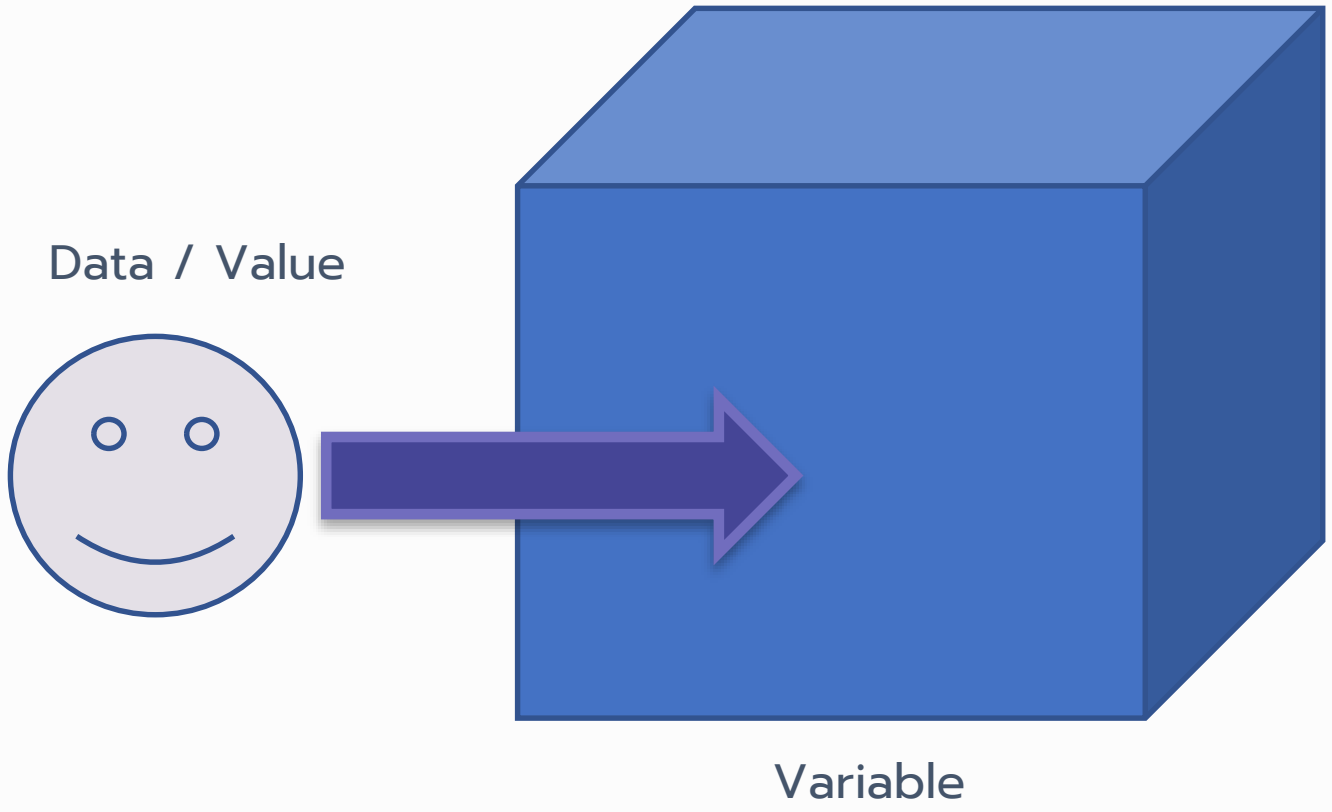
พื้นฐานการเขียนโปรแกรมภาษาไพธอน

บทที่ 7 อาร์เรย์

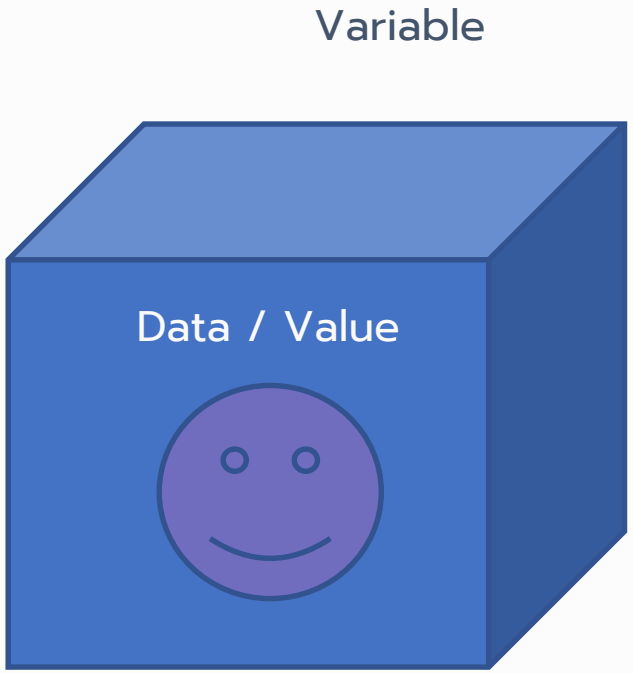
ข้อจำกัดของการใช้ตัวแปร



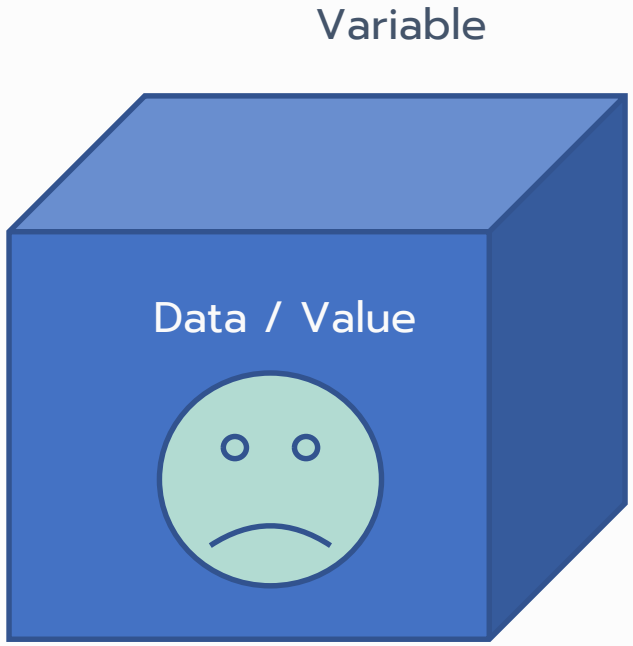
ตัวแปรแบบทั่วไป
สามารถเก็บค่าได้เพียง
ค่าเดียว

ข้อจำกัดของการใช้ตัวแปร

ตัวแปรแบบทั่วไป สามารถเก็บค่าได้เพียงค่าเดียว



หากต้องการเก็บมากกว่า 1 ค่า จำเป็นต้องมีตัวแปรเพิ่ม



ข้อจำกัดของการใช้ตัวแปร

```
my_string = "Hello World"

num_1 = 999

num_2 = 199.99

bool_ex = True
```

กรณีศึกษา

ท่านต้องการที่จะเก็บส่วนสูงของคนที่ทำงานในห้องเดียวกัน จำนวน 5 คน ท่านจำเป็นจะต้องใช้ตัวแปรทั้งหมดกี่ตัว

กรณีศึกษา

friend_1 = 178.0
friend_2 = 190.5
friend_3 = 186.7
friend_4 = 159.1
friend_5 = 162.0

