เขียนโปรแกรมภาษา Go

สำหรับผู้เริ่มต้น | 2022

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/

เหมาะสำหรับ

- ผู้ที่สนใจเรียนรู้การเขียนโปรแกรมภาษา Go
- ไม่มีความรู้เรื่องการเขียนโปรแกรมก็เรียนได้
- เนื้อหาทั้งหมดเรียนฟรี!!

รู้จักกับภาษา Go

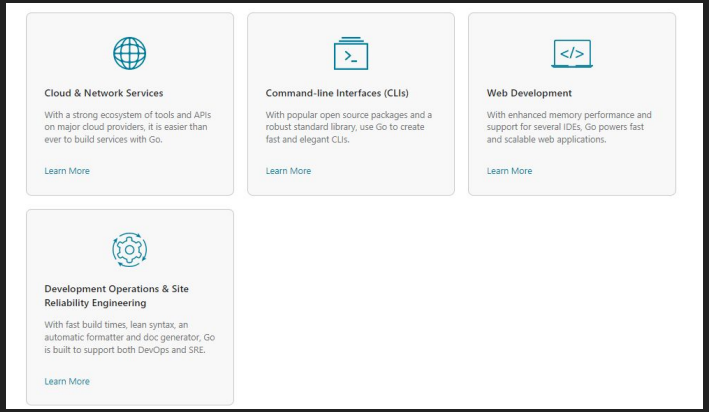
ภาษา Go เป็นภาษาโปรแกรมในรูปแบบ Open-Source ถูกพัฒนาโดยบริษัท Google มีจุดเด่นในเรื่องของ Performance ที่สามารถทำงานได้อย่างรวดเร็ว

บริษัทใดใช้ภาษา Go บ้าง

ที่มา : https://go.dev/

ใช้ทำอะไรได้บ้าง (Use Case)

- Web Development (Backend)
- DevOps Automation
- Cloud Computing Systems
- Command Line Interface Tools



ที่มา : https://go.dev/

ข้อดีของภาษา Go

- เป็นภาษาที่ทำงานเร็ว
- มีโครงสร้างไวยากรณ์ภาษาที่เข้าใจง่าย
- เหมาะสำหรับงานที่ต้องรองรับ Request เป็นจำนวนมาก
- มีไลบรารี่ที่ครอบคลุมการใช้งานในแอพพลิเคชั่นในยุคปัจจุบัน สามารถศึกษาเพิ่มเติมได้ที่ https://pkg.go.dev/
- ทำ Concurrent Programming และ Multithreading ได้

Break

ติดตั้ง Go Compiler

- go version

ติดตั้ง VSCode

- Visual Studio Code (Text Editor)
- Go (Extension)
- Code Runner (Extension)
- Error Lens (Extension)

สร้างโปรเจกต์

- สร้าง Go Modules ขึ้นมา (go.mod) สำหรับจัดการโมดูลที่ใช้ในโปรเจกต์พร้อมระบุเวอร์ชั่นที่ใช้งาน

** โมดูลคือส่วนที่ใช้จัดการ Go Package ที่ใช้งานในโปรเจกต์ทั้งรูปแบบ Internal / External

```
go mod init ชื่อโมดูล / ชื่อโปรเจกต์
```

Break

โครงสร้างคำสั่งภาษา Go

- ไฟล์ที่เก็บโค้ดภาษา Go จะมีนามสกุลไฟล์ .go
- ฟังก์ชั่น main() คือ ฟังก์ชั่นพิเศษ กลุ่มคำสั่งที่อยู่ในฟังก์ชั่นนี้จะถูกสั่งให้ทำงานโดยอัตโนมัติเป็นลำดับแรกเสมอ
- ขอบเขต (Block) ใช้สัญลักษณ์ {} เพื่อบอกขอบเขตการทำงานของกลุ่มคำสั่งว่ามีจุดเริ่มต้นและสิ้นสุดที่ตำแหน่งใด
- คำอธิบาย (Comment) ใช้สัญลักษณ์ // หรือ /* */

โครงสร้างคำสั่งภาษา Go

```
package main
import “fmt”
func main()
{
    fmt.Println(“Hello World”);
}
```

โครงสร้างคำสั่งภาษา Go

import “fmt”

นำคำสั่งพื้นฐานที่อยู่ Package ที่ชื่อว่า fmt เข้ามาทำงาน เช่น กลุ่มคำสั่งที่ต้องการแสดงผลออกทางจอภาพ เป็นต้น

โครงสร้างคำสั่งภาษา Go

```
package main
import “fmt”
func main()
{
    fmt.Println(“Hello World”);
}
```

โครงสร้างคำสั่งภาษา Go

```
package main
import “fmt”
func main()
{
    fmt.Println(“Hello World”);
}
```

แพกเกจที่รวมเครื่องมือจัดรูปแบบ input และ output

โครงสร้างคำสั่งภาษา Go

```
package main
import “fmt”
func main()
{
    fmt.Println(”Hello World”);
}
```

แพกเกจที่รวมเครื่องมือจัดรูปแบบ input และ output

Println คือ ฟังก์ชั่นสำหรับแสดงผลข้อความแล้วขึ้นบรรทัดใหม่

หมายเหตุ (Comment)

จุดประสงค์

- อธิบายหน้าที่หรือความหมายของโค้ดที่เขียน
- ยกเลิกโค้ดชั่วคราว ส่งผลให้ตัวแปลภาษาไม่สนใจโค้ดในบรรทัดที่ถูกทำหมายเหตุ

หมายเหตุ (Comment)

วิธีที่ 1 โดยใช้เครื่องหมาย Slash (/) ใช้ในการอธิบายคำสั่งสั้นๆ ในรูปแบบบรรทัดเดียว

วิธีที่ 2 เขียนคำอธิบายไว้ในเครื่องหมาย /* ... */ ใช้ในการอธิบายคำสั่งยาวๆหรือแบบหลายบรรทัด

Break

ตัวแปรและชนิดข้อมูล

ตัวแปร (variable) คือ **ชื่อที่ถูกนิยามขึ้นมา**เพื่อใช้เก็บค่าข้อมูลสำหรับนำไปใช้งานในโปรแกรม โดยข้อมูลประกอบด้วย ข้อความ ตัวเลข หรือผลลัพธ์จากการประมวลผลข้อมูลค่าที่เก็บในตัวแปร **สามารถเปลี่ยนแปลงค่าได้**

ชนิดข้อมูลพื้นฐาน (Data Type)

ชนิดข้อมูล	คำอธิบาย	Zero Value
bool	ค่าทางตรรกศาสตร์ (true/false)	false
int int8 int16 int32 int64 uint uint8 uint16 uint32 uint64 uintptr	ตัวเลขที่ไม่มีจุดทศนิยม	0
float32 float64	ตัวเลขที่มีจุดทศนิยม	0
string	ชุดข้อความ	“ ”

ยิ่งจำนวนของ bit มากเท่าไร แสดงว่าเราสามารถเก็บค่าได้มากเท่านั้น

Break

การนิยามตัวแปร

ภาษา Go เป็นรูปแบบ Static-Type หมายถึง **ต้องประกาศตัวแปรและชนิดข้อมูลก่อนใช้งาน** โดยแบ่งการนิยามตัวแปรออกเป็น 2 รูปแบบ

