้

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
```

การเขียนโปรแกรมพื้นฐาน 101

```
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Current count : 10
Current count : 11
Current count : 12
Current count : 13
Current count : 14
Current count : 15
Current count : 16
Current count : 17
Current count : 18
Current count : 19
Good bye!
```

และถึงแม้ว่าเราจะเพิ่มโค้ดเข้าไปเพื่อควบคุมตัวแปร ไม่ให้เปลี่ยนแปลงระหว่างการวนลูปเช่น ทำให้ตัวแปร letter ให้มีค่าเท่ากับตัว t เสมอ fruit ให้มีค่าเท่ากับตัว apple เสมอ หรือ count ให้มีค่าเท่ากับ 15 เสมอ เพื่อไม่ให้โปรแกรมวนจนครบทุกรอบได้ ก็ไม่ส่งผลกระทบต่อการทำงานของโปรแกรม

```
for letter in 'Python':      # First Example
    print 'Current Letter :', letter
    letter = 't'

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:        # Second Example
    print 'Current fruit :', fruit
    fruit = 'apple'

for count in range(10,20) :   # Second Example
    print 'Current count :', count
    count = 15

print "Good bye!"
```

ผลที่ได้คือ

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
```

การเขียนโปรแกรมพื้นฐาน 101

Current fruit : banana

Current fruit : apple

Current fruit : mango

Current count : 10

Current count : 11

Current count : 12

Current count : 13

Current count : 14

Current count : 15

Current count : 16

Current count : 17

Current count : 18

Current count : 19

Good bye!

จะเห็นว่าไม่มีความแตกต่าง เพราะว่าตัวทำงานของโปรแกรม จะไม่ยอมให้เราเข้าถึง เพื่อแก้ไขค่าของตัวแปร ในส่วนของ “คำชี้แจง” ได้

คำสั่งการวนลูปแบบ for นั้น ก็มีการรองรับคำสั่ง else เช่นเดียวกับ while แต่สำหรับ for ลูปนั้น else จะทำงานเมื่อตัวแปรนั้นอยู่นอกช่วงการทำงาน ตามคำชี้แจงใน for ลูปนั้น เช่น

```
for num in range(10,20): #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0: #to determine the first factor
            j=num/i #to calculate the second factor
            print '%d equals %d * %d' % (num,i,j)
            break #to move to the next number, the #first FOR
    else: # else part of the loop
        print num, 'is a prime number'
```

ผลที่ได้

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
```

การเขียนโปรแกรมพื้นฐาน 101

```
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

จะเห็นว่า ค่าที่เป็น prime number (จำนวนเฉพาะ) นั้น คือค่าที่อยู่นอกการวนของ for ลูป ที่มีค่าชี้แจงว่า วนในช่วง 2 ไปจนถึงค่า num - 1 หากมีตัวเลขใดในการวนที่มีการหารแล้วลงตัว โปรแกรมจะหยุดวน และหลุดออกจากลูป for นี้ด้วยคำสั่ง break แต่เมื่อไม่มีเลขใดหารลงตัว ในช่วง 2 ไปจนถึงค่า num - 1 ค่าสุดท้ายของ i คือเท่ากับ num ก็จะทำให้เข้าเงื่อนไข else ของลูป for นี้ แล้วจึงแสดงผลว่านี่คือเลข prime number ตามคำสั่งที่อยู่ใน else นั้นเอง

ดังนั้นหากเราไม่ใส่คำสั่ง break ในการวน for ลูปนั้น การวนก็จะเกิดกับเลขทุกชุด ทุกตัว และเลขทุกตัวก็จะ เป็น prime number ไปด้วย

```
for num in range(10,20): #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0: #to determine the first factor
            j=num/i #to calculate the second factor
            print '%d equals %d * %d' % (num,i,j)
            #break #to move to the next number, the #first FOR
        else: # else part of the loop
            print num, 'is a prime number'
```