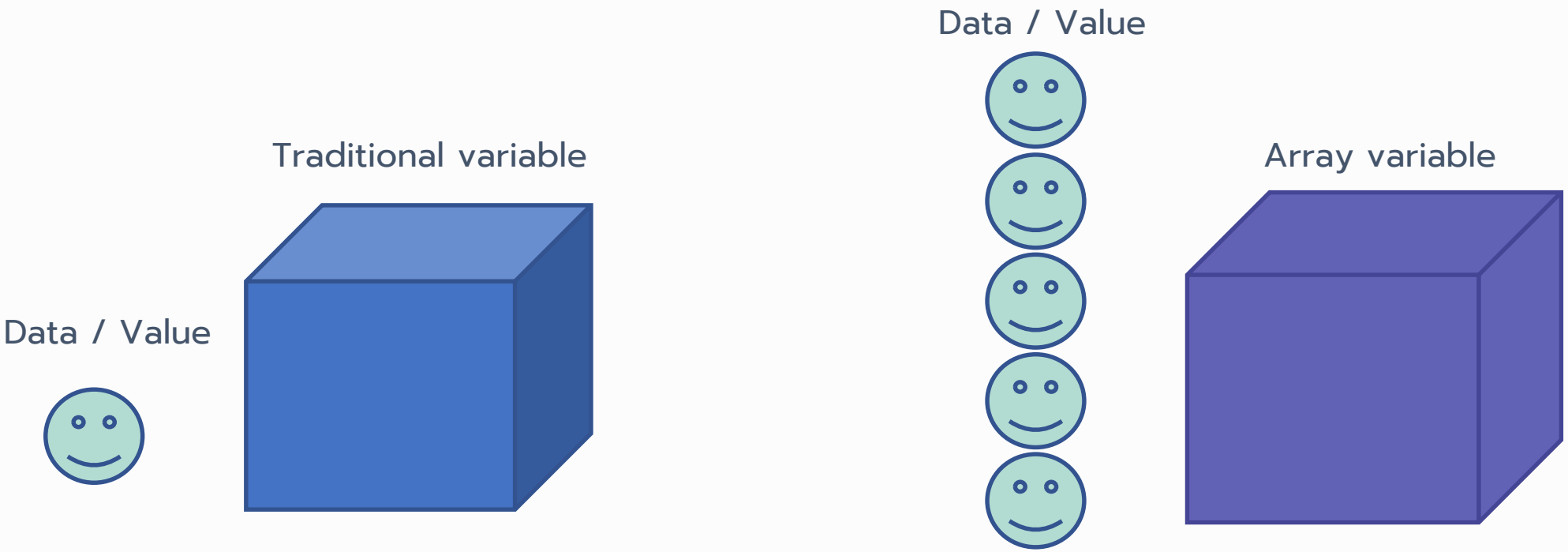
มีจำนวนเพื่อนร่วมงาน
ทั้งหมด 5 คน
=
5 ตัวแปรที่ใช้ในการเก็บ

ปัญหา

- จำเป็นต้องมีการประกาศตัวแปร เท่ากับจำนวนข้อมูลที่ต้องใช้
- การเข้าถึงและการจัดการทำได้ยาก

อะไรคืออาร์เรย์

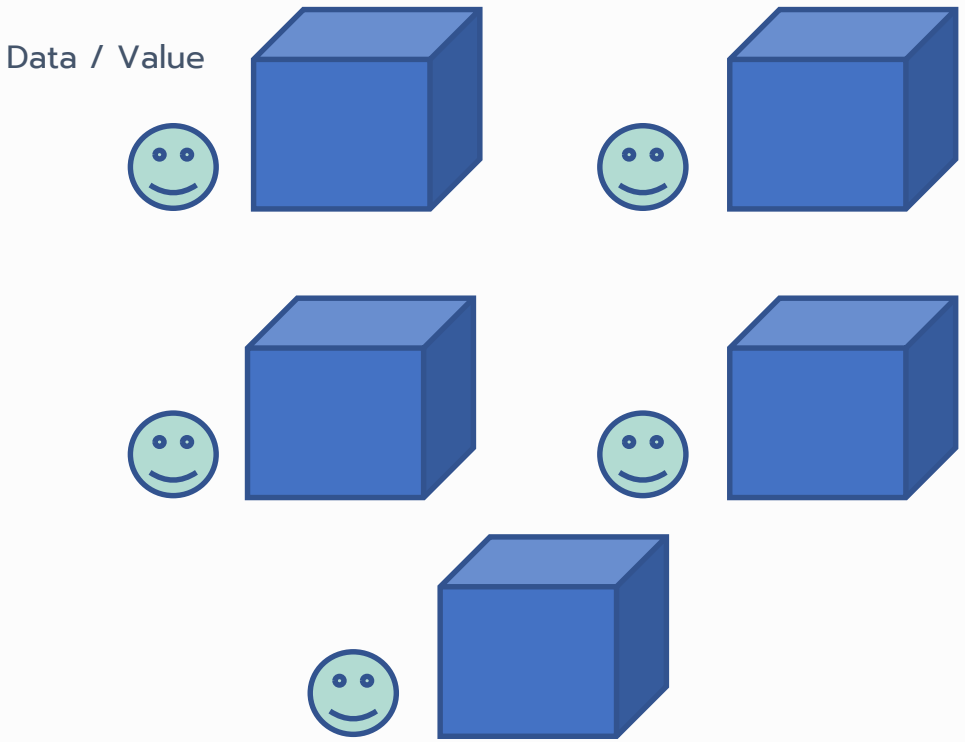
- อาร์เรย์ คือ ตัวแปรแบบชุด (Collection) ประเภทหนึ่ง ที่ใช้ตัวแปรเพียงตัวเดียวในการอ้างอิงถึงชุดของข้อมูล



อะไรคืออาร์เรย์

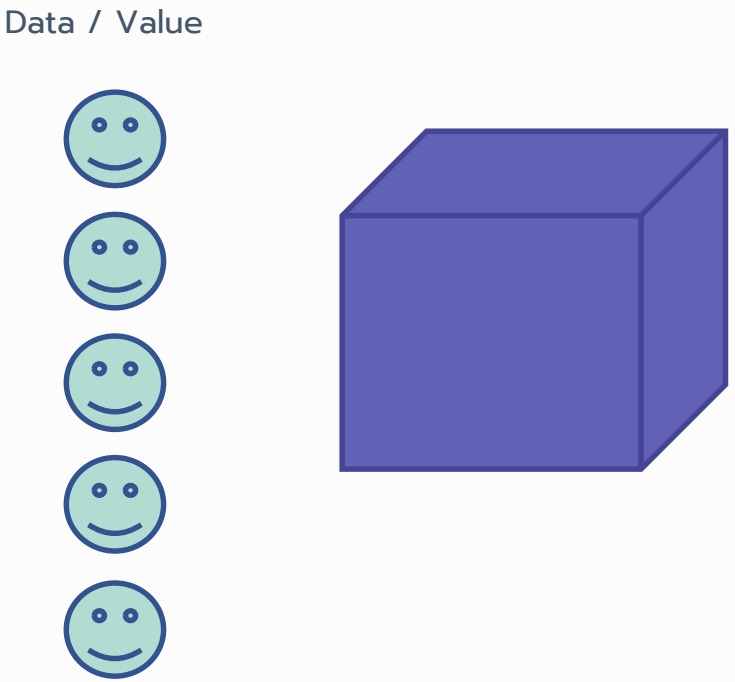
จากกรณีศึกษา เราสามารถที่จะจำลองการเก็บข้อมูลได้ดังนี้

