Dart เบื้องต้น

ปูพื้นฐานภาษา Dart ก่อนลุยโปรเจค Flutter

หัวข้อนี้จะอยู่ 3 Playlist สามารถเลือกเรียนได้

- ปูพื้นฐาน Dart ก่อนลุย Flutter (คลิปเดียวจบ)
- ปูพื้นฐาน Dart ก่อนลุย Flutter (หัวข้อย่อยใน Playlist Flutter)
- พัฒนา Mobile App ด้วย Flutter (Android / iOS) (คลิปเดียวจบ)

Dart เบื้องต้น

- เป็นภาษาโปรแกรมพัฒนาโดย Google
- โครงสร้างภาษาคล้าย Java C , C++
- มีความสามารถด้าน OOP คือ มี class และ inheritance ให้ใช้งาน

Dart เป็นภาษากลุ่ม Compiler นั่นคือจำเป็นต้องมี Compiler ก่อนถึงจะนำโปรแกรมไปรันได้ดังนั้นในหัวข้อนี้ให้ทำงานใน vscode และติดตั้ง code runner extension ลงไป

ถ้ามีพื้นฐาน C , Java หรือ JavaScript จะดีมากๆ

1.กรณีที่เรียนใน พัฒนา Mobile App ด้วย Flutter (Android / iOS)

- รันผ่าน Visual Studio Code โดยลง Extension Code Runner ต่อจาก Dart & Flutter

2.ต้องการเรียน Dart อย่างเดียวทดลองเขียนโปรแกรมภาษา Dart ได้ที่

- http://dartpad.dev

เมธอด Main

เป็นเมธอดที่เป็นจุดเริ่มต้นการทำงานของโค้ดเราอันดับแรกๆ

```
void main (){
// คำสั่งต่างๆ
}
```

การแสดงผลข้อมูล

```
void main (){
    print(“ข้อความที่ต้องการแสดงผล”);
}
```

การเขียนคำอธิบาย (Comment)

วิธีที่ 1 โดยใช้เครื่องหมาย Slash (/) ใช้ในการอธิบาย**คำสั่งสั้นๆในรูปแบบบรรทัดเดียว**

วิธีที่ 2 เขียนคำอธิบายไว้ในเครื่องหมาย /* ... */ ใช้ในการอธิบายคำสั่ง**ยาวๆหรือแบบหลายบรรทัด**

ตัวแปรและชนิดข้อมูล

ตัวแปร คือ **ชื่อที่ถูกนิยาม**ขึ้นมาเพื่อใช้เก็บค่าข้อมูลสำหรับนำไปใช้งานในโปรแกรม โดยข้อมูลอาจจะประกอบด้วย ข้อความ ตัวเลข ตัวอักษรหรือผลลัพธ์จากการประมวลผลข้อมูล

ชนิดข้อมูล	คำอธิบาย	รูปแบบข้อมูล
bool	ค่าทางตรรกศาสตร์	True /False
num	ตัวเลขที่ไม่มีจุดทศนิยม (int)	20
	ตัวเลขที่มีจุดทศนิยม (double)	30.15
string	ข้อความ	“kongruksiam”
List<type>	โครงสร้างข้อมูล	[”มะม่วง”, ”มะละกอ”, ”ส้ม”]
Map <index,value>	โครงสร้างข้อมูล	{firstName:”kong”, lastName:”ruksiam”, age:20};
Dynamic	ตัวแปรเปลี่ยนค่าได้	20,30,15,True,”kong”

รูปแบบตัวแปรในภาษา Dart

- **ตัวแปรแบบ Dynamic Typing** คือชนิดตัวแปรจะเป็นอะไรก็ได้ตามค่าที่ตัวมันเก็บโดยไม่ต้องประกาศชนิดข้อมูล

- **ตัวแปรแบบ Static Typing** ต้องประกาศชนิดข้อมูลในตอนเริ่มต้น เช่น int, double เพื่อบอกว่าตัวแปรนี้จะเก็บข้อมูลชนิดไหน

การนิยามตัวแปร (Static Type)

ชนิดข้อมูล ชื่อตัวแปร ;

ชนิดข้อมูล ชื่อตัวแปร = ค่าเริ่มต้น;

แบบหลายตัวแปรในบรรทัดเดียว

ชนิดข้อมูล ชื่อตัวแปร = ค่าเริ่มต้น,ชื่อตัวแปร = ค่าเริ่มต้น

***ตัวแปรที่ประกาศไว้แต่ยังไม่ได้กำหนดค่า จะมีค่าเป็น null โดยอัตโนมัติ

การนิยามตัวแปร (Dynamic Typing)

var (นิยามตัวแปรตามค่าที่กำหนดเริ่มต้น)

var ชื่อตัวแปร;

var ชื่อตัวแปร = ค่าเริ่มต้น;

var ชื่อตัวแปร = ค่าเริ่มต้น,ชื่อตัวแปร = ค่าเริ่มต้น

```
var money;
var money=100;
money= “200”; // error
```