- Manual Type Declaration
- Type Inference

Manual Type Declaration

คือ ประกาศตัวแปรพร้อมระบุชนิดข้อมูล

โครงสร้างคำสั่ง

```
var <ชื่อตัวแปร> <ชนิดข้อมูล>
```

Example

```
var name string
name = "KongRuksiam"
```

Type Inference

คือ ประกาศตัวแปรโดยไม่ต้องระบุชนิดข้อมูล

โครงสร้างคำสั่ง

<ชื่อตัวแปร> := <value>

Example

name :=”KongRuksiam”

ตัวอย่างการสร้างตัวแปร

```
name := "KongRuksiam"

age := 25
```

ค่าคงที่ (Constant)

มีลักษณะการการใช้งานคล้ายกับตัวแปร แต่ค่าคงที่คือค่าจะไม่สามารถเปลี่ยนแปลงได้ ตอนประกาศใช้งานค่าคงที่ต้องมีการประกาศค่าเริ่มต้นเสมอ

การนิยามค่าคงที่ (Constant)

โครงสร้างคำสั่ง

```
const <ชื่อตัวแปร> <ชนิดข้อมูล> = value
```

Example

```
const name string = "KongRuksiam"
```

Break

แสดงผลชนิดข้อมูล

```
import “fmt”

name := ”KongRuksiam”

fmt.Printf(“My Name is %v \n”,name) //value

fmt.Printf(“Data Type =  %T \n”,name) //type
```

Break

ตัวดำเนินการทางคณิตศาสตร์

Operator	คำอธิบาย
+	บวก
-	ลบ
*	คูณ
/	หาร
%	หารเอาเศษ

ตัวดำเนินการเปรียบเทียบ

Operator	คำอธิบาย
==	เท่ากับ
!=	ไม่เท่ากับ
>	มากกว่า
<	น้อยกว่า
>=	มากกว่าเท่ากับ
<=	น้อยกว่าเท่ากับ

รับค่าจากคีย์บอร์ดด้วย Scanf

โครงสร้างคำสั่ง

fmt.Scanf(string_format, address_list)

- string_format คือ รูปแบบตัวแทนชนิดข้อมูล
- address_list คือ ตัวเก็บข้อมูล

ตัวอย่าง String Format

ชนิดข้อมูล	ตัวแทนชนิดข้อมูล
string	%s
integer	%d
floating point	%f

Break

โครงสร้างควบคุมแบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- Switch..Case

รูปแบบคำสั่งแบบเงื่อนไขเดียว

If Statement

เป็นคำสั่งที่ใช้กำหนดเงื่อนไขในการตัดสินใจทำงานของโปรแกรม
ถ้าเงื่อนไขเป็นจริงจะทำตามคำสั่งต่างๆ ที่กำหนดภายใต้เงื่อนไขนั้นๆ

รูปแบบคำสั่งแบบเงื่อนไขเดียว

```
if เงื่อนไข { //true
    คำสั่งเมื่อเงื่อนไขเป็นจริง ;
}
```

If...Else Statement

```
if เงื่อนไข {
    คำสั่งเมื่อเงื่อนไขเป็นจริง
}else{
    คำสั่งเมื่อเงื่อนไขเป็นเท็จ
}
```

โจทย์ปัญหา

คำนวณคะแนนสอบวิชาคอมพิวเตอร์ของนักเรียนในห้องโดย มีคะแนนเต็ม 100 คะแนน ต้องการอยากทราบว่านักเรียนคนใดสอบผ่านบ้าง ใช้เกณฑ์ดังนี้ คือ

- คะแนนตั้งแต่ 50 คะแนนขึ้นไป => สอบผ่าน
- คะแนนน้อยกว่า 50 คะแนน => สอบไม่ผ่าน

โจทย์ปัญหา : เลือกใช้บริการระบบธนาคาร

ให้ป้อนหมายเลขตัวเลือกเพื่อใช้บริการ

- ถ้าป้อนเลข 1 คือ เปิดบัญชีธนาคาร
- ถ้าป้อนเลข 2 คือ ถอน-ฝากเงิน
- ถ้าพิมพ์ตัวเลขอื่น แจ้งว่าข้อมูลไม่ถูกต้อง

หาเลขคู่ - เลขคี่

โครงสร้างควบคุมแบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- Switch..Case

โครงสร้างควบคุมแบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- Switch..Case

Switch..Case

Switch เป็นคำสั่งที่ใช้กำหนดเงื่อนไขคล้ายๆกับ if แต่จะเลือกเพียงหนึ่งทางเลือกออกมาทำงานโดยนำค่าในตัวแปรมากำหนดเป็นทางเลือกผ่านคำสั่ง case (ตัวแปรควบคุม)

รูปแบบคำสั่ง

```
switch ค่าที่เก็บในตัวแปรควบคุม {
    case ค่าที่ 1 : คำสั่งที่ 1
    case ค่าที่ 2 : คำสั่งที่ 2
    case ค่าที่ N : คำสั่งที่ N
    default : คำสั่งเมื่อไม่มีค่าที่ตรงกับที่ระบุใน case
}
```

โจทย์ปัญหา : เลือกใช้บริการระบบธนาคาร

ให้ป้อนหมายเลขตัวเลือกเพื่อใช้บริการ

- ถ้าป้อนเลข 1 คือ เปิดบัญชีธนาคาร
- ถ้าป้อนเลข 2 คือ ถอน-ฝากเงิน
- ถ้าพิมพ์ตัวเลขอื่น แจ้งว่าข้อมูลไม่ถูกต้อง

อาร์เรย์ (Array)

การประกาศตัวแปรแต่ละครั้ง

ตัวแปร 1 ตัวสามารถเก็บข้อมูลได้แค่ 1 ค่าเท่านั้น เช่น

```
var number1 int = 1;
```

ถ้าอยากเก็บเลข 10 ค่าต้องทำอย่างไร ?

ต้องประกาศตัวแปร 10 ตัวแปร หรือไม่ ?