ตัวแปรแบบทั่วไป



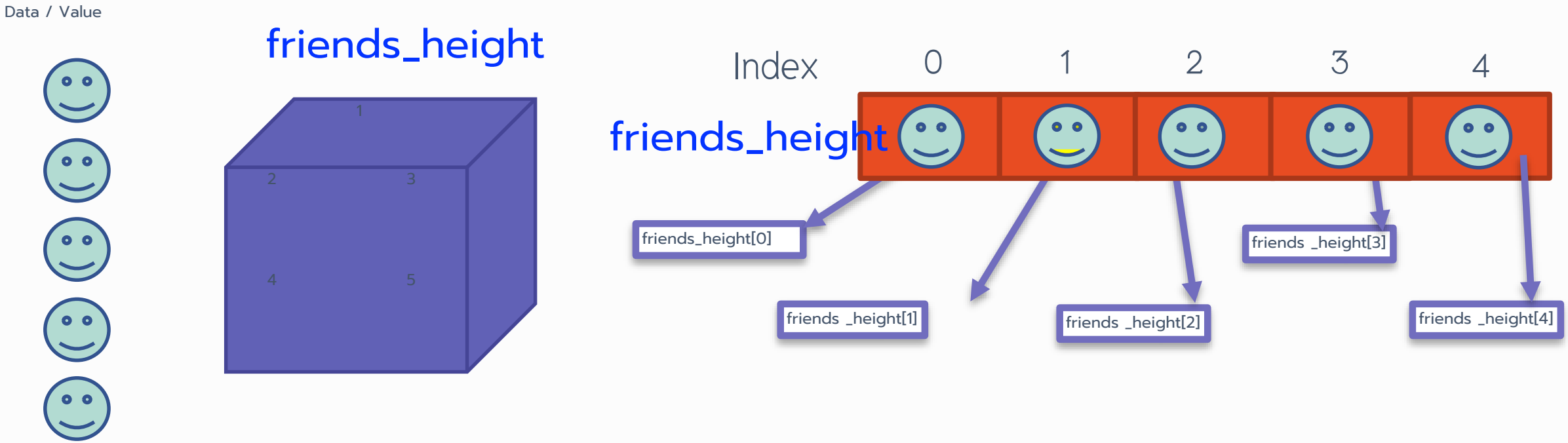
VS

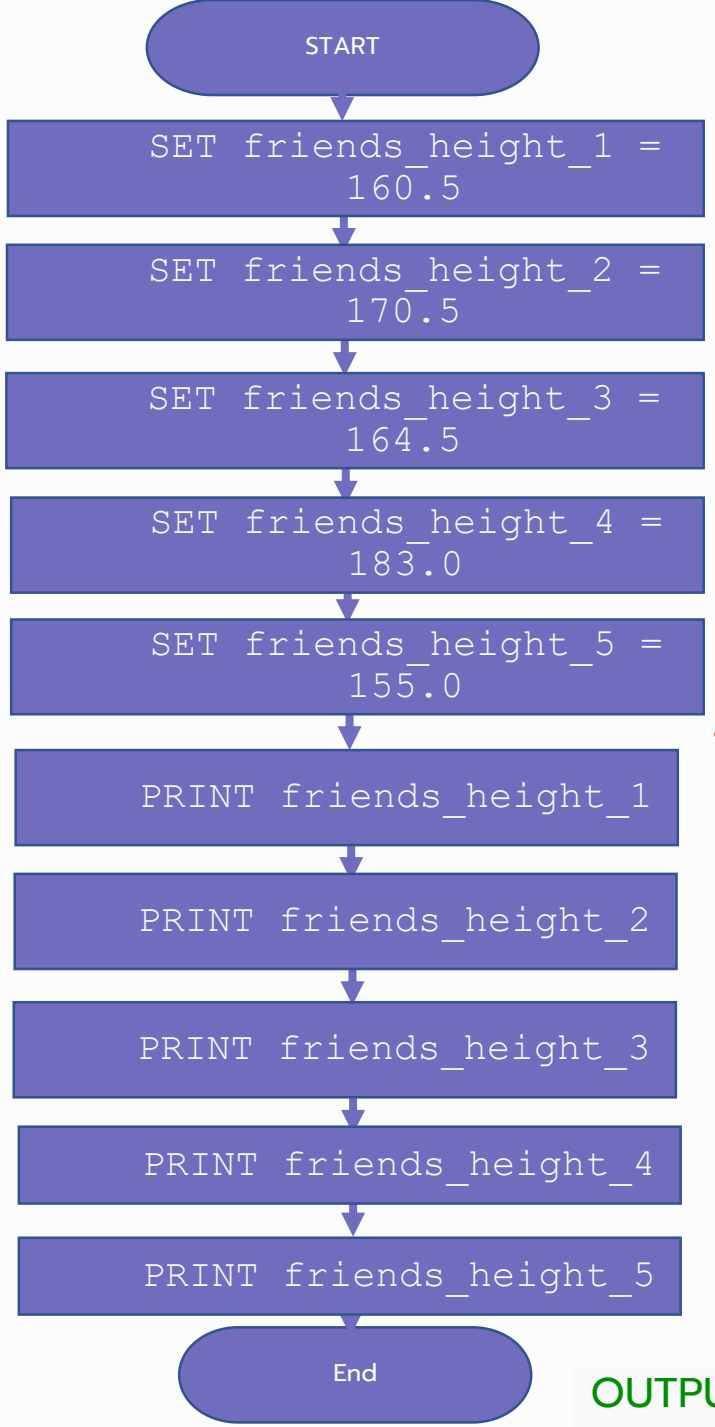
ตัวแปรแบบอาร์เรย์



คุณสมบัติของอาร์เรย์

- อาร์เรย์เป็นตัวแปรชนิดหนึ่ง
- การเข้าถึงสมาชิก (Element) แต่ละตัวในอาร์เรย์ต้องใช้ อินเด็กซ์ (Index)
  - อินเด็กซ์ = จำนวนเต็มบวกที่เริ่มต้นที่ 0 ที่ใช้ในการเข้าถึงข้อมูลในแต่ละตำแหน่ง
- การเข้าถึงข้อมูลไม่จำเป็นต้องเข้าถึงตามลำดับ สามารถเข้าถึงข้อมูลในตำแหน่งใดก็ได้