***ตัวแปรที่ประกาศไว้แต่ยังไม่ได้กำหนดค่า จะมีค่าเป็น null โดยอัตโนมัติ

การนิยามตัวแปร (Dynamic Typing)

dymanic (นิยามตัวแปรตามค่าที่กำหนดเริ่มต้น)

dymanic ชื่อตัวแปร;

dymanic ชื่อตัวแปร = ค่าเริ่มต้น;

dynamic ชื่อตัวแปร = ค่าเริ่มต้น,ชื่อตัวแปร = ค่าเริ่มต้น

```
dynamic money;
dynamic money=100;
money= “200”; //ทำงานผ่าน
```

***ตัวแปรที่ประกาศไว้แต่ยังไม่ได้กำหนดค่า จะมีค่าเป็น null โดยอัตโนมัติ

สรุป Dynamic กับ Var ต่างกันยังไง

- dynamic ค่าที่อยู่ในตัวแปรสามารถเปลี่ยนค่าหรือชนิดข้อมูลได้เรื่อยๆ ไม่สามารถตรวจสอบชนิดข้อมูลในตัวแปรนั้นได้

- var ค่าที่อยู่ในตัวแปรสามารถเปลี่ยนค่าได้แต่เปลี่ยนชนิดข้อมูลไม่ได้ในภายหลังที่ประกาศตัวแปรออกไป

การนิยามค่าคงที่ Constant

```
const ชนิดข้อมูล  ชื่อตัวแปร ;

const ชนิดข้อมูล ชื่อตัวแปร = ค่าเริ่มต้น;

int x = 10;

const int y= x+20; // ไม่สามารถนำค่าจากตัวแปรอื่นมาคำนวณได้
```

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

การนิยามค่าคงที่ Final

final ชนิดข้อมูล ชื่อตัวแปร ;

final ชนิดข้อมูล ชื่อตัวแปร = ค่าเริ่มต้น;

```
int x = 10;

final int y= x+20; // สามารถนำค่าจากตัวแปรอื่นมาคำนวณได้
```

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

กฎการตั้งชื่อตัวแปร

- ประกอบด้วยตัวเลข ตัวอักษร เครื่องหมาย
- อักษรตัวแรกห้ามขึ้นต้นด้วยตัวเลขและสัญลักษณ์พิเศษ ยกเว้น _ (Underscore) และ $
- ห้ามซ้ำกับคำสงวน (Keyword)
- Case Sensitive

จัดการอักขระและข้อความด้วย string

การประกาศ string ขึ้นมาใช้ ต้องกำหนดเนื้อหาหรือค่าอยู่ในเครื่องหมาย ’ (single quote) หรือ ” (double quote)

```
var a = ’kongruksiam’;
string b = ”สอน dart เบื้องต้น”;
string c = ’basic to advance’;
```

จัดการอักขระและข้อความด้วย string

การทำงานระหว่าง number กับ String

```
int x = 10 , y =20
print(“ค่า x = ”+x.toString());
print(“ค่า x = $x”);
print(“ผลบวก = ${x+y}”);
```

ตัวดำเนินการ (Operator)

กลุ่มของเครื่องหมายหรือสัญลักษณ์ที่ใช้ในการเขียนโปรแกรม

A+B

1. ตัวดำเนินการ (Operator)
2. ตัวถูกดำเนินการ (Operand)

ตัวดำเนินการทางคณิตศาสตร์

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

จัดการตัวเลข (Number)

```
int x = 20; // integer
double y = 20.15; // double
double = int / int // ผลหาร int จะได้ double
```

จัดการตัวเลข (Number)

ทำให้เป็นจำนวนเต็ม

```
int  z = 10 / 5 ; // error
int z = 10 ~/5 ; หรือ
int z = (10 / 5).toInt();
```

ตัวดำเนินการเปรียบเทียบ

**** ชนิดข้อมูล boolean

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

ตัวดำเนินการเพิ่มค่า - ลดค่า

Operator	รูปแบบการเขียน	ความหมาย
++ (Prefix)	++a	เพิ่มค่าให้ a ก่อน 1 ค่าแล้วนำไปใช้
++ (Postfix)	a++	นำค่าปัจจุบันใน a ไปใช้ก่อนแล้วค่อยเพิ่มค่า
-- (Prefix)	--b	ลดค่าให้ b ก่อน 1 ค่าแล้วนำไปใช้
-- (Postfix)	b--	นำค่าปัจจุบันใน b ไปใช้ก่อนแล้วค่อยลดค่า