ผลลัพธ์ที่ได้

```
10 equals 2 * 5
10 equals 5 * 2
10 is a prime number
11 is a prime number
12 equals 2 * 6
12 equals 3 * 4
12 equals 4 * 3
12 equals 6 * 2
12 is a prime number
13 is a prime number
14 equals 2 * 7
14 equals 7 * 2
14 is a prime number
15 equals 3 * 5
15 equals 5 * 3
15 is a prime number
16 equals 2 * 8
16 equals 4 * 4
16 equals 8 * 2
16 is a prime number
17 is a prime number
18 equals 2 * 9
18 equals 3 * 6
18 equals 6 * 3
```


การเขียนโปรแกรมพื้นฐาน 101

```
18 equals 9 * 2
18 is a prime number
19 is a prime number
```

การซ้อนกันของลูป (nested loops)

ในบางเวลาที่เราต้องพัฒนาโปรแกรมที่มีความซับซ้อน มีการวนคำนวณค่าในกลุ่มจำนวนย่อย จากกลุ่มจำนวนใหญ่ หรือมีการตรวจสอบข้อมูลที่เป็นตารางหลายๆตาราง แม้กระทั่งการจัดเรียงข้อมูลให้เป็นระเบียบ เรามักจำเป็นที่จะต้องในการวนลูป หลายๆลูปซ้อนกันเพื่อให้สามารถเข้าถึงข้อมูลได้ทั้งหมด

ตัวอย่างการคำนวณตารางคูณค่า โดยใช้การซ้อนกันของ while ลูป

```
i = 2
while i < 10 :
    print "Multiply of " , i
    j = 1
    while j <= 12 :
        print "%d x %d = %d"%(i,j,i*j)
        j = j + 1
    i=i+1
```

ตัวอย่างการคำนวณตารางคูณค่า โดยใช้การซ้อนกันของ for ลูป

```
for i in range (2,10):
    print "Multiply of ",i
    for j in range (2,13):
        print "%d x %d = %d" %(i,j,i*j)
```

ผลลัพธ์ที่ได้

Multiply of 2

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
2 x 11 = 22
2 x 12 = 24
Multiply of 3
3 x 1 = 3
```

การเขียนโปรแกรมพื้นฐาน 101

```
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
3 x 11 = 33
3 x 12 = 36
Multiply of 4
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
4 x 11 = 44
4 x 12 = 48
Multiply of 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
5 x 11 = 55
5 x 12 = 60
Multiply of 6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
6 x 11 = 66
6 x 12 = 72
Multiply of 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
```

การเขียนโปรแกรมพื้นฐาน 101

```
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
7 x 11 = 77
7 x 12 = 84
Multiply of 8
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
8 x 11 = 88
8 x 12 = 96
Multiply of 9
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
9 x 11 = 99
9 x 12 = 108
```

ข้อระวังในการการวนลูปซ้อนของ while ลูปคือ ตัวแปรของเงื่อนไขในการวนลูป เนื่องจากมันสามารถถูกแก้ไขค่าได้ทุกที่ในช่วงลูปของมัน อาจจะทำให้โปรแกรมทำงานผิดพลาดได้ เช่น

```
i = 2
while i < 10 :
    print "Multiply of " , i
    j = 1
    while j <= 12 :
        print "%d x %d = %d"%(i,j,i*j)
        j = j + 1
        i = j
    i=i+1
```

ผลลัพธ์ที่ได้

```
Multiply of 2
2 x 1 = 2
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
```

การเขียนโปรแกรมพื้นฐาน 101

```
8 x 8 = 64
9 x 9 = 81
10 x 10 = 100
11 x 11 = 121
12 x 12 = 144
```

แต่ปัญหานี้จะไม่เกิดขึ้นกับรูปแบบ for เพราะคำสั่ง for แต่ละคำสั่ง ตัวแปรใน คำชี้แจงของลูปนั้น เป็นคนละตัว แยกจากกัน และไม่สามารถแก้ไขค่าได้

```
for i in range (2,10):
    print "Multiply of ",i
    for i in range (2,13):
        print "%d x %d = %d" %(i,i*i)
```

ผลลัพธ์ที่ได้