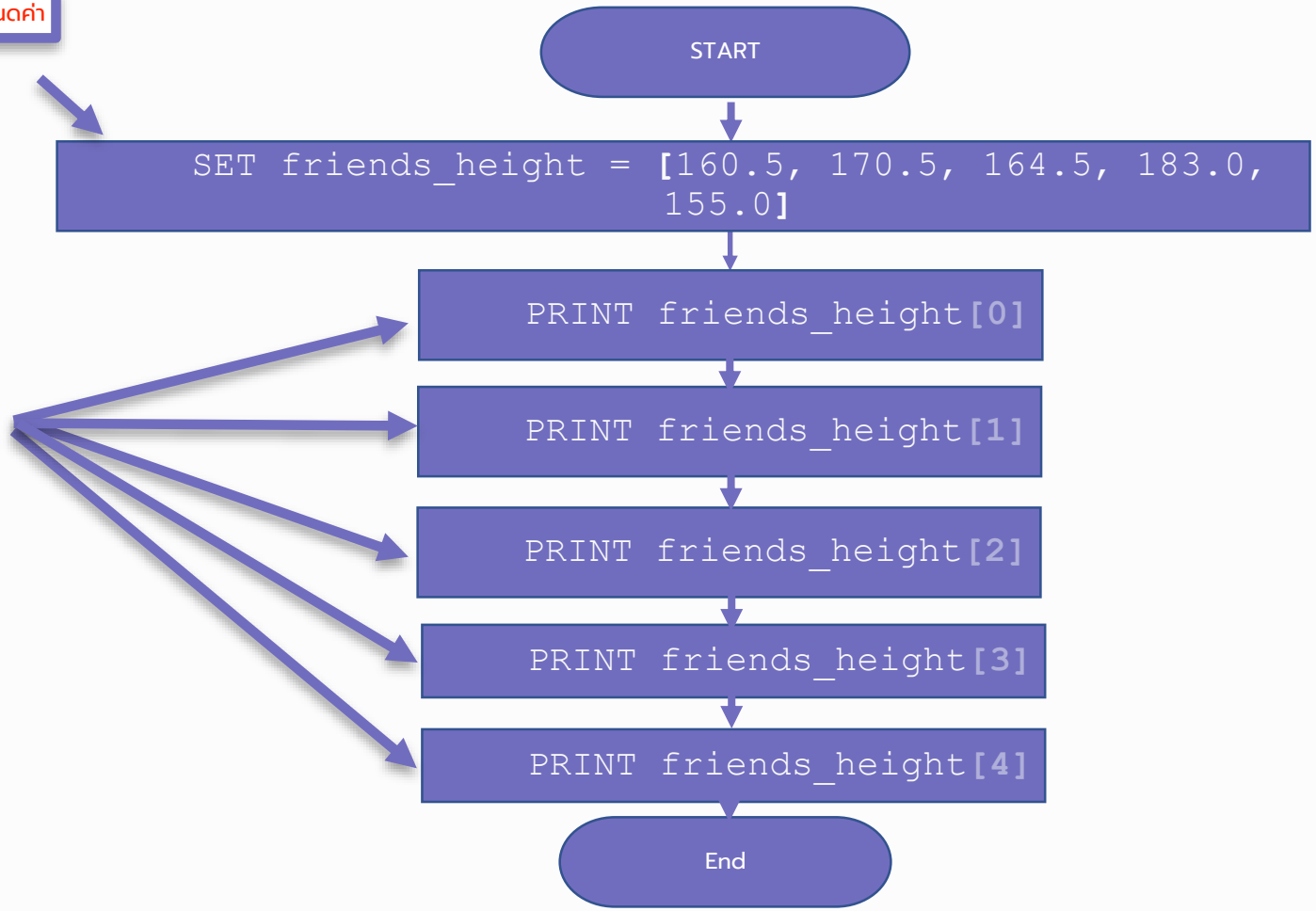


สามารถเข้าถึงได้โดยใช้อินเด็กซ์

arrayName[index]

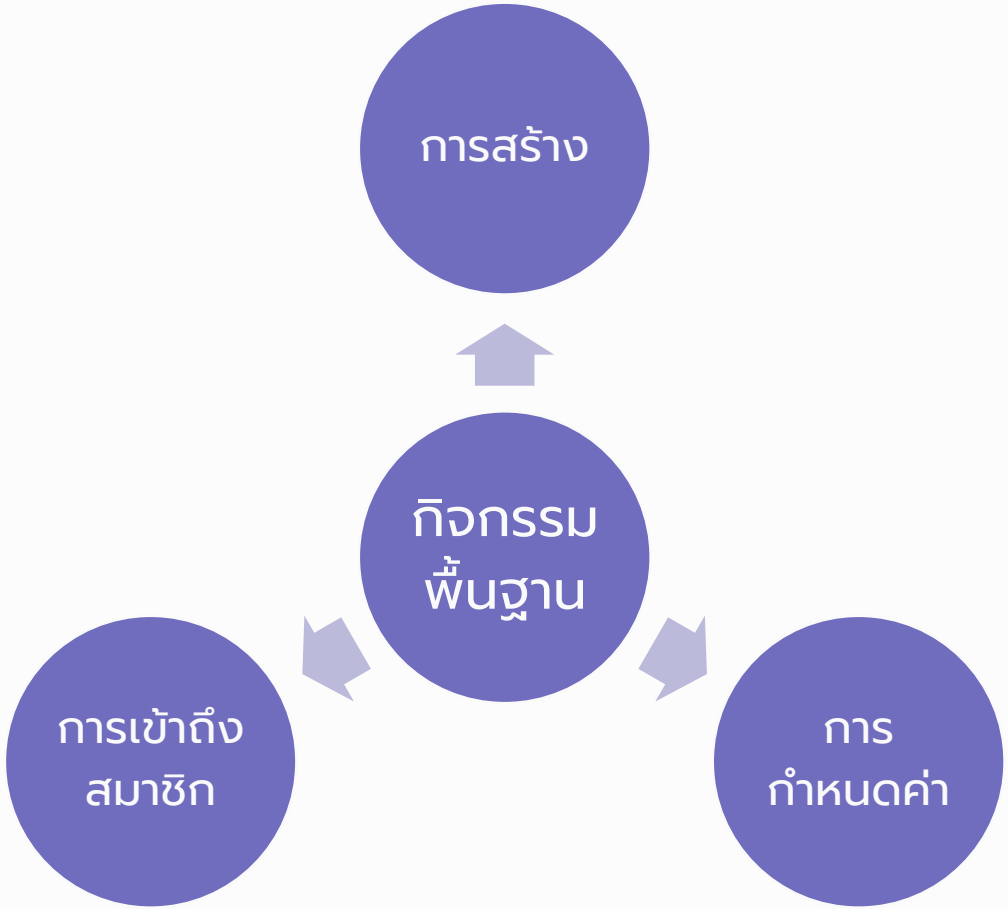
OUTPUT : 160.5 170.5 164.5 183.0 155.0

การกำหนดค่า



OUTPUT : 160.5 170.5 164.5 183.0 155.0

กิจกรรมที่สามารถดำเนินการได้กับอาร์เรย์



CHIANG MAI UNIVERSITY

การสร้างอาร์เรย์ในไพธอน

- ในภาษาไพธอนมีโครงสร้างข้อมูลหลายประเภท ที่สามารถนำมาใช้เป็นอาร์เรย์ได้
- ในบทเรียนนี้ เราจะใช้โครงสร้างที่มีชื่อว่า list
  - มีโครงสร้างชนิดอื่นอีก เช่น dict, tuple, set, dataframe, numpy array
  - แต่ละชนิดมีคุณลักษณะที่แตกต่างกัน
- การสร้าง list เพื่อใช้เป็นอาร์เรย์ สามารถสร้างได้โดยใช้เครื่องหมาย [ ]

การสร้างอาร์เรย์ในไพธอน

- รูปแบบทั่วไปของการสร้างอาร์เรย์โดยใช้ list

  [ชื่อตัวแปร] = [สมาชิกเริ่มต้นตัวที่ 1 , สมาชิกเริ่มต้นตัวที่ 2, ...]