Compound Assignment

Assignment	รูปแบบการเขียน	ความหมาย
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x~/=y	x=x~/y
%=	x%=y	x=x%y

โครงสร้างควบคุม (Control Structure)

คือ กลุ่มคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- แบบลำดับ (Sequence)
- แบบมีเงื่อนไข (Condition)
- แบบทำซ้ำ (Loop)

แบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกเงื่อนไขต่างๆ ภายในโปรแกรมมาทำงาน

- if
- Switch..Case

รูปแบบคำสั่งแบบเงื่อนไขเดียว

- **if statement**

เป็นคำสั่งที่ใช้กำหนดเงื่อนไขในการตัดสินใจทำงานของโปรแกรม

ถ้าเงื่อนไขเป็นจริงจะทำตามคำสั่งต่างๆ ที่กำหนดภายใต้เงื่อนไขนั้นๆ

```
if(เงื่อนไข){
    คำสั่งเมื่อเงื่อนไขเป็นจริง ;
}
```

รูปแบบคำสั่งแบบ 2 เงื่อนไข

```
if(เงื่อนไข){
    คำสั่งเมื่อเงื่อนไขเป็นจริง ;
}else{
    คำสั่งเมื่อเงื่อนไขเป็นเท็จ ;
}
```

รูปแบบคำสั่งแบบหลายเงื่อนไข

```
if(เงื่อนไขที่ 1){
        คำสั่งเมื่อเงื่อนไขที่ 1 เป็นจริง ;
}else if(เงื่อนไขที่ 2){
        คำสั่งเมื่อเงื่อนไขที่ 2 เป็นจริง ;
}else if(เงื่อนไขที่ 3){
        คำสั่งเมื่อเงื่อนไขที่ 3 เป็นจริง ;
}else{
        คำสั่งเมื่อทุกเงื่อนไขเป็นเท็จ ;
}
```

ตัวดำเนินการทางตรรกศาสตร์

Operator	คำอธิบาย
&&	AND
||	OR
!	NOT

ตัวดำเนินการทางตรรกศาสตร์

a	!a	a	b	a && b	a || b
true	false	false	false	false	false
false	true	false	true	false	true
		true	false	false	true
		true	true	true	true

if..else แบบลดรูป (Ternary Operator)

ตัวแปร = (เงื่อนไข) ? คำสั่งเมื่อเงื่อนไขเป็นจริง : คำสั่งเมื่อเงื่อนไขเป็นเท็จ;

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

แบบมีเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกเงื่อนไขต่างๆ ภายในโปรแกรมมาทำงาน

- **Switch..Case**

Switch เป็นคำสั่งที่ใช้กำหนดเงื่อนไขคล้ายๆกับ if แต่จะเลือกเพียงหนึ่งทางเลือกออกมาทำงานโดยนำค่าในตัวแปรมากำหนดเป็นทางเลือกผ่านคำสั่ง case

รูปแบบคำสั่ง

```
switch(สิ่งที่ต้องการตรวจสอบ) {

        case ค่าที่ 1 : คำสั่งที่ 1;
                break;
         case ค่าที่ 2 : คำสั่งที่ 2;
                break;
                .....
         case ค่าที่ N : คำสั่งที่ N;
                break;

        default : คำสั่งเมื่อไม่มีค่าที่ตรงกับที่ระบุใน case
}
```

***คำสั่ง break จะทำให้โปรแกรมกระโดดออกไปทำงานนอกคำสั่ง switch ถ้าไม่มีคำสั่ง break โปรแกรมจะทำคำสั่งต่อไปเรื่อยๆ จนจบการทำงาน

แบบทำซ้ำ (Loop)

กลุ่มคำสั่งที่ใช้ในการวนรอบ (loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

คำสั่ง While

- **While Loop**

จะทำงานตามคำสั่งภายใน while ไปเรื่อยๆเมื่อเงื่อนไขที่กำหนดเป็นจริง

```
while(เงื่อนไข){
    คำสั่งที่จะทำซ้ำเมื่อเงื่อนไขเป็นจริง ;
}
```

คำสั่ง For

- **For Loop**

เป็นรูปแบบที่ใช้ในการตรวจสอบเงื่อนไข มีการกำหนดค่าเริ่มต้น และเปลี่ยนค่าไปพร้อมๆกัน เมื่อเงื่อนไขในคำสั่ง for เป็นจริงก็จะทำงานตามคำสั่งที่แสดงไว้ภายในคำสั่ง for ไปเรื่อยๆ

โครงสร้างคำสั่ง

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

```
for(var i = 1;i<=10;i++) {
    คำสั่งเมื่อเงื่อนไขเป็นจริง;
}
```