```
Multiply of  2
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of  3
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of  4
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of  5
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
```

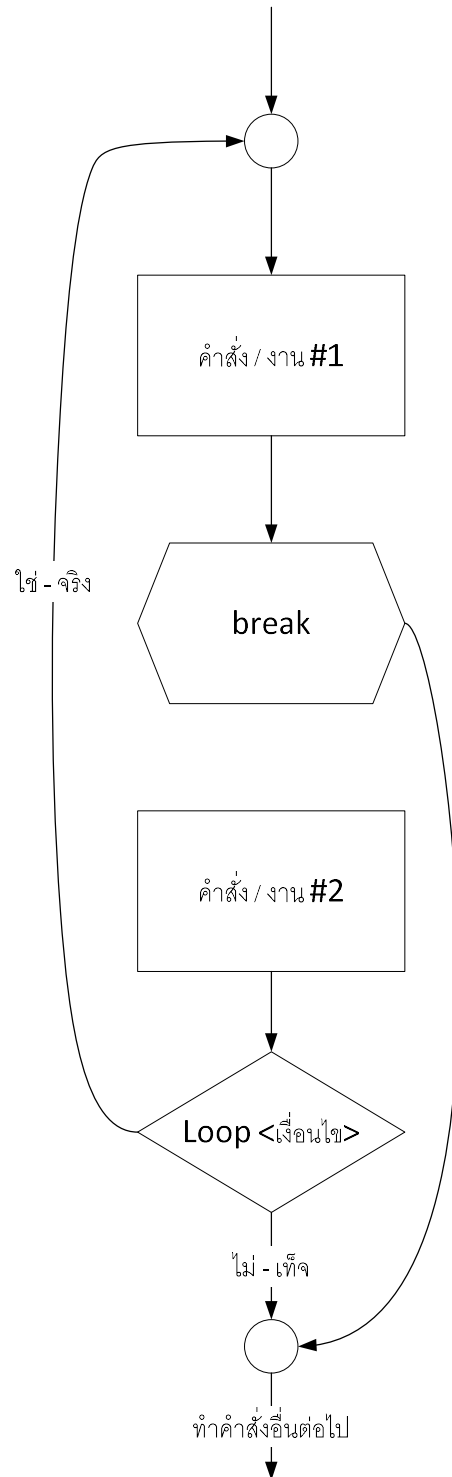
การเขียนโปรแกรมพื้นฐาน 101

```
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of 6
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of 7
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of 8
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
Multiply of 9
2 x 2 = 24
3 x 3 = 36
4 x 4 = 48
5 x 5 = 60
6 x 6 = 72
7 x 7 = 84
8 x 8 = 96
9 x 9 = 108
10 x 10 = 120
11 x 11 = 132
12 x 12 = 144
```

การเขียนโปรแกรมพื้นฐาน 101

คำสั่งควบคุมการวนซ้ำ

break – เป็นคำสั่งให้โปรแกรมออกจากลูปที่วนอยู่ และไปทำงานในคำสั่งถัดไปนอกลูปนั้น



การเขียนโปรแกรมพื้นฐาน 101

```
for letter in 'Python':      # First Example
    if letter == 'h':
        break
    print "print after break"
    print 'Current Letter :', letter