- ตัวอย่างการสร้าง เช่น

  friends_height = [160.5, 170.5, 164.5, 183.0, 155.0]

  pet_name = []

  car_owner = list()

การเข้าถึงสมาชิกของอาร์เรย์ในไพธอน

- การเข้าถึงสมาชิกของอาร์เรย์
  - ผู้ใช้สามารถเข้าถึงสมาชิกแต่ละตัวของอาร์เรย์ได้โดยใช้ ชื่อตัวแปร ร่วมกับ อินเด็กซ์
  - รูปแบบทั่วไปของการเข้าถึง คือ

    [ชื่อตัวแปร][อินเด็กซ์]
- ตัวอย่างการใช้งาน เช่น

```
friends_height = [160.5, 170.5, 164.5, 183.0, 155.0]

print(friends_height[0])

print(friends_height[1]+10)

print(friends_height[2]+friends_height[3])
```

```
160.5
180.5
347.5
```

การกำหนดค่าให้อาร์เรย์ในไพธอน

- การกำหนดค่าใหม่ให้สมาชิกในอาร์เรย์ (Assignment) สามารถดำเนินการได้เช่นเดียวกับตัวแปรธรรมดา โดยใช้ ชื่อตัวแปร ร่วมกับ อินเด็กซ์

- รูปแบบทั่วไปของการกำหนดค่า คือ

  [ชื่อตัวแปร][อินเด็กซ์] = [ค่าใหม่]

- ตัวอย่างการใช้งาน เช่น

```
friends_height = [160.5, 170.5, 164.5, 183.0, 155.0]

print(friends_height[0])

friends_height[0] = friends_height[0] +10
print(friends_height[0])

friends_height[0] = friends_height[0] +friends_height[1]
print(friends_height[0])
```

```
160.5
170.5
341.0
```

พื้นฐานการเขียนโปรแกรมภาษาไพธอน

บทที่ 8 ฟังชัน

กรณีศึกษา

```
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
```

ผลลัพธ์บนหน้าจอ

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

จงเขียนชุดรหัสนี้ใหม่
โดยใช้โครงสร้างการ
ทำซ้ำที่ให้ผลลัพธ์
เหมือนเดิม

กรณีศึกษา

```
for counter in range(5):
    print("Hello World")
```

ผลลัพธ์บนหน้าจอ

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

จงเขียนชุดรหัสนี้ใหม่ โดยใช้โครงสร้างการทำซ้ำที่ให้ผลลัพธ์เหมือนเดิม

กรณีนี้ สามารถใช้โครงสร้างการทำซ้ำมาใช้งาน เพื่อให้ได้ผลลัพธ์ที่ยังเหมือนเดิม เนื่องจากการใช้งานคำสั่ง print(“Hello World”) เป็นการเรียกใช้แบบต่อเนื่องกัน

กรณีศึกษา

```
print("Hello World")
print("Do something else 1 ")
print("Hello World")
print("Hello World")
print("Do something else 2 ")
print("Do something else 3 ")
print("Hello World")
print("Hello World")
```

จงเขียนชุดรหัสนี้ใหม่ โดยใช้โครงสร้างการทำซ้ำที่ให้ผลลัพธ์เหมือนเดิม

ผลลัพธ์บนหน้าจอ

```
Hello World
Do something else 1
Hello World
Hello World
Do something else 2
Do something else 3
Hello World
Hello World
```

กรณีศึกษา

```
print("Hello World")
print("Do something else 1 ")
print("Hello World")
print("Hello World")
print("Do something else 2 ")
print("Do something else 3 ")
print("Hello World")
print("Hello World")
```

จงเขียนชุดรหัสนี้ใหม่ โดยใช้โครงสร้างการทำซ้ำ ที่ให้ผลลัพธ์เหมือนเดิม

กรณีศึกษานี้ สามารถใช้ โครงสร้างการทำซ้ำมาช่วยได้ยาก เนื่องจาก print(“Hello World”) มีความกระจัดกระจาย

การใช้ชุดรหัสซ้ำ (Reusability)

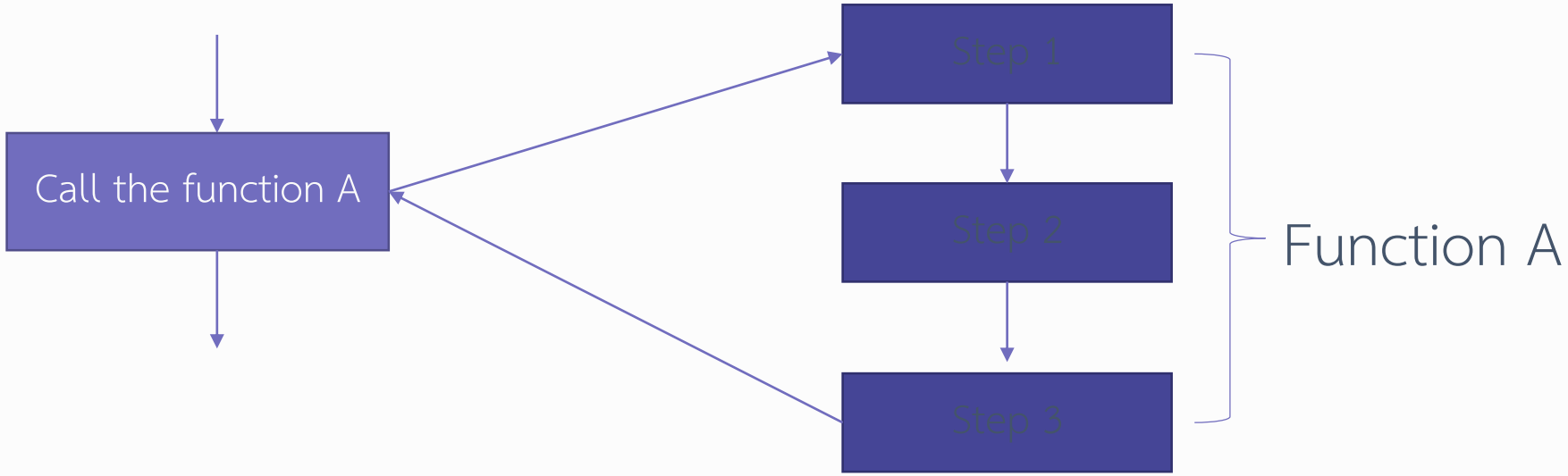
- การใช้ชุดรหัสซ้ำ คือ การที่นักพัฒนาซอฟต์แวร์ใช้ชุดรหัสที่มีการทำงานคล้ายคลึงกัน ซ้ำในหลายจุดของโปรแกรม
  - แต่ในแต่ละครั้งของการเรียกใช้ อาจจะมีค่าบางอย่างที่แตกต่างกัน
  - การเรียกใช้แต่ละครั้งไม่จำเป็นต้องต่อเนื่องกัน

- ชุดรหัสที่ถูกใช้ซ้ำในลักษณะนี้เรียกว่า ฟังก์ชัน
  - บางครั้งถูกเรียกว่า เมธอด (Method) หรือ โพรซีเยอร์ (Procedure)
  - ฟังก์ชันเป็นชุดรหัสนอกชุดรหัสหลัก ที่ทำงานเป็นเอกเทศ