คำสั่ง Do..While

- **Do..While**

โปรแกรมจะทำงานตามคำสั่งอย่างน้อย 1 รอบ เมื่อทำงานเสร็จจะมาตรวจสอบเงื่อนไขที่คำสั่ง while ถ้าเงื่อนไขเป็นจริงจะวนกลับขึ้นไปทำงานที่คำสั่งใหม่อีกรอบ แต่ถ้าเป็นเท็จจะหลุดออกจากลูป

โครงสร้างคำสั่ง

```
do {
        คำสั่งต่างๆ เมื่อเงื่อนไขเป็นจริง;
 } while(เงื่อนไข);
```

คำสั่งที่เกี่ยวข้องกับ Loop

- **break** ถ้าโปรแกรมพบคำสั่งนี้จะหลุดจากการทำงานในลูปทันที เพื่อไปทำคำสั่งอื่นที่อยู่นอกลูป

- **continue** คำสั่งนี้จะทำให้หยุดการทำงานแล้วย้อนกลับไปเริ่มต้นการทำงานที่ต้นลูปใหม่

ข้อแตกต่างและการใช้งาน Loop

- For ใช้ในกรณีรู้จำนวนรอบที่ชัดเจน
- While ใช้ในกรณีที่ไม่รู้จำนวนรอบ
- Do..while ใช้ในกรณีที่อยากให้ลองทำก่อน 1 รอบ แล้วทำซ้ำไปเรื่อยๆทราบเท่าที่เงื่อนไขเป็นจริง

ฟังก์ชั่น คืออะไร

ความหมายที่ 1:

ชุดคำสั่งที่นำมาเขียนรวมกันเป็นกลุ่มเพื่อให้เรียกใช้งานตามวัตถุประสงค์ที่ต้องการ และลดความซ้ำซ้อนของคำสั่งที่ใช้งานบ่อยๆ ฟังก์ชั่นสามารถนำไปใช้งานได้ทุกที่และแก้ไขได้ในภายหลัง ทำให้โค้ดในโปรแกรมมีระเบียบและใช้งานได้สะดวกมากยิ่งขึ้น

ความหมายที่ 2 :

โปรแกรมย่อยที่นำเข้ามาเป็นส่วนหนึ่งของโปรแกรมหลัก เพื่อให้สามารถเรียกใช้งานได้โดยไม่จำเป็นต้องเขียนโค้ดคำสั่งใหม่ทั้งหมด

รูปแบบของฟังก์ชั่น

1.ฟังก์ชั่นที่ไม่มีการรับและส่งค่า

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

การเรียกใช้งานฟังก์ชั่น

```
ชื่อฟังก์ชั่น ();
```

รูปแบบของฟังก์ชั่น

2.ฟังก์ชั่นที่มีการรับค่าเข้ามาทำงาน

```
void ชื่อฟังก์ชั่น(parameter1,parameter2,.....){

// กลุ่มคำสั่งต่างๆ

}
```

อาร์กิวเมนต์ คือ ตัวแปรหรือค่าที่ต้องการส่งมาให้กับฟังก์ชัน (ตัวแปรส่ง)
พารามิเตอร์ คือ ตัวแปรที่ฟังก์ชันสร้างไว้สำหรับรับค่าที่จะส่งเข้ามาให้กับฟังก์ชัน (ตัวแปรรับ)

การเรียกใช้งานฟังก์ชั่น

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

รูปแบบของฟังก์ชั่น

3.ฟังก์ชั่นที่มีส่งค่าออกมา

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

รูปแบบของฟังก์ชั่น

4.ฟังก์ชั่นที่มีการรับค่าเข้ามาและส่งค่าออกไป

```
type ชื่อฟังก์ชั่น(parameter1,parameter2,.....){
        retrun ค่าที่จะส่งออกไป
}
```

ขอบเขตตัวแปร

- **local variable** ตัวแปรที่ทำงานอยู่ในฟังก์ชั่นมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของฟังก์ชั่น

- **global variable** ตัวแปรที่ทำงานอยู่นอกฟังก์ชั่นมีขอบเขตการทำงานตั้งแต่จุดเริ่มต้นไปจนถึงจุดสิ้นสุดของไฟล์ที่ประกาศใช้

Arrow Function

การเขียน Arrow ฟังก์ชั่นจะคล้ายๆกับใน JavaScript ใช้ลดรูปการเขียนฟังก์ชั่นแบบเดิมโดยใช้ => (arrow) ให้มีความสั้นกระชับมากยิ่งขึ้น

Arrow Function

แบบเดิม

```
String getName(){
    return “KongRuksiam”;
}
int getBonus(){
    return 500;
}
```