อาร์เรย์คืออะไร

1. ชุดของตัวแปรที่อยู่ในรูปลำดับใช้เก็บค่าข้อมูลให้อยู่ในกลุ่มเดียวกัน โดยข้อมูลภายในอาร์เรย์จะถูกเก็บในตำแหน่งที่ต่อเนื่องกัน

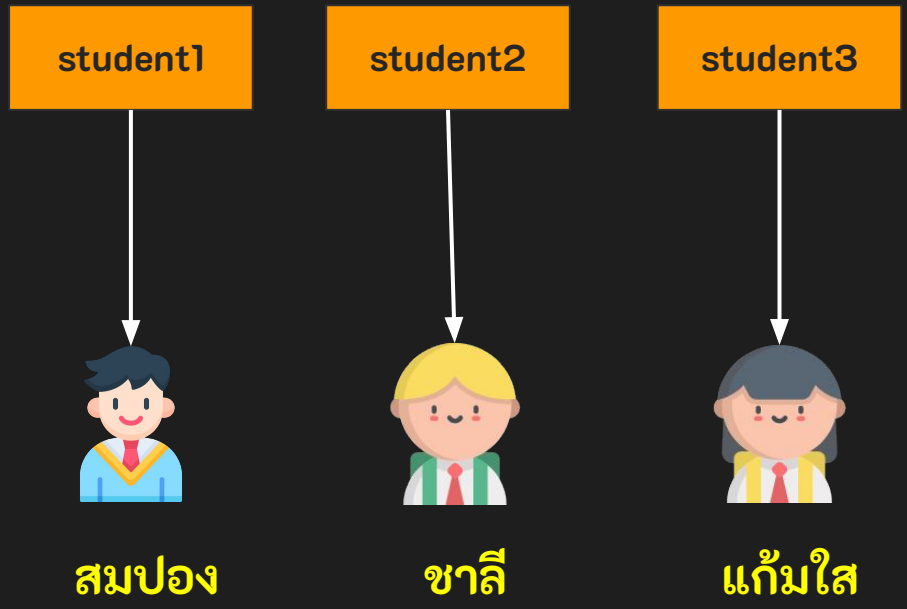
2. เป็นตัวแปรที่ใช้ในการเก็บข้อมูลที่มีลำดับที่ต่อเนื่อง ซึ่งข้อมูลมีค่าได้หลายค่าโดยใช้ชื่ออ้างอิงได้เพียงชื่อเดียว และใช้หมายเลขกำกับ (index) ให้กับตัวแปรเพื่อจำแนกความแตกต่างของค่าตัวแปรแต่ละตัว

คุณสมบัติของอาร์เรย์

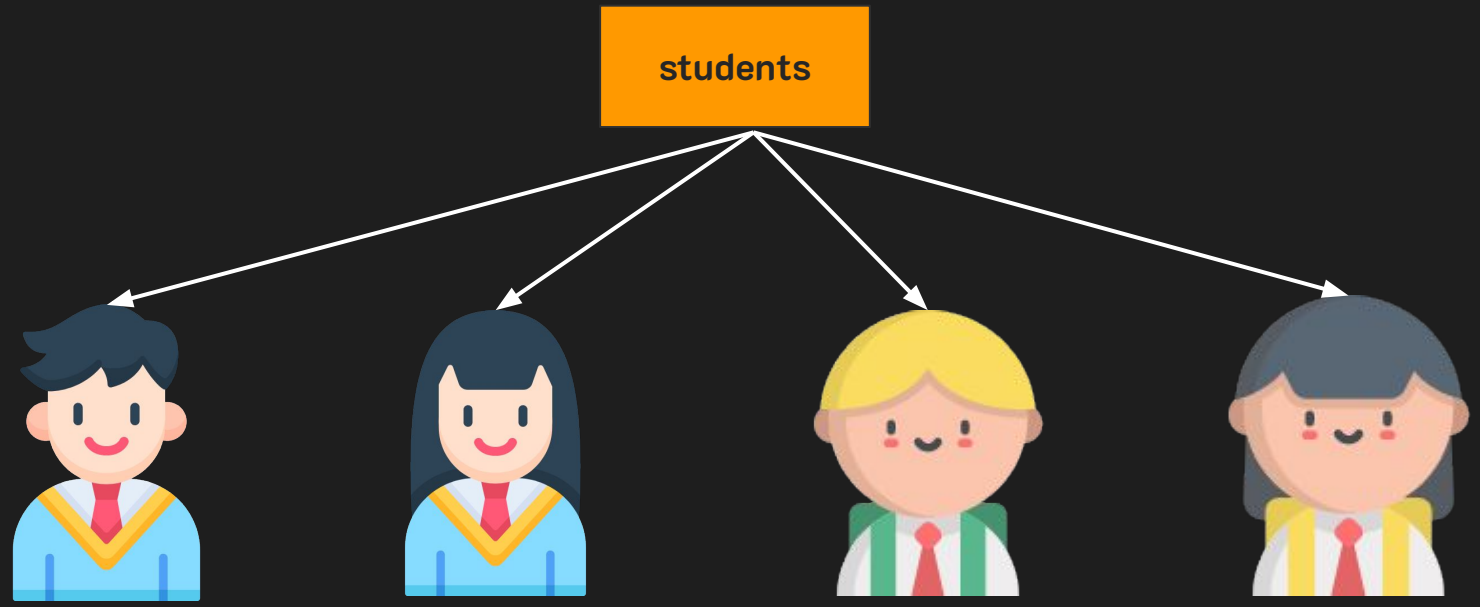
1. ใช้เก็บกลุ่มของข้อมูล
2. ข้อมูลที่อยู่ในอาร์เรย์จะเรียกว่าสมาชิก หรือ อิลิเมนต์ (element)
3. แต่ละอิลิเมนต์ (element) จะเก็บค่าข้อมูล (value) และ อินเด็กซ์ (Index) เอาไว้
4. Index หมายถึงคีย์ของอาร์เรย์ใช้อ้างอิงตำแหน่งของ element เริ่มต้นที่ 0
5. สมาชิกใน array ต้องมีชนิดข้อมูลเหมือนกัน
6. สมาชิกใน array จะถูกคั่นด้วยเครื่องหมาย comma

ตัวแปรแบบปกติ

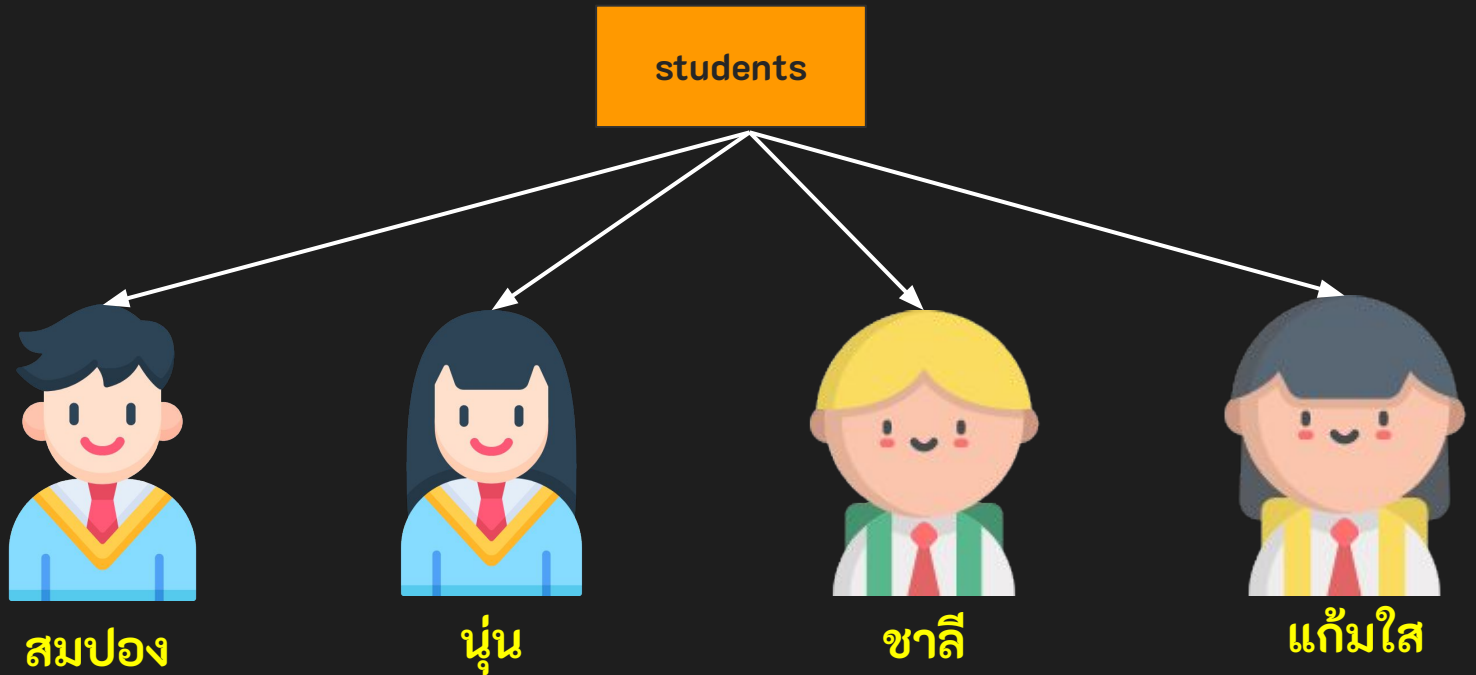
```
var student1 = “สมปอง”
var student2 = “ชาลี”
var student3 = “แก้มใส”
```