var = 10                      # Second Example
while var > 0:
    var = var -1
    if var == 5:
        break
    print "print after break"
    print 'Current variable value :', var
print "Good bye!"
```

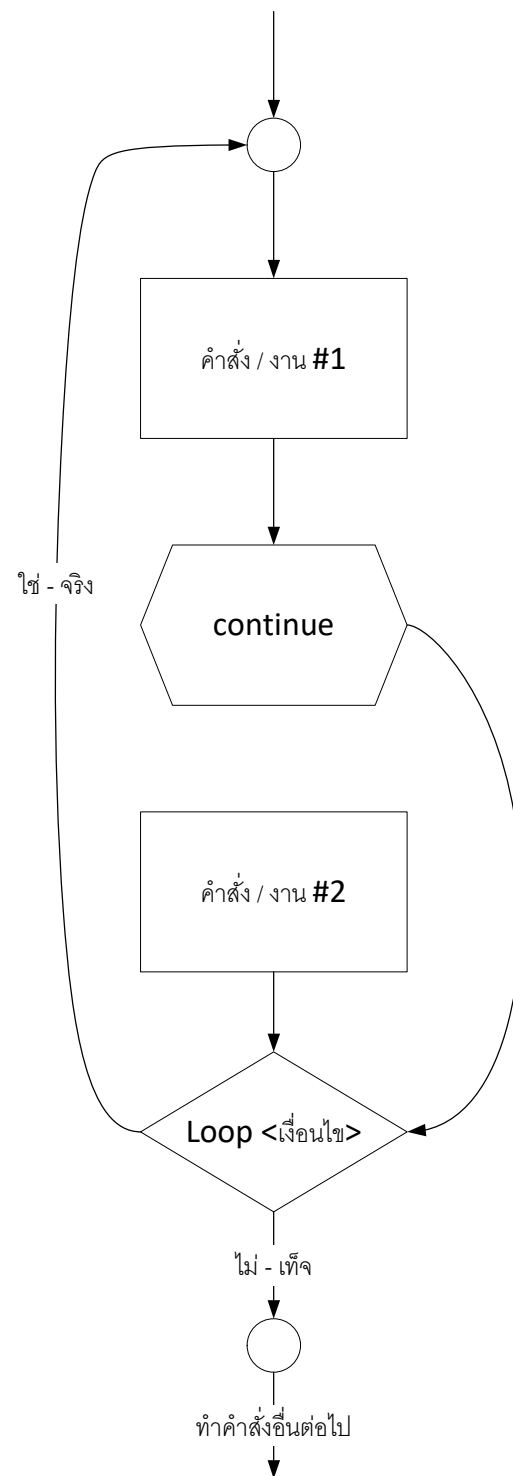
ผลลัพธ์ที่ได้

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

จะเห็นว่าโปรแกรมทำงานถึงแค่เงื่อนไขที่เราตั้ง break ไว้เท่านั้น แล้วออกจากการวนลูปไปทำงานในคำสั่งอื่นต่อ

การเขียนโปรแกรมพื้นฐาน 101

continue – เป็นคำสั่งให้โปรแกรมข้ามรอบงานในลูปที่ทำอยู่นี้ไปทำในรอบถัดไปทันที



การเขียนโปรแกรมพื้นฐาน 101

```
for letter in 'Python':      # First Example
    if letter == 'h':
        continue
    print "print after continue"
    print 'Current Letter :', letter

var = 10                      # Second Example
while var > 0:
    var = var -1
    if var == 5:
        continue
    print "print after continue"
    print 'Current variable value :', var
print "Good bye!"
```

ผลลัพธ์ที่ได้

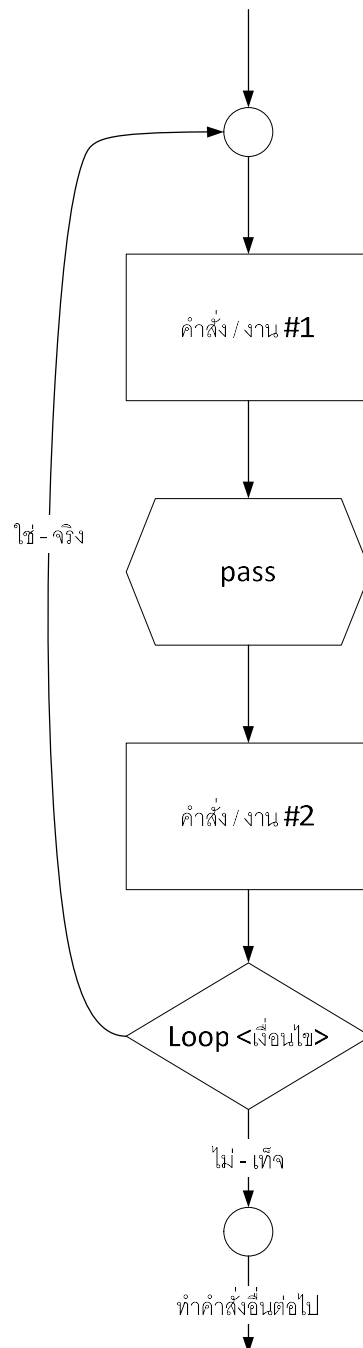
```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