แบบ Arrow

```
String getName()=>“KongRuksiam”
int getBonus()=>500
```

Arrow Function

แบบเดิม

```
int plus(int x, int y) {

    return x + y;

}
```

แบบ Arrow

```
  plus(x, y) => x + y;
```

Arrow Function

แบบเดิม

```
void show(var name) {
      print(name);
}
```

แบบ Arrow

```
show(name)=> print(name);
```

ฟังก์ชั่นแบบกำหนดค่าเริ่มต้น (Optional Parameter)

```
ชื่อฟังก์ชั่น (String name , [String city = “กรุงเทพมหานคร”]){

    // คำสั่งต่างๆ

}
```

Named Parameter (กำหนดชื่อและลำดับ Parameter)

```
type ชื่อฟังก์ชั่น ({String fname , String lname,int age}){

   print($fname $lname,$age)

}
```

```
int age =20;
String fname =”kong”,lname=”ruksiam”;
ชื่อฟังก์ชั่น (fname:fname,age:age,lname:lname)
```

First-Class Function

การทำให้ฟังก์ชั่นกลายเป็นตัวแปรประเภทฟังก์ชั่นได้ หลักการเหมือนการสืบทอดคุณสมบัติในการเขียนโปรแกรมเชิงวัตถุ (OOP)

```
int getAge()=> 20;
void main(){

    var myAge = getAge;
     print(myAge())

}
```

List คืออะไร

ความหมายที่ 1 เป็นโครงสร้างข้อมูลหรือตัวแปรที่ใช้ในการเก็บข้อมูลและข้อมูลมีค่าได้หลายค่าโดยใช้ชื่ออ้างอิงได้เพียงชื่อเดียวและใช้หมายเลขกำกับ (index) โดยมีค่าเริ่มต้นคือ 0 ให้กับตัวแปรเพื่อจำแนกความแตกต่างของค่าตัวแปรแต่ละตัว

List คืออะไร

ความหมายที่ 2 ชุดของตัวแปรที่อยู่ในรูปลำดับใช้เก็บค่าข้อมูลให้อยู่ในกลุ่มเดียวกันหรือต่างชนิดกันก็ได้โดยถ้าอยากให้จัดกลุ่มข้อมูลเดียวกันให้กำหนดชนิดข้อมูลได้ภายใน <> เพื่อบอกว่า List นี้เก็บกลุ่มข้อมูลอะไร (Type Interface) หรือ Generic เพื่อไม่ให้เพิ่มชนิดข้อมูลอื่นเข้ามาได้

List ต่างจาก Array

1. ขนาดของ List ยืนหยุ่นได้ แต่ Array มีขนาดที่แน่นอน
2. ข้อมูลใน Array ต้องมีชนิดข้อมูลเหมือนกัน

การสร้าง List

List ชื่อตัวแปร = [‘ข้อมูล’];

List <ชนิดข้อมูล> ชื่อตัวแปร = [‘ข้อมูล’];

var ชื่อตัวแปร = [‘ข้อมูล’];

List Properties & Function

หาจำนวนสมาชิก

```
List color = [“แดง”, “น้ำเงิน”, “เหลือง”];
var x = color.length; //จำนวนสมาชิก
```

สมาชิกตัวแรกและตัวสุดท้าย

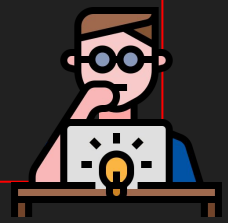
```
var first = color[0];
var last = color[color.length-1];
```

การเพิ่มสมาชิก

```
color.add(“สีเทา”);
```

การเปลี่ยนแปลงข้อมูลสมาชิก Array

```
var number = [10, 20, 30, 40];
number[1] = 100;
List <String>pets = [“แมว”,”กระต่าย”];
pets [1] = “เต่า”;
```



เข้าถึงสมาชิกใน List ด้วย For Loop

```
var color = [“แดง”, “น้ำเงิน”, “เหลือง”];
var count = color.length;

for (var i = 0; i < count ; i++) {
        print(color[i]);
}
```

เข้าถึงสมาชิกด้วย ForEach

```
var color = [“แดง”, “น้ำเงิน”, “เหลือง”];

 for (var item in color){
      print(item)
}
```

ฟังก์ชั่นที่ทำงานใน List

- add(value) // เพิ่มสมาชิกต่อท้าย
- addAll(list)
- insert(index,value)
- insertAll(index,list)

ฟังก์ชั่นที่ทำงานใน List

- remove(value) ลบสมาชิก
- removeRange(start,stop-1) ลบสมาชิกแบบกำหนดช่วง
- removeAt(index) // ลบสมาชิกในตำแหน่งที่ต้องการ
- removeWhere(condition) เช่น

  removeWhere((item) => item % 2 == 0);