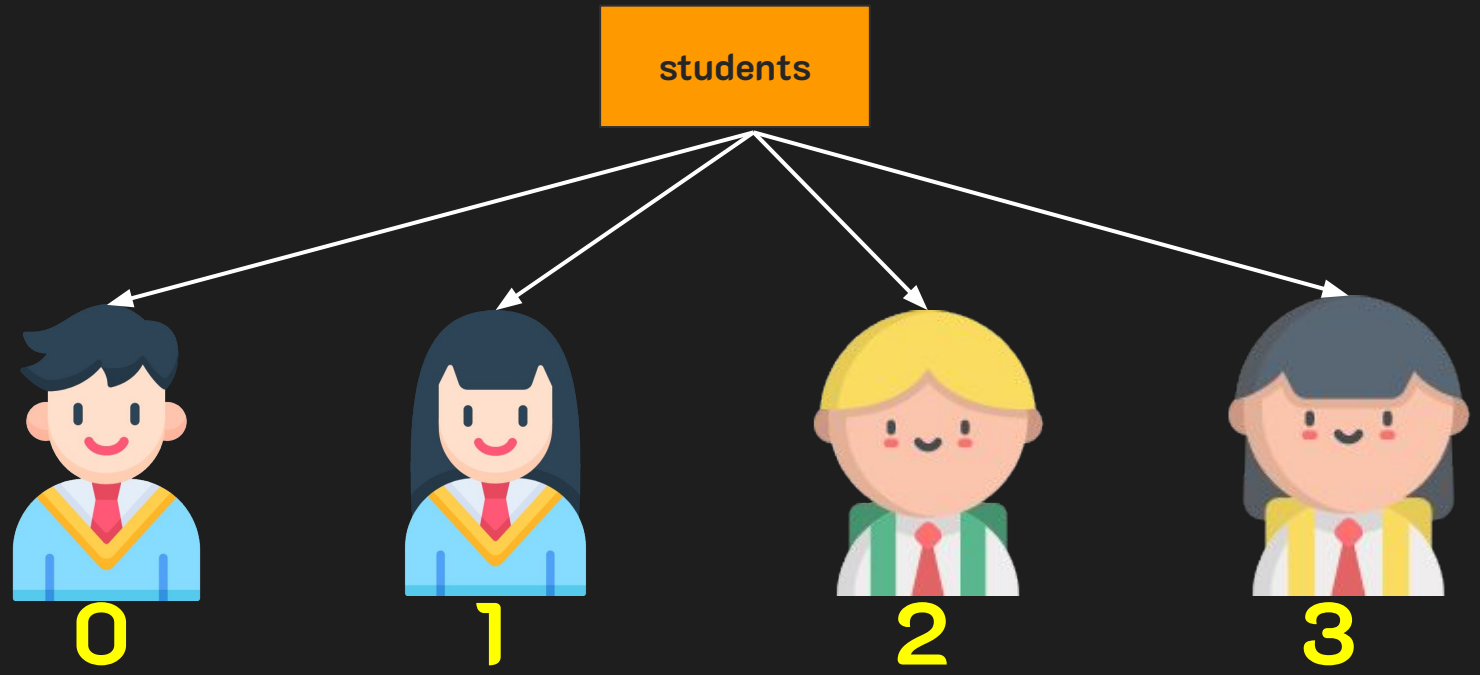
ตัวแปรแบบอาร์เรย์



ตัวแปรแบบอาร์เรย์

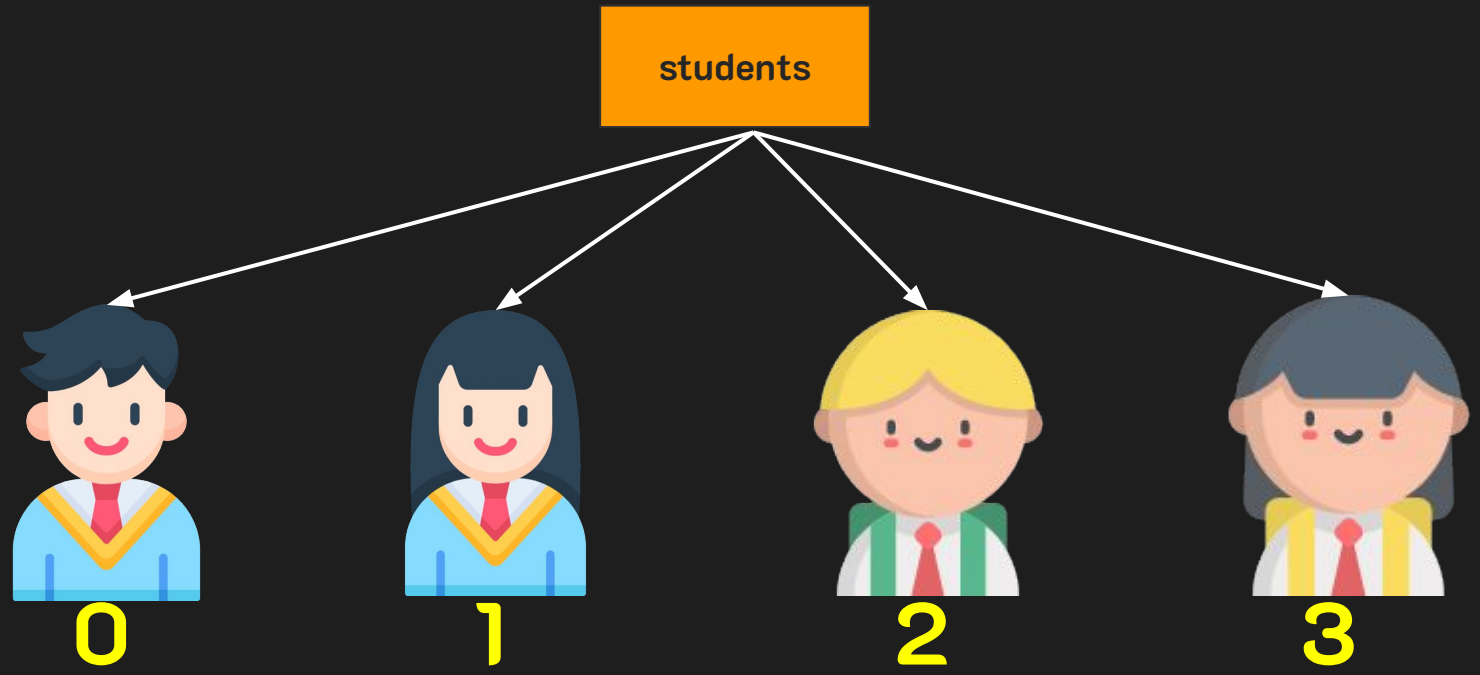


ตัวแปรแบบอาร์เรย์



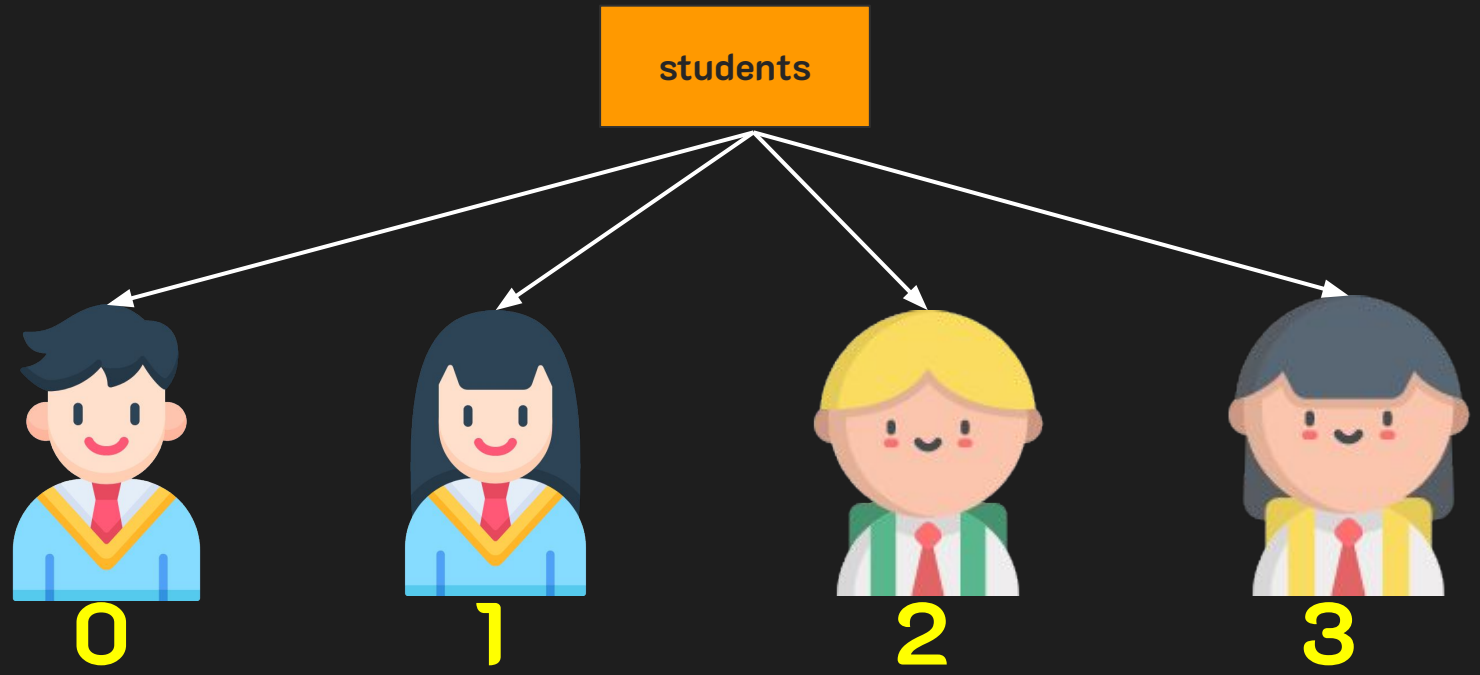
ใช้หมายเลขกำกับอ้างอิงข้อมูล (เป็นลำดับ)

ตัวแปรแบบอาร์เรย์



มีขนาดที่แน่นอน (Fixed Size)

ตัวแปรแบบอาร์เรย์



ใช้จัดเก็บกลุ่มข้อมูลที่มีชนิดข้อมูลเหมือนกัน

สรุปอาร์เรย์

1. ใช้เก็บกลุ่มของข้อมูล ที่มีชนิดข้อมูลเดียวกัน
2. ใช้ตัวแปรชื่อเดียวกัน
3. ใช้หมายเลขกำกับเพื่ออ้างอิงตำแหน่งของข้อมูลในอาร์เรย์