จะเห็นว่าโปรแกรมจะข้ามข้อความหลังคำสั่ง continue และไม่แสดงตัวอักษรที่เรามีเงื่อนไขตรงกับที่ได้

continue

การเขียนโปรแกรมพื้นฐาน 101

pass – เป็นคำสั่งที่หมายถึง ให้ ”ผ่าน” ไป นั่นคือโปรแกรมจะไม่ทำอะไรเลย และจะผ่านไปตามขั้นตอนปกติ มักจะใช้เมื่อเราไม่ต้องการให้เงื่อนไขนี้มีการทำอะไร และผ่านไปเฉยๆ แต่ภาษาไพทอนนั้น ไม่ยินยอมให้มีการใช้คำสั่งแบบตรวจสอบเงื่อนไข หรือการวนลูป โดยไม่มีงานใดๆในเงื่อนไขนั้น หรือในลูปนั้นได้ เราจึงต้องใช้ คำสั่ง pass



การเขียนโปรแกรมพื้นฐาน 101

```
for letter in 'Python':      # First Example
    if letter == 'h':
        pass
    print "print after break"
    print 'Current Letter :', letter

var = 10                      # Second Example
while var > 0:
    var = var -1
    if var == 5:
        pass
    print "print after break"
    print 'Current variable value :', var
print "Good bye!"
```

ผลลัพธ์ที่ได้

```
Current Letter : P
Current Letter : y
Current Letter : t
print after break
Current Letter : h
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
print after break
Current variable value : 5
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

คำสั่ง `pass` นี้ไม่มีผลใดๆต่อการทำงานของโปรแกรมเลย แต่มันสามารถใช้ในกรณีที่ เรามีเงื่อนไขที่เราต้องการตรวจสอบ แต่ไม่ต้องให้มีผลอะไรเช่น

```
for letter in 'Python':      # First Example
    if letter == 'h':
        pass
    print 'Current Letter :', letter