Map คืออะไร

เป็นโครงสร้างข้อมูลที่เก็บข้อมูล Key กับ Value มีลักษณะคล้ายๆ กับ List แต่สามารถกำหนดชื่อของ Index ได้โดยมีโครงสร้างการนิยาม Map ดังนี้

Map <key,value> , Map<dynamic,dynamic>

การนิยาม

```
Map <String,String> color = {“Red”:”แดง”,....};
Map <String,int> money= {“balance”:5000,.....}
Map <String,String>word= {“th”:”ประเทศไทย”}
Map[ <int,String> code = {404:”Not Found”,200:”Ok”}
```

การเข้าถึงข้อมูล

```
fruit[‘Mango’]
color[‘Red’]
```

การเพิ่มสมาชิกใน Map

```
Map <String,String> color = {“Red”:”แดง”};
color[“Green”] = “เขียว”
```

การลบสมาชิกใน Map

```
color.remove(‘Green’)
```

การแปลง List เป็น Map

```
List<String> data = ['A', 'B', 'C'];
Map<int, String> item = data.asMap();
print(item)
```

การแปลง Map เป็น List

```
Map<String, int> data = {'X': 50, 'Y': 100, 'Z': 150};
data.keys      // [X, Y, Z]
data.values    // [50, 100, 150]
```

การเขียนโปรแกรมเชิงวัตถุ

แนวความคิดภาษาเชิงวัตถุ

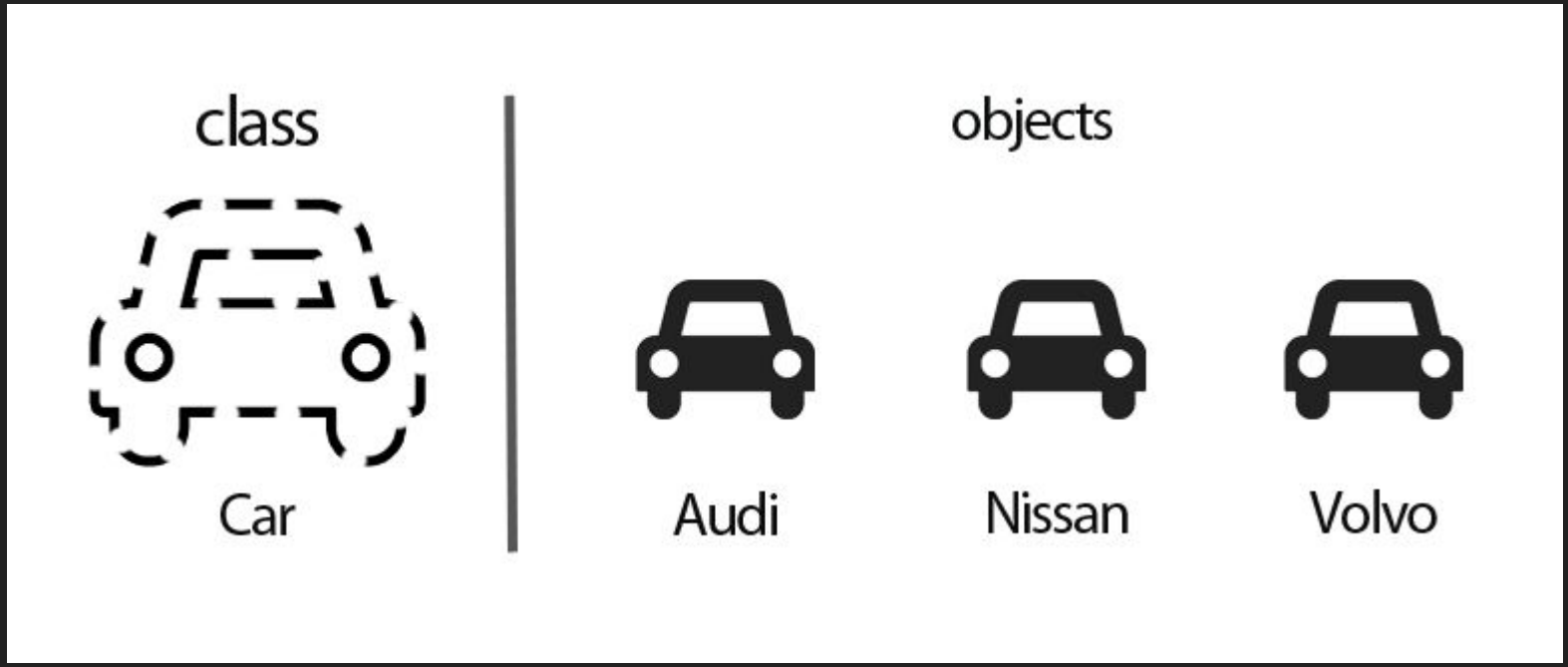
คลาส (class) คือต้นแบบของวัตถุ การจะสร้างวัตถุขึ้นมาอย่างหนึ่งจะต้องสร้างคลาสขึ้นมาเป็นโครงสร้างต้นแบบสำหรับวัตถุก่อนเสมอ