- โครงสร้างการทำซ้ำไม่ได้รองรับการทำงานในลักษณะนี้

หลักการทำงานของฟังก์ชัน

- เมื่อส่วนอื่นของชุดรหัสมีการเรียกใช้ฟังก์ชัน
  - ข้อมูลจะถูกส่งไปยังฟังก์ชัน
  - ฟังก์ชันจะทำงานตามที่ถูกออกแบบไว้
  - ฟังก์ชันจะส่งค่ากลับมาที่จุดที่มีการเรียกฟังก์ชันนั้นๆ



องค์ประกอบของฟังก์ชันในภาษาไพธอน

- ฟังก์ชันมีองค์ประกอบทั้งหมด 4 องค์ประกอบ
  - ชื่อฟังก์ชัน
  - ผลลัพธ์
  - ข้อมูลนำเข้า
  - กระบวนการการทำงาน

ฟังก์ชัน : ชื่อฟังก์ชัน

- ชื่อฟังก์ชัน เป็นส่วนที่นักพัฒนาจะใช้ในการอ้างอิงถึง/การเรียกใช้ฟังก์ชันนี้

- การตั้งชื่อฟังก์ชันจะต้องถูกต้องตามหลักการของภาษาไพธอน
  - ชื่อฟังก์ชันจะต้อง ไม่เป็นคำสงวนของภาษาไพธอน (Keywords)
  - ชื่อฟังก์ชันจะต้อง ไม่มีสัญลักษณ์พิเศษ !, @, #, $, % เป็นต้น (_ สามารถใช่ได้)

- การตั้งชื่อฟังก์ชันจะต้องตั้งตามแนวทางที่ **สื่อความหมาย** และ **มีบรรทัดฐาน**
  - ชื่อฟังก์ชันจะต้องเป็นเอกลักษณ์ (Unique) – ไม่มีฟังก์ชันที่มีชื่อเดียวกัน สำหรับงานเดียวกัน
  - ชื่อฟังก์ชันจะต้องสื่อความหมาย (Meaningful) - อ่านแล้วสามารถเข้าใจได้
  - ชื่อฟังก์ชันจะต้องสอดคล้องกับมาตรฐานการตั้งชื่อ (Naming Convention) – เช่น snake case, camel case

ฟังก์ชัน : ชื่อฟังก์ชัน

- การตั้งชื่อฟังก์ชันในภาษาไพธอน จะใช้คำสั่งว่า def

```
def [ชื่อฟังก์ชัน] (รายการข้อมูลนำเข้า):
  [การทำงาน 1]
  [การทำงาน 2]
  return [ข้อมูลส่งกลับ]
```

- ตัวอย่างการกำหนดชื่อให้ฟังก์ชัน

```
def function_name(input_1,input_2):
  temp = input_1 + input_2
  return temp
```

ฟังก์ชัน : ข้อมูลนำเข้า

- ข้อมูลนำเข้า (Input parameter , input argument) คือ จุดที่ข้อมูลถูกส่งต่อมาจากจุดที่เรียกใช้ฟังก์ชัน เพื่อให้ข้อมูลที่จำเป็นต่อการประมวลผลสามารถถูกใช้งานในฟังก์ชันได้

- ข้อมูลนำเข้าสามารถมีได้หลายตัว หรืออาจจะไม่มีเลย

- การตั้งชื่อตัวข้อมูลนำเข้า จะเป็นไปตามหลักการการตั้งชื่อตัวแปร

ฟังก์ชัน : ข้อมูลนำเข้า

- **รายการข้อมูลจำเข้าจะต่อจากชื่อของฟังก์ชัน อยู่ใน (…)**

```
def [ชื่อฟังก์ชัน] (รายการข้อมูลนำเข้า):
  [การทำงาน 1]
  [การทำงาน 2]
  return [ข้อมูลส่งกลับ]
```

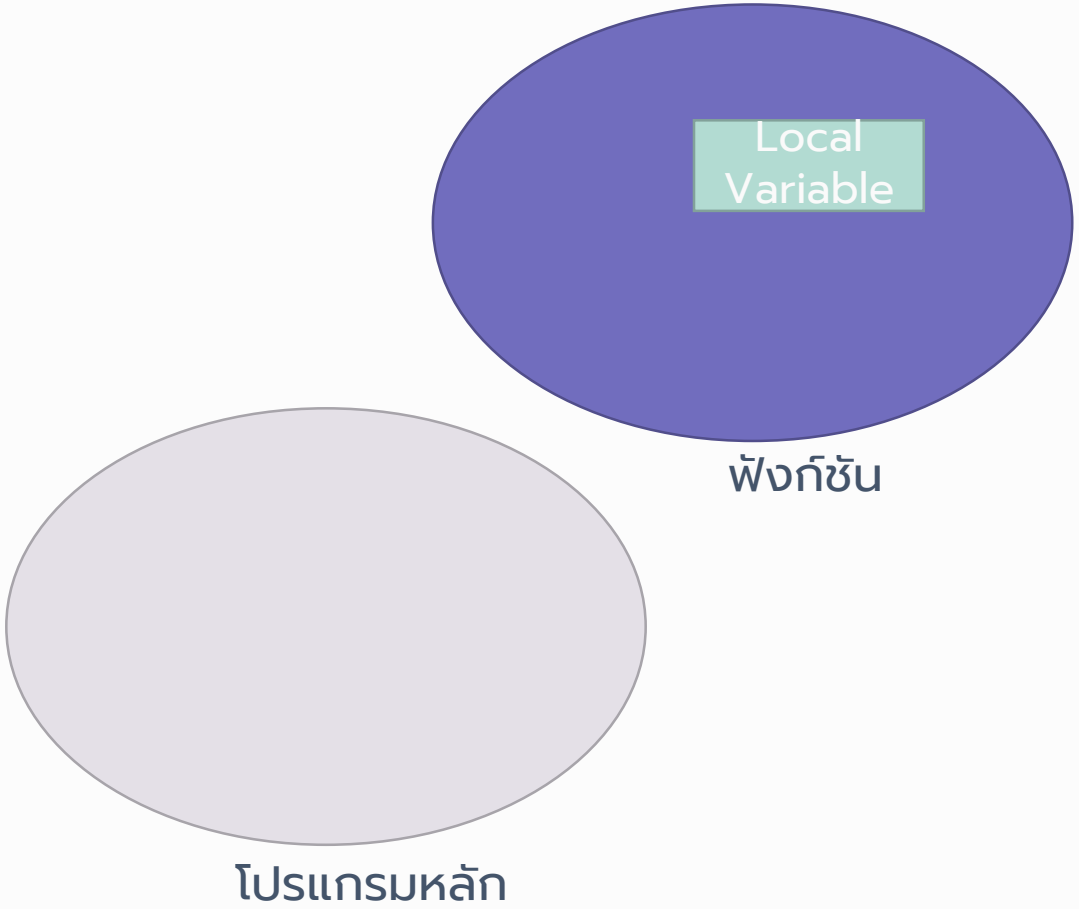
- **ตัวอย่างการกำหนดรายการข้อมูลนำเข้า**

```
def function_name(input_1,input_2):
  temp = input_1 + input_2
  return temp
```

ตัวแปรเฉพาะที่ (Local Variable) และ ตัวแปรส่วนกลาง (Global Variable)