var = 10                      # Second Example
while var > 0:
    var = var -1
    if var == 5:
        pass
    print 'Current variable value :', var
print "Good bye!"
```

การเขียนโปรแกรมพื้นฐาน 101

ผลลัพธ์ที่ได้

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 5
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

ซึ่งในภาพท่อน หลังการตรวจสอบเงื่อนไข หรือการวนลูปใดๆ จำเป็นที่จะต้องรองรับด้วยคำสั่งเพื่อทำงานเสมอ แต่ในบางกรณีเรายังไม่ต้องการให้มีการทำงาน แต่เราต้องการให้มีคำสั่งตรวจสอบ เราสามารถใช้ pass มาเป็นคำสั่งแทนได้

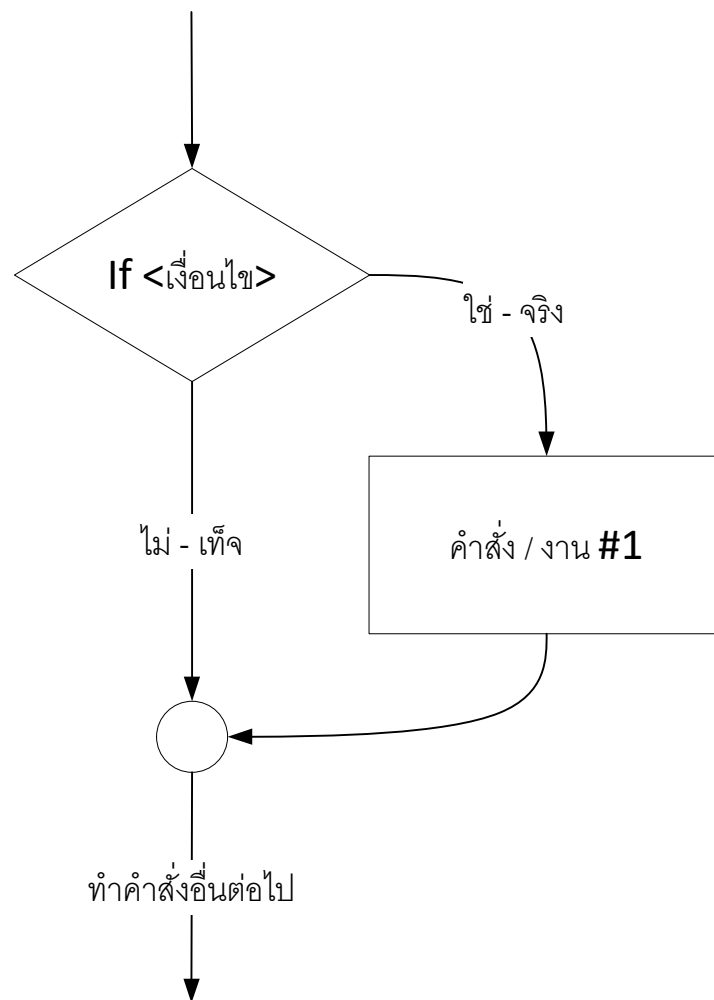
คำสั่งเงื่อนไข

เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไข เพื่อตัดสินใจว่าจะทำคำสั่งหรือไม่

คำสั่งประเภทเงื่อนไขนี้มี 2 คำสั่งคือ if และ if...else ข้อแตกต่างสำคัญของ if และ if-else นั้นคือ if จะทำคำสั่งเมื่อเงื่อนไขเป็นจริงเพียงอย่างเดียว ส่วน if-else นั้น เมื่อเงื่อนไขเป็นจริง ก็จะทำคำสั่งใน if แต่หากเงื่อนไขเป็นเท็จ จะมาทำคำสั่งใน else แปลว่า if-else จะมีคำสั่งทำทุกครั้งที่มีการตรวจสอบเงื่อนไข

If – if <เงื่อนไข> : เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไข เพื่อตัดสินใจว่าจะทำคำสั่งหรือไม่ และจะทำเมื่อเงื่อนไขเป็นจริงเพียงอย่างเดียวเท่านั้น

การเขียนโปรแกรมพื้นฐาน 101



ตัวอย่างการใช้ if

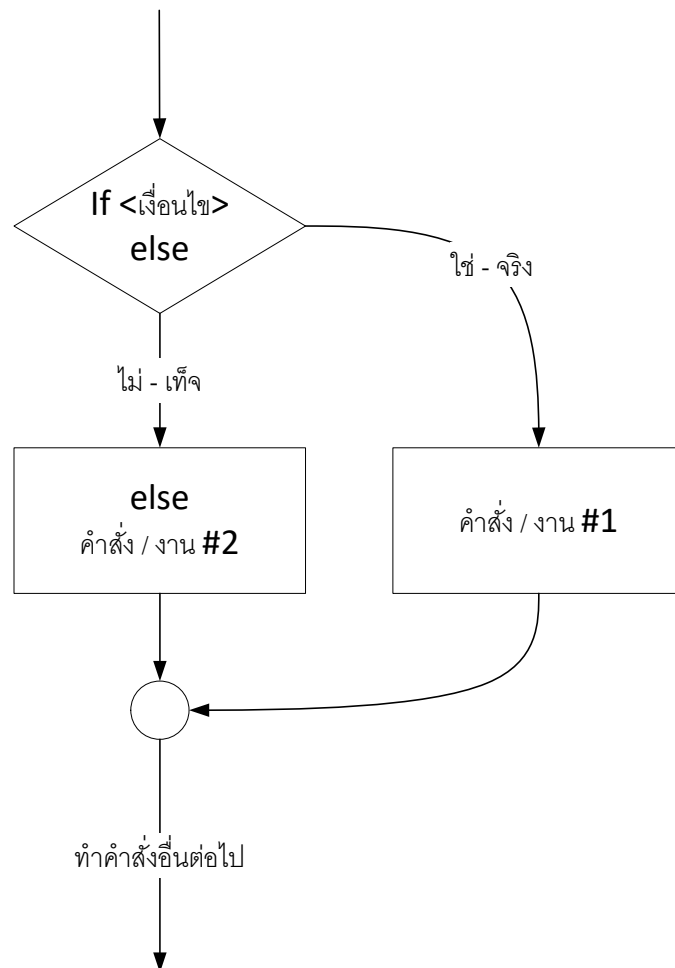
```
for i in range (1,11):  
    print i  
    if i == 5 :  
        print "this is five"
```

ผลลัพธ์ที่ได้