4. มีขนาดที่แน่นอนไม่สามารถปรับเปลี่ยนขนาดได้

BREAK!

รูปแบบการนิยามอาร์เรย์

1. แบบกำหนดจำนวนสมาชิก (ระบุตัวเลข)
2. ไม่กำหนดจำนวนสมาชิก (ไม่ระบุตัวเลข)

การนิยามอาร์เรย์แบบกำหนดขนาดเริ่มต้น

```
var numbers [4] int = [4] int{10, 20, 30, 40}
```


```
names :=[2] string {“สมชาย”,”แก้วตา”}
```

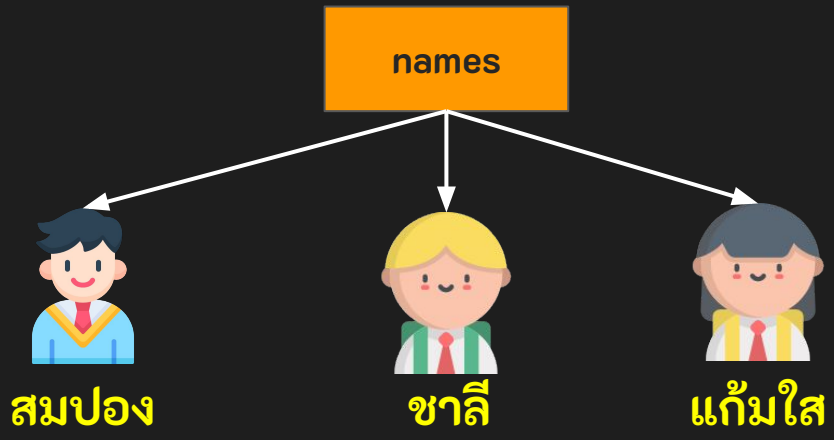
การเข้าถึงสมาชิกอาร์เรย์

```
var numbers [4] int = [4] int{10, 20, 30, 40}
```


```
names :=[2] string {“สมชาย”,”แก้วตา”}
```