ตัวแปรเฉพาะที่

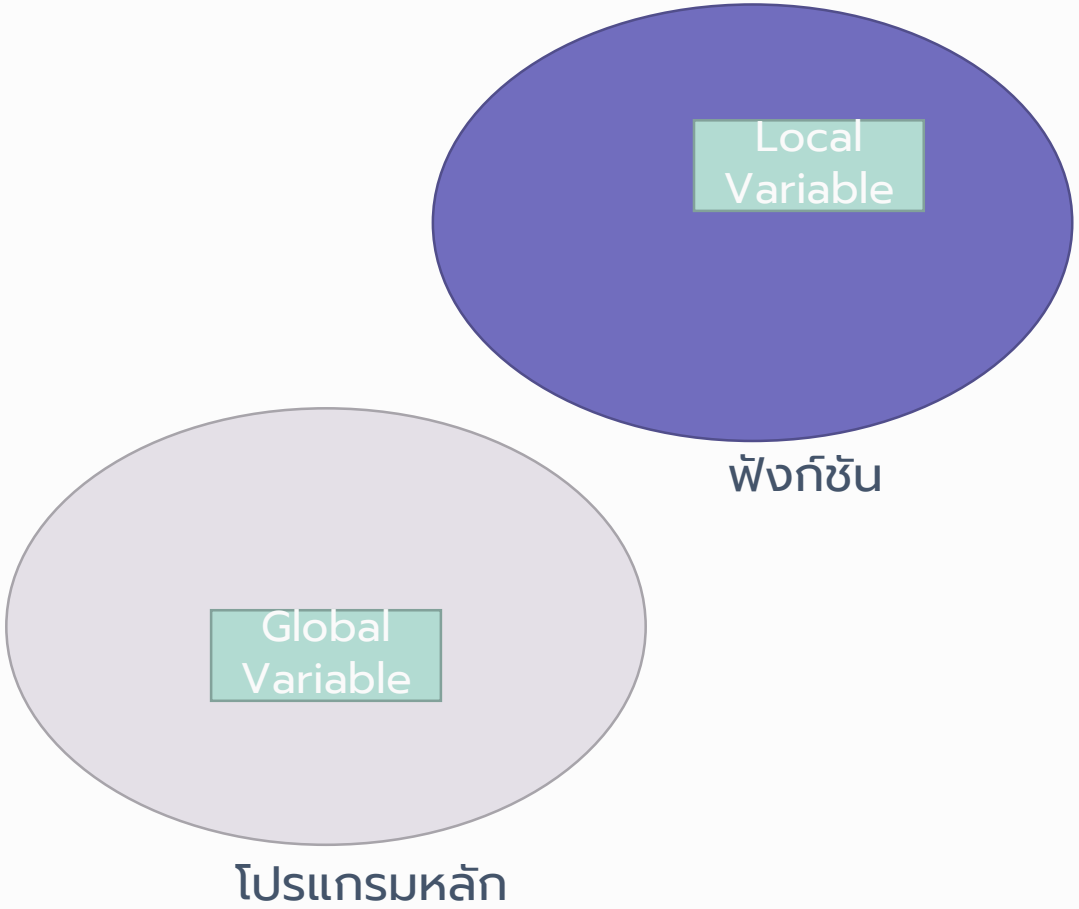
- ตัวแปรชนิดนี้ คือ ตัวแปรที่อยู่ในถูกประกาศใช้ ที่เป็นส่วนหนึ่งของโครงสร้างอื่น เช่น ฟังก์ชัน
- ตัวแปรชนิดนี้ จะสามารถใช้งานได้เพียงภายในของฟังก์ชันนั้น เท่านั้น
- เมื่อสิ้นสุดการทำงานของฟังก์ชันนั้น ตัวแปรเฉพาะที่ จะถูกทำลายไปด้วย



ตัวแปรเฉพาะที่ (Local Variable) และ ตัวแปรส่วนกลาง (Global Variable)

ตัวแปรส่วนกลาง

- ตัวแปรชนิดนี้ คือ ตัวแปรที่อยู่ในถูกประกาศใช้ ในโปรแกรมหลัก
- ทุกส่วนของโปรแกรมสามารถเข้าถึงตัวแปรชนิดนี้ได้
- ตัวแปรชนิดนี้ถูกสร้างเมื่อโปรแกรมเริ่มต้นการทำงานและตัวแปรชนิดนี้จะถูกทำลายเมื่อโปรแกรมสิ้นสุด



ฟังก์ชัน :กระบวนการการทำงาน

- กระบวนการทำงานของฟังก์ชัน คือ สิ่งที่นักพัฒนาซอฟต์แวร์ต้องการให้ฟังก์ชันทำงาน จะเป็นขั้นตอนการแปลข้อมูลนำเข้าที่รับมาจากจุดที่เรียกใช้ฟังก์ชัน แปลงเป็นผลลัพธ์ที่พึงประสงค์

- การเขียนกระบวนการการทำงานจะอยู่ในย่อหน้า ใต้ คำสั่ง def

```
def [ชื่อฟังก์ชัน] (รายการข้อมูลนำเข้า):
    [การทำงาน 1]
    [การทำงาน 2]
    return [ข้อมูลส่งกลับ]
```

```
def function_name(input_1,input_2):
    temp = input_1 + input_2
    return temp
```

ฟังก์ชัน : ผลลัพธ์

- ผลลัพธ์ คือ ค่าที่ฟังก์ชันจะส่งกลับไปจุดที่เรียกฟังก์ชัน
  - โดยทั่วไป แล้วจะมีการส่งค่ากลับเพียงค่าเดียว
- ฟังก์ชัน อาจจะมี หรือ อาจจะไม่มี ผลลัพธ์ที่ส่งกลับไปยังจุดที่เรียกฟังก์ชัน ก็ได้
- คำสั่งที่ใช้ในการระบุค่าที่จะเป็นการส่งกลับ คือ return

```
def [ชื่อฟังก์ชัน] (รายการข้อมูลนำเข้า):
    [การทำงาน 1]
    [การทำงาน 2]
    return [ข้อมูลส่งกลับ]
```

```
def function_name(input_1,input_2):
    temp = input_1 + input_2
    return temp
```

การเรียกใช้ฟังก์ชัน

- การเรียกใช้ฟังก์ชัน จะเป็นการส่งต่อค่า (ข้อมูลนำเข้า) เพื่อไปประมวลผลที่ ฟังก์ชัน และส่งกลับมายังจุดที่เรียก

- วิธีการเรียกใช้ฟังก์ชัน คือ การใช้ชื่อฟังก์ชันพร้อมทั้งระบุข้อมูลที่จะส่งไป
  - หาก ฟังก์ชันนั้นมีการส่งค่ากลับ จำเป็นจะต้องการระบุตัวแปรที่จะเอารับค่า