```
1  
2  
3  
4  
5  
this is five  
6  
7  
8  
9  
10
```

การเขียนโปรแกรมพื้นฐาน 101

If-else – if <เงื่อนไข> : ... else : คำสั่งนี้จะมีการทำคำสั่งทั้งในส่วน of if เมื่อเงื่อนไขเป็นจริง และ else เมื่อเงื่อนไขเป็นเท็จ



ตัวอย่างการใช้ if-else

```
for i in range (1,11):  
    if i == 5 :  
        print "this is five"  
    else :  
        print i
```

ผลลัพธ์ที่ได้

```
1  
2  
3  
4  
this is five
```

การเขียนโปรแกรมพื้นฐาน 101

6
7
8
9
10

การใช้คำสั่ง if-else ซ้อนกัน หลายเงื่อนไข เพื่อให้ง่ายต่อการแยกเงื่อนไขการตัดสินใจของโปรแกรม ในกรณีที่
มีเงื่อนไขมากกว่า 1 เงื่อนไข เช่น

การแทนตัวเลขด้วยข้อความข้างต้น แต่มีการใช้เงื่อนไขเพิ่มขึ้น เช่น ในช่วง 3-7 นั้น ให้แสดงคำว่า “this is
number : “ แล้วตามด้วยตัวเลข ยกเว้นเลข 5 ที่ต้องแสดงข้อความว่า “this is five” เช่นเดิม และ ที่เลข 9
นั้นแสดงคำว่า “This is nine” แทนที่เลข 9

ตัวอย่างการใช้ if-else ซ้อนกัน

```
for i in range (1,11):  
    if i == 5 :  
        print "this is five"  
    elif i == 9 :  
        print "this is nine"  
    elif i < 8 and i > 1 :  
        print "this is number : ",i  
    else :  
        print i
```

ผลลัพธ์ที่ได้

```
1  
this is number : 2  
this is number : 3  
this is number : 4  
this is five  
this is number : 6  
this is number : 7  
8  
this is nine  
10
```

ซึ่งผลที่ได้จะต่างจากการใช้ if ธรรมดา

```
for i in range (1,11):  
    if i == 5 :  
        print "this is five"  
    if i == 9 :  
        print "this is nine"  
    if i < 8 and i > 1 :  
        print "this is number : ",i  
    else :  
        print i
```

การเขียนโปรแกรมพื้นฐาน 101

ผลลัพธ์ที่ได้

```
1
this is number : 2
this is number : 3
this is number : 4
this is five
this is number : 5
this is number : 6
this is number : 7
8
this is nine
9
10
```

ข้อควรระวังในการใช้ if-else แบบซ้อนกันคือ ลำดับการตรวจสอบเงื่อนไขของ if ที่ซ้อนกันอยู่ ซึ่งบางครั้งการสลับลำดับการตรวจสอบเงื่อนไข อาจจะทำให้ผลที่ได้เปลี่ยนไปด้วย เช่น

```
for i in range (1,11):
    if i == 9 :
        print "this is nine"
    elif i < 8 and i > 1 :
        print "this is number : ",i
    elif i == 5 :
        print "this is five"
    else :
        print i
```

ผลลัพธ์ที่ได้