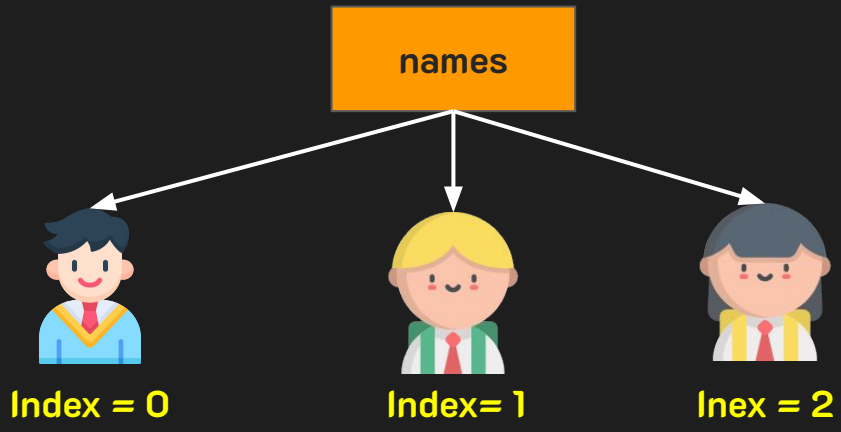
ตัวอย่างที่ 1

```
names :=[3] string{“สมปอง”,”ชาลี”,”แก้มใส”}
```



ตัวอย่างที่ 1

names :=[3] string{“สมปอง”,”ชาลี”,”แก้มใส”}



จำนวนสมาชิกในอาร์เรย์

```
numbers := int [4]{10, 20, 30, 40}
```

นับจำนวนสมาชิกในอาร์เรย์

```
numbers := int [4]{10, 20, 30, 40}
len(numbers)
pets:=string[2]{“แมว”,”กระต่าย”}
len(pets)
```

Slices

Slices

มีลักษณะคล้าย Array แต่มีความสามารถในการปรับเปลี่ยนขนาดสมาชิกได้ (Dynamic Size)

การนิยาม Slices

names :=[] string {“สมชาย”,”แก้วตา”}

สมชาย	แก้วตา

เพิ่มสมาชิกลงใน Slice

```
names :=[] string {“สมชาย”,”แก้วตา”}
names = append(names,”สมปอง”)
names = append(names,”ลูกน้ำ”)
```

การเข้าถึงสมาชิกใน Slices

names := [] string {“สมชาย”,”แก้วตา”,”ลูกน้ำ”}


```
fmt.Println(names[0])
fmt.Println(names[1])
```

การเข้าถึงสมาชิกแบบกำหนดช่วง

names := [] string {“สมชาย”,”แก้วตา”,”ลูกน้ำ”}

สมชาย (0)	แก้วตา (1)	ลูกน้ำ(2)

โครงสร้างคำสั่ง : slice[low:high]

- names[1:] // index ที่ 1 ถึงสุดท้าย
- name[:1] // index ที่ 0 - 1

Maps

Maps คืออะไร

ตัวแปรที่เก็บข้อมูลในรูปแบบคู่ key , value มีลักษณะคล้ายกับอาร์เรย์ แต่จะใช้ key เป็น index เพื่อเชื่อมโยงข้อมูล (value) ที่เก็บใน key นั้นๆ ถ้าทราบ key ก็สามารถเข้าถึง value หรือข้อมูลได้นั่นเอง

Array

```
country := [2] string{“Thailand”,”England”}
```

ใช้เลข index ที่เป็นตัวเลขจำนวนเต็มในการอ้างอิงข้อมูล

การนิยาม Maps

var ชื่อตัวแปร map [key_type]value_type

var country map [string] string

Maps

```
country:=map [string] string { }
country[“TH”] = “Thailand”;
country[“EN”] = “England”;
```

TH=>Thailand	EN=>England

Maps

```
coin:=map [string] string { }
coin["ETH"] = "Ether";
coin["BTC"] = "Bitcoin";
```

ETH=>Ether	BTC=>Bitcoin

Maps

```
population:=map [string] int { }
population[“Thailand”] = 70;
population[“England”] = 55;
```

Thailand => 70	England => 55

Maps แบบบรรทัดเดียว

```
coins:=map [string] string {“ETH”:“Ether” , “BTC” : “Bitcoin”}
```

ETH=>Ether	BTC=>Bitcoin

ดึงค่าจาก Map

```
coins:=map [string] string {“ETH”:“Ether” , “BTC” : “Bitcoin”}
```

โครงสร้างคำสั่ง

```
value , check :=map[key]
```

- value คือ ค่าจาก map
- check คือ bool สำหรับตรวจสอบว่ามีคีย์นี้ใน map หรือไม่

BREAK!

For Loop

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/

โครงสร้างควบคุมแบบทำซ้ำ (Loop)

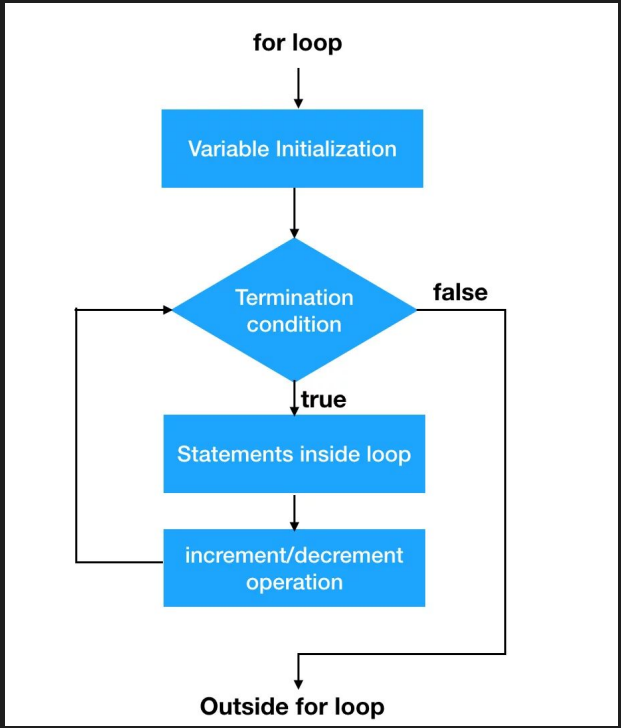
กลุ่มคำสั่งที่ใช้ในการวนรอบ (loop) โปรแกรมจะทำงานไปเรื่อยๆ จนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- For Loop

For Loop

เป็นรูปแบบการซ้ำที่ใช้ในการตรวจสอบเงื่อนไขการทำงาน มีการกำหนดค่าเริ่มต้นและเปลี่ยนค่าไปพร้อมๆกัน เมื่อเงื่อนไขในคำสั่ง for เป็นจริงก็จะทำงานตามคำสั่งที่แสดงไว้ภายในคำสั่ง for ไปเรื่อยๆ

```
for ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร {
        คำสั่งเมื่อเงื่อนไขเป็นจริง;
}
```



https://cdn.journaldev.com/wp-content/uploads/2017/10/java-for-loop.png.webp

1. กำหนดค่าเริ่มต้น
2. เช็คเงื่อนไขถ้าเป็นจริงให้ทำคำสั่งซ้ำใน Statement
3. ถ้าเป็นเท็จให้ออกจาก Loop

```
for ค่าเริ่มต้นของตัวแปร; เงื่อนไข; เปลี่ยนแปลงค่าตัวแปร{
        fmt.Println(“Hello Go Programming”)
}
```