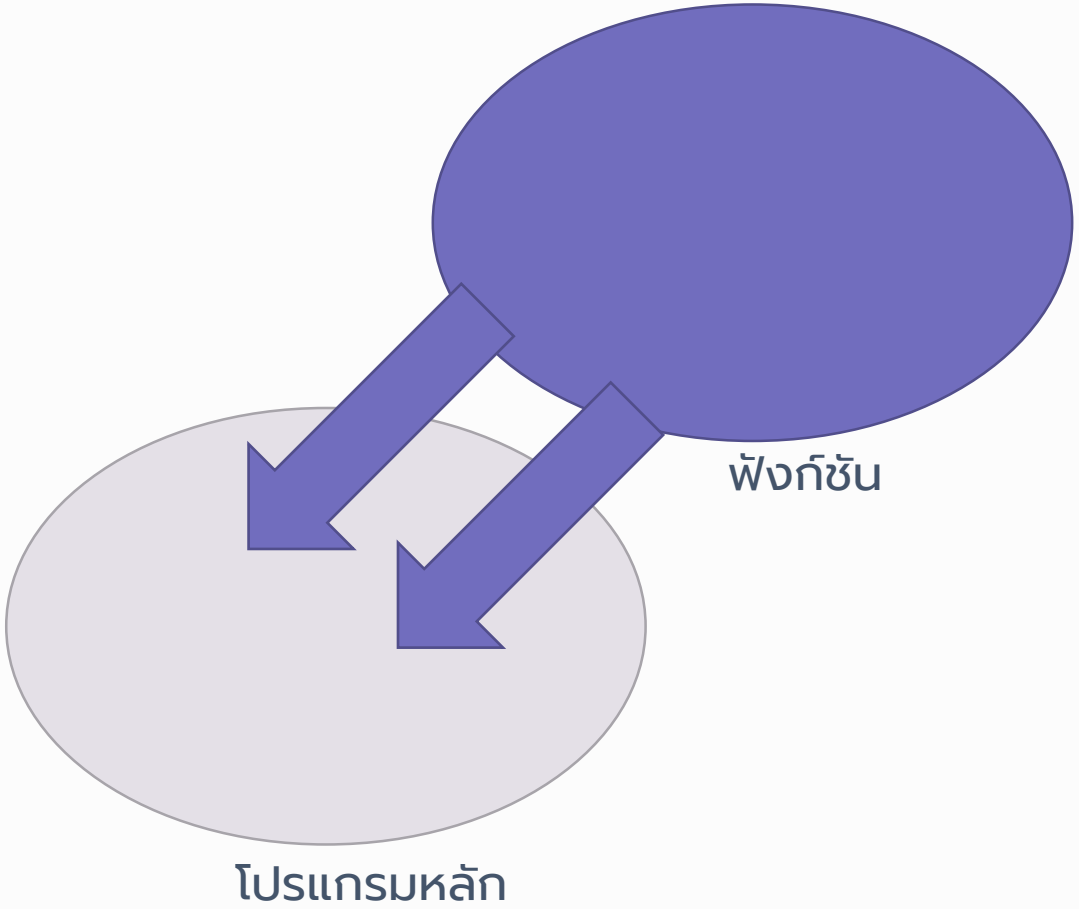
```
def function_name(input_1,input_2):
  temp = input_1 + input_2
  return temp

print(function_name(1,2))
print(function_name(5,6))
```

```
3
11
```

ประโยชน์ของการใช้ฟังก์ชัน

- ลดการเขียนชุดรหัสเดิมซ้ำหลายที่
- ทำให้โปรแกรมสามารถอ่านและเข้าได้ได้ง่าย
  - ลดความซับซ้อนของโปรแกรม
- สามารถปรับปรุงโปรแกรมได้ง่าย
  - ทดสอบโปรแกรมได้ง่าย



ฟังก์ชันที่มีการสร้างไว้แล้ว (Pre-Defined Function)

- ฟังก์ชันที่มีการสร้างไว้แล้ว คือฟังก์ชันประเภทหนึ่งที่ 1) ถูกสร้างในระบบโดยผู้พัฒนา 2) ผู้พัฒนาคนอื่นเป็นผู้สร้าง

- ฟังก์ชันที่มีการสร้างไว้แล้ว เป็นฟังก์ชันที่ทำงานธรรมดา จนถึงการประมวลผลขั้นสูง

- ผู้ที่เรียกใช้ฟังก์ชันที่มีการสร้างไว้แล้วไม่ต้องเข้าใจถึงรายละเอียดเชิงลึกของฟังก์ชันประเภทนี้ แต่อาจจะต้องสนใจกับข้อมูลนำเข้าที่ต้องส่งให้ และผลลัพธ์ที่ส่งกลับมา

ฟังก์ชันที่มีการสร้างไว้แล้ว - คณิตศาสตร์

- การเรียกใช้ฟังก์ชันทางคณิตศาสตร์จำเป็นจะต้องมีการเรียกใช้ กลุ่มคำสั่ง (library หรือ package) ที่มีชื่อว่า math

- ยกกำลัง

  pow(x, y) ผลลัพธ์คือ x ยกกำลัง y

- รากที่ 2

  sqrt(x) ผลลัพธ์คือ รากที่ 2 ของ x

- ปัด

  ceil(x) ผลลัพธ์คือ ปัด x ขึ้นเป็นจำนวนเต็มที่ใกล้ที่สุด

  floor(x) ผลลัพธ์คือ ปัด x ลงเป็นจำนวนเต็มที่ใกล้ที่สุด

ฟังก์ชันที่มีการสร้างไว้แล้ว - คณิตศาสตร์

```python
import math

print(math.pow(2,10))

print(math.sqrt(144))

print(math.ceil(10.5))
print(math.floor(10.5))
```

```
1024.0
12.0
11
10
```

ฟังก์ชันที่มีการสร้างไว้แล้ว -ข้อความ

- การเรียกใช้ฟังก์ชันที่เกี่ยวข้องกับการประมวลข้อความ เป็นฟังก์ชันที่มีมากับตัวภาษาไพธอน ดังนั้นจึงไม่จำเป็นต้องมีการนำเข้าชุดคำสั่งก่อนเรียกใช้

- การค้นหาข้อความ

  find(x) ผลลัพธ์คือ ตำแหน่งของ x ที่ปรากฏในข้อความหลัก

- การจัดรูปแบบข้อความ

  format (x) ผลลัพธ์คือ ข้อความที่ถูกจัดรูปแบบ ตามที่ผู้ใช้กำหนด

- การตรวจสอบว่าข้อความเป็นตัวเลขทั้งหมด

  isdigit(x) ผลลัพธ์คือ True เมื่อทุกอักขระในข้อความเป็นตัวเลข และ False เมื่อไม่ใช่

ฟังก์ชันที่มีการสร้างไว้แล้ว -ข้อความ

```
[4]  string = "Hello I am Yam."

     print(string.find("am"))
```

```
     8
```

```
[6]  string = "The number is {number:.2f}"

     print(string.format(number=199.9999999))
```

```
     The number is 200.00
```

```
     string_1 = "99999"
     string_2 = "9999AA.9999"

     print(string_1.isdigit())
     print(string_2.isdigit())
```

```
     True
     False
```

ฟังก์ชันที่มีการสร้างไว้แล้ว - List

- การเรียกใช้ฟังก์ชันที่เกี่ยวข้องกับการประมวล list เป็นฟังก์ชันที่มีมากับตัวภาษาไพธอน ดังนั้นจึงไม่จำเป็นต้องมีการนำเข้าชุดคำสั่งก่อนเรียกใช้

- การเชื่อม list

  append(x) ผลลัพธ์คือ list ที่มี x ต่อด้านหลัง

- การล้าง list

  clear() ผลลัพธ์คือ list ที่ไม่มีสมาชิก

- การคัดลอก

  copy(x) ผลลัพธ์คือ list ที่มีสมาชิกเหมือนกับต้นฉบับทุกประการ