วัตถุหรือออบเจ็ค (object) คือสิ่งที่ประกอบไปด้วยคุณสมบัติ 2 ประการ คือ คุณลักษณะ และพฤติกรรม

คุณลักษณะ (attribute หรือ data member) คือ สิ่งที่บ่งบอกลักษณะทั่วไปของวัตถุ

พฤติกรรม (behavior หรือ method) คือ พฤติกรรมทั่วไปของวัตถุที่สามารถกระทำได้

แนวความคิดภาษาเชิงวัตถุ



Pokemon
Name: Pikachu
Type: Electric
Health: 70
attack()
dodge()
evolve()

Fields

Methods

https://www.youtube.com/channel/UCQ1r_4x-P-fETLIU4pqf98w https://www.facebook.com/KongRuksiamTutorial/

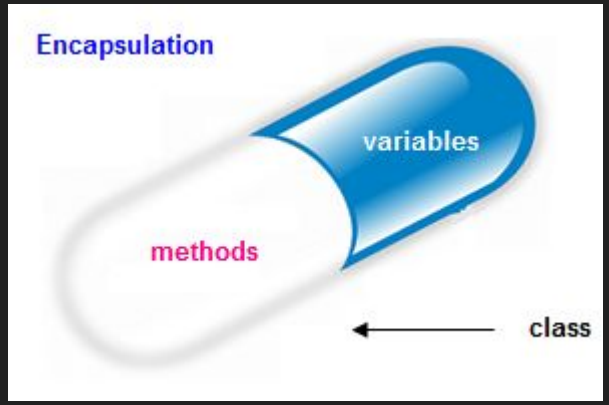


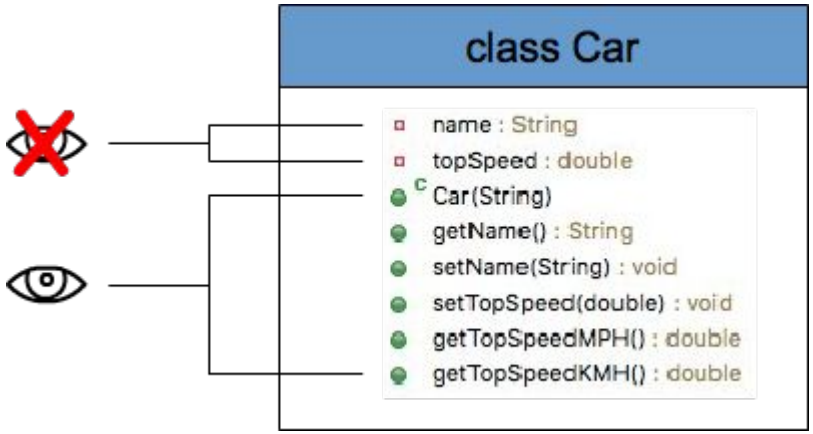



https://www.youtube.com/channel/UCQ1r_4x-P-fETLIU4pqf98w https://www.facebook.com/KongRuksiamTutorial/

การห่อหุ้ม (Encapsulation)

- เป็นกระบวนการซ่อนรายละเอียดการทำงานและข้อมูลไว้ภายในไม่ให้ภายนอกสามารถมองเห็นได้
- ทำให้ภายนอกไม่สามารถทำการเปลี่ยนแปลงแก้ไขข้อมูลภายในได้ ซึ่งเป็นผลทำให้เกิดความเสียหายแก่ข้อมูล
- ข้อดีของการห่อหุ้มคือสามารถสร้างความปลอดภัยให้แก่ข้อมูลได้ เนื่องจากข้อมูลจะถูกเข้าถึงจากผู้มีสิทธิเท่านั้น