Output

แสดงข้อความ “Hello Go Programming” จำนวน 3 ครั้ง

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

Output

แสดงข้อความ “Hello Go Programming” จำนวน 3 ครั้ง

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming , count = 2
- Hello Go Programming

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming , count = 2
- Hello Go Programming

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3
- Hello Go Programming

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3
- Hello Go Programming

```
for count :=1; count <=3;count++ {
        fmt.Println(“Hello Go Programming”)
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3
- Hello Go Programming , count = 4

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3
- Hello Go Programming , count = 4

```
for count :=1; count <=3;count++ {
        fmt.Println("Hello Go Programming")
}
```

- Hello Go Programming , count = 2
- Hello Go Programming , count = 3
- Hello Go Programming , count = 4

คำสั่งที่เกี่ยวข้องกับ Loop

- **break** ถ้าโปรแกรมพบคำสั่งนี้จะหลุดจากการทำงานในลูปทันที เพื่อไปทำคำสั่งอื่นที่อยู่นอกลูป

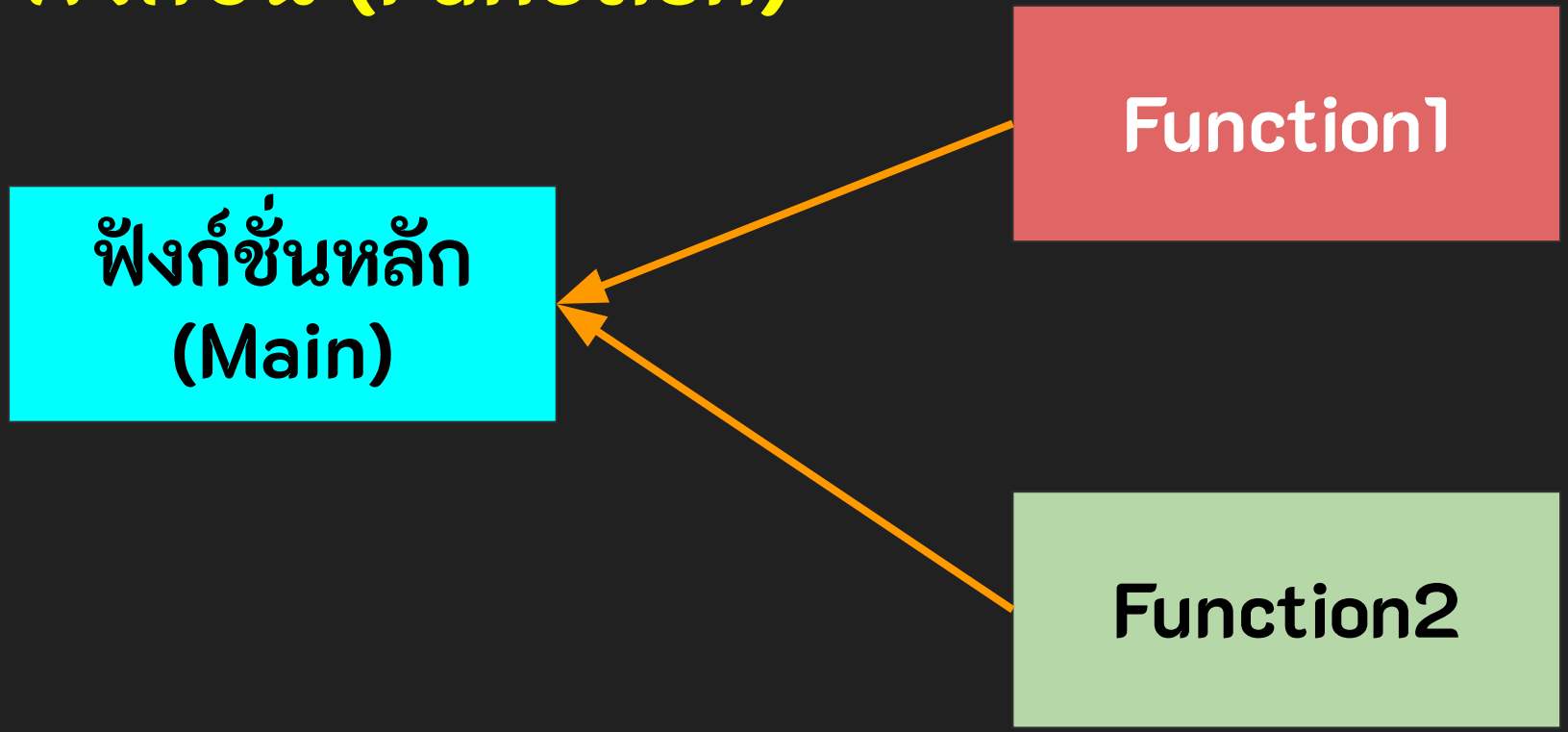
- **continue** คำสั่งนี้จะทำให้หยุดการทำงานแล้วย้อนกลับไปเริ่มต้นการทำงานที่ต้นลูปใหม่

Break

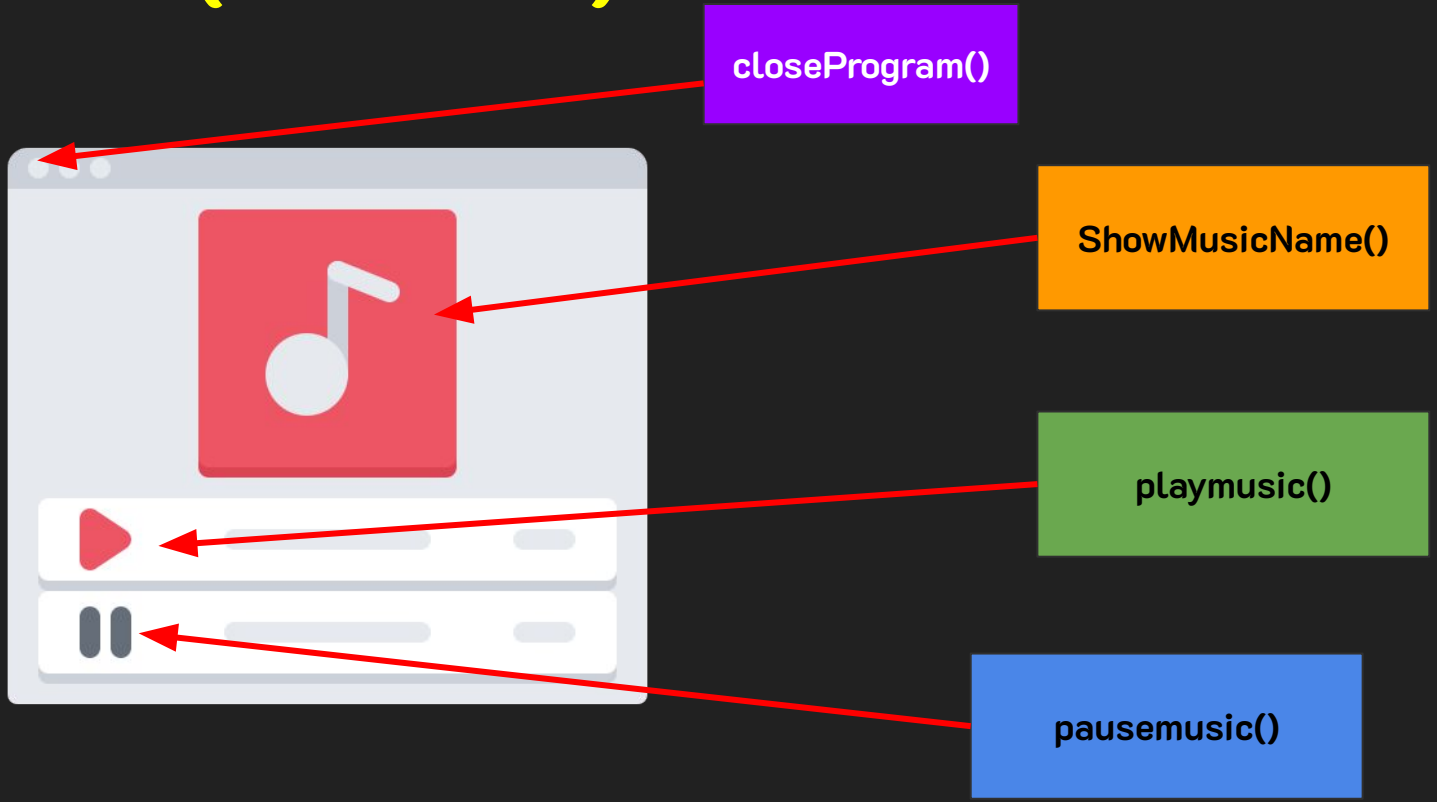
ฟังก์ชั่น (Function) คืออะไร

ชุดคำสั่งที่นำมาเขียนรวมกันเป็นกลุ่มเพื่อให้เรียกใช้งานตามวัตถุประสงค์ที่ต้องการและลดความซ้ำซ้อนของคำสั่งที่ใช้งานบ่อยๆ ฟังก์ชั่นสามารถนำไปใช้งานได้ทุกที่และแก้ไขได้ในภายหลัง ทำให้โค้ดในโปรแกรมมีระเบียบและใช้งานได้สะดวกมากยิ่งขึ้น

ฟังก์ชั่น (Function)



ฟังก์ชั่น (Function)



รูปแบบของฟังก์ชั่น

ฟังก์ชั่นที่ไม่มีการรับและส่งค่า