```
1
this is number : 2
this is number : 3
this is number : 4
this is number : 5
this is number : 6
this is number : 7
8
this is nine
10
```


การเขียนโปรแกรมพื้นฐาน 101

เชิงอรรถ

1. “Hello World” เป็นประโยคที่เหล่านักพัฒนาโปรแกรมนิยมใช้กัน ในการทดลองเมื่อเริ่มต้นเรียนรู้ภาษาใหม่ๆ หรือเครื่องมือใหม่ ที่ใช้สำหรับพัฒนาโปรแกรม โดยเป้าหมายคือ สั่งให้โปรแกรมแสดงผลคำว่า “Hello World” บนหน้าจอ เพียงเท่านั้น แต่มันหมายถึงว่า เราเข้าใจการทำงานของภาษาในเบื้องต้น และเข้าใจกระบวนการใช้งาน IDE ได้ถูกต้อง จุดเริ่มต้นของการใช้คำว่า “Hello World” ในการทดสอบนั้น มาจากชายที่ชื่อ Brian Kernighan ผู้แต่งหนังสือการเขียนโปรแกรมด้วยภาษา C ที่ชื่อว่า “The C Programming Language” ในปี พ.ศ. 2512 (ค.ศ. 1978) ได้เขียนตัวอย่างโปรแกรมที่แสดงผลคำว่า “hello world” บนหน้าจอ ซึ่งหนังสือเล่มนี้มีการใช้งานอย่างแพร่หลายมาก ส่งผลให้นักพัฒนาในยุคต่อมา มักจะเขียนโปรแกรมเริ่มต้นด้วยคำว่า “Hello World” ตามตัวอย่างในหนังสือตามไปด้วย
2. “Python” ถูกนำมาใช้ครั้งแรกในปี ค.ศ.1989 (พ.ศ.2532) โดย Guido van Rossum ผู้พัฒนาภาษา Python ออกแบบมาให้เป็นภาษาสคริปต์ระดับสูง คือมีความเข้าใจภาษามนุษย์มากขึ้น เราสามารถพอจะคาดเดาผลจากการใช้งานของคำสั่งในแต่ละคำสั่งได้จากภาษาที่ใช้ และรองรับรูปแบบการพัฒนาโปรแกรมสมัยใหม่ รวมไปถึงแนวความคิดโดยทั่วไป ในการพัฒนาโปรแกรมสมัยใหม่ ซึ่งสามารถนำมาใช้ได้กับการพัฒนาโปรแกรมบนภาษาอื่นๆ ได้เช่นเดียวกัน
3. “Zen of Python” - “คำพราวนิติแห่งไพทอน”
Beautiful is better than ugly.
ความงามล้ำค่ากว่าไร้ระเบียบ
Explicit is better than implicit.
ความชัดเจนมีความหมายกว่าทุกนัยยะ
Simple is better than complex.
ความเรียบง่ายสมบูรณ์กว่าซับซ้อน
Complex is better than complicated.
แต่ความซับซ้อนก็ยังดีกว่าความสับสน
Flat is better than nested.
ความแบนเรียบมันคงดีกว่าการซ้อนทับ
Sparse is better than dense.
ความบางเบาสบายกว่าทึบแน่น
Readability counts.
จงเคารพต่อผู้อ่าน
Special cases aren't special enough to break the rules.
ไม่มีกรณีใด พิเศษพอที่จะอยู่เหนือกฎ
Although practicality beats purity.
แม้ว่าประสิทธิภาพจะทำลายความไร้เดียงสา
Errors should never pass silently.
ข้อผิดพลาดก็ไม่เคยที่ผ่านไปอย่างเงียบงัน
Unless explicitly silenced.
ยกเว้นว่ามันจะเงียบเฉียบ
In the face of ambiguity, refuse the temptation to guess.
การห้ามใจจากการคาดเดา เมื่อต้องเผชิญหน้ากับความไม่แน่ใจ

การเขียนโปรแกรมพื้นฐาน 101

There should be one-- and preferably only one --obvious way to do it.

มีเพียงทางเดียว ทางเดียวที่ชัดเจน ทางเดียวที่ทำได้

Although that way may not be obvious at first unless you're Dutch.

แม้ว่าทางนั้น ดูเหมือนจะผิดในตอนแรก เว้นแต่คุณจะเป็นชาวดัตช์

Now is better than never.

และตอนนี้ก็ดีกว่าที่เคย

Although never is often better than *right* now.

แม้ว่าบ่อยครั้งที่ผ่านมาก็ดีกว่าตอนนี้

If the implementation is hard to explain, it's a bad idea.

ความคิดแยๆ คืออะไรก็ตามที่ไม่สามารถอธิบายได้

If the implementation is easy to explain, it may be a good idea.

ถ้าการทำงานนั้นเข้าใจได้ง่าย มันอาจจะเป็นความคิดที่ดี

Namespaces are one honking great idea -- let's do more of those!

เนมสเปซ คือวิธีการที่เยี่ยมในการประกาศความคิด -- จงทำให้มากขึ้นไปอีก

-19-Aug-2004-Tim Peters (วิศวกรคอมพิวเตอร์)

อ้างอิง

1. <http://blog.hackerrank.com/the-history-of-hello-world/>
2. https://www.tutorialspoint.com/python/python_quick_guide.htm
3. <http://www.fordantitrust.com/files/python.pdf>