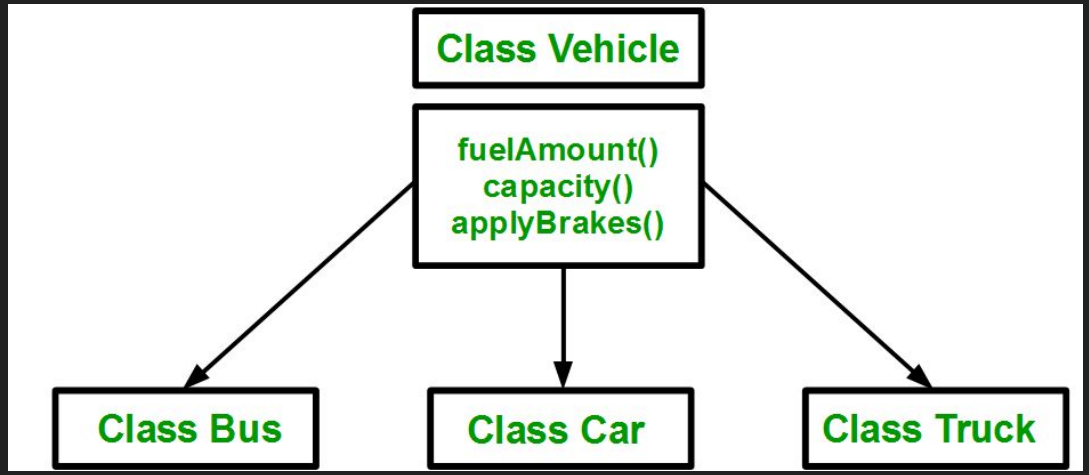




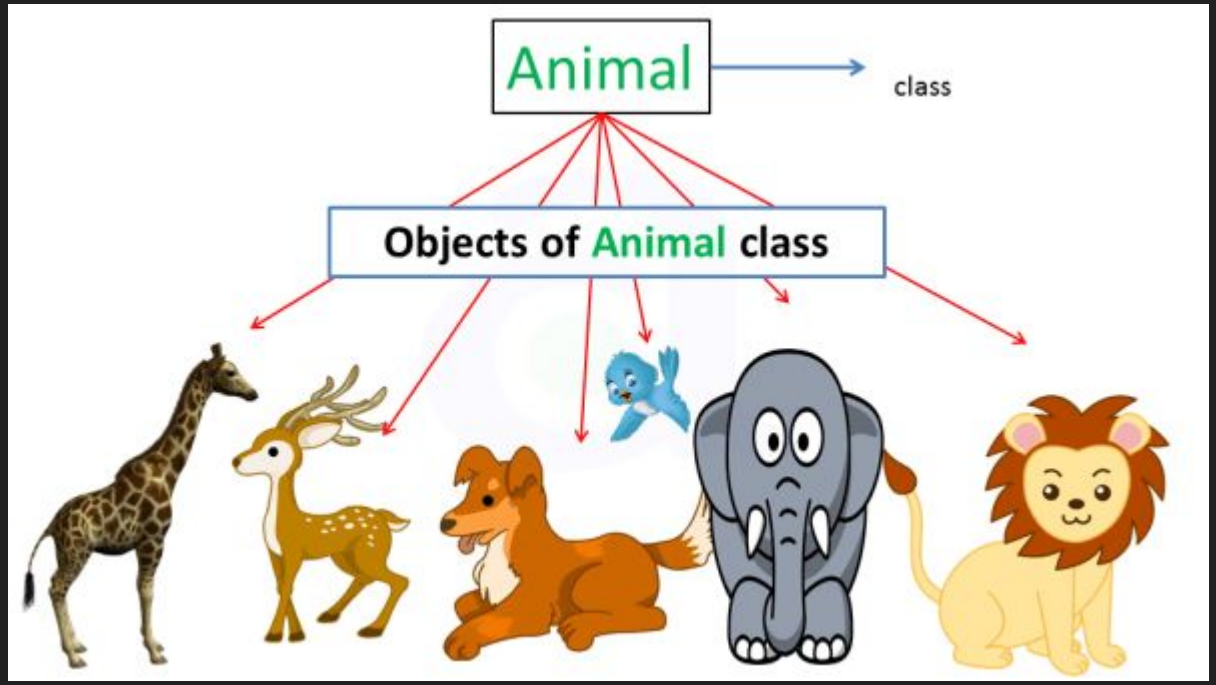
https://www.youtube.com/channel/UCQ1r_4x-P-fETLIU4pqf98w https://www.facebook.com/KongRuksiamTutorial/

การสืบทอดคุณสมบัติ (Inheritance)

หลักการของ inheritance คือ ทำการสร้างสิ่งใหม่ขึ้นด้วยการสืบทอด หรือรับเอา (inherit) คุณสมบัติบางอย่างมาจากสิ่งเดิมที่มีอยู่แล้ว โดยการสร้างเพิ่มเติมจากสิ่งที่มีอยู่แล้วได้เลย ข้อดีของการ inheritance คือ จากการที่สามารถนำสิ่งที่เคยสร้างขึ้นแล้วนำกลับมาใช้ใหม่ (re-use) ได้ ทำให้ช่วยประหยัดเวลาการทำงานลง เนื่องจากไม่ต้องเสียเวลาพัฒนาใหม่หมด