```
func ชื่อฟังก์ชั่น(){
    // คำสั่งต่างๆ
}
```

การเรียกใช้งานฟังก์ชั่น

```
ชื่อฟังก์ชั่น ();
```

รูปแบบของฟังก์ชั่น

ฟังก์ชั่นที่มีการรับค่าเข้ามาทำงาน

```
func ชื่อฟังก์ชั่น(parameter1,parameter2,.....){
// กลุ่มคำสั่งต่างๆ
}
```

- อาร์กิวเมนต์ คือ ตัวแปรหรือค่าที่ต้องการส่งมาให้กับฟังก์ชัน
- พารามิเตอร์ คือ ตัวแปรที่ฟังก์ชันสร้างไว้สำหรับรับค่าที่จะส่งเข้ามาให้กับฟังก์ชัน

การเรียกใช้งานฟังก์ชั่น

```
ชื่อฟังก์ชั่น (argument1,argument2,.....)
```

รูปแบบของฟังก์ชั่น

ฟังก์ชั่นที่มีส่งค่าออกมา

```
func ชื่อฟังก์ชั่น(){
    return ค่าที่จะส่งออกไป
}
```

รูปแบบของฟังก์ชั่น

ฟังก์ชั่นที่มีการรับค่าเข้ามาและส่งค่าออกไป

```
func ชื่อฟังก์ชั่น(parameter1,parameter2,.....){
    return ค่าที่จะส่งออกไป
}
```

Variadic Function

```
func ชื่อฟังก์ชั่น(parameter.....type){

    //statement

}
```

Variadic Function

```
func ชื่อฟังก์ชั่น(parameter.....type){

    //statement

}
```

Struct

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/

สตรัคเจอร์ (Structure)

ข้อจำกัดของ Array ในกรณีที่มีการเก็บข้อมูลลงไปใน Array สมาชิกทุกตัวที่อยู่ใน Array ต้องมีชนิดข้อมูลเหมือนกัน

แล้วถ้าต้องการอยากเก็บข้อมูลที่มีชนิดข้อมูลต่างกันจะทำอย่างไร ?

สตรัคเจอร์ (Structure)

คือ ข้อมูลแบบโครงสร้างที่นำเอาข้อมูลที่มีชนิดข้อมูลต่างกันมารวบรวมเข้าด้วยกัน แต่มีความสัมพันธ์ของข้อมูลแบบต่อกัน มาเก็บไว้ภายในโครงสร้างเดียวกัน

เปรียบเสมือนกับสร้างชนิดข้อมูลขึ้นมาใช้งานเอง

การนิยามสตรัคเจอร์

```
type ชื่อสตรัคเจอร์ struct{
        ตัวแปรที่ 1 ชนิดข้อมูลตัวที่ 1 ;
        ตัวแปรที่ 2 ชนิดข้อมูลตัวที่ 2 ;
        .....
}
```

เก็บข้อมูลสินค้า (Product)

- ชื่อสินค้า (string)
- ราคา (float)
- หมวดหมู่ (string)
- ส่วนลด (int)

Package

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/

Package

คือ สิ่งที่ช่วยให้ผู้พัฒนาโปรแกรมสามารถจัดการโค้ดเป็นกลุ่มการทำงานได้ ด้วยวิธีการแยกไฟล์ออกเป็นส่วนๆตามรูปแบบการทำงาน

ซึ่งสามารถเรียกใช้ภายในส่วนต่างๆของโปรแกรมได้ ส่งผลให้โค้ดมีความเป็นระเบียบและนำกลับมาใช้งานใหม่ได้ง่ายและสะดวกมากยิ่งขึ้น

ตัวอย่าง Package

```
import “fmt”


fmt.Println(“Hello Go Prgramming”)
```

ตัวอย่าง Package

```
import “fmt”

fmt.Println(“Hello Go Prgramming”)
```

นำ Package มาทำงาน

ตัวอย่าง Package

```
import “fmt”
```

นำ Package มาทำงาน

เรียกใช้ Package

```
fmt.Println(“Hello Go Prgramming”)
```

ตัวอย่าง Package

```
import “fmt”
fmt.Println(“Hello Go Prgramming”)
```

นำ Package มาทำงาน

เรียกใช้ Package

เรียกใช้ฟังก์ชั่นใน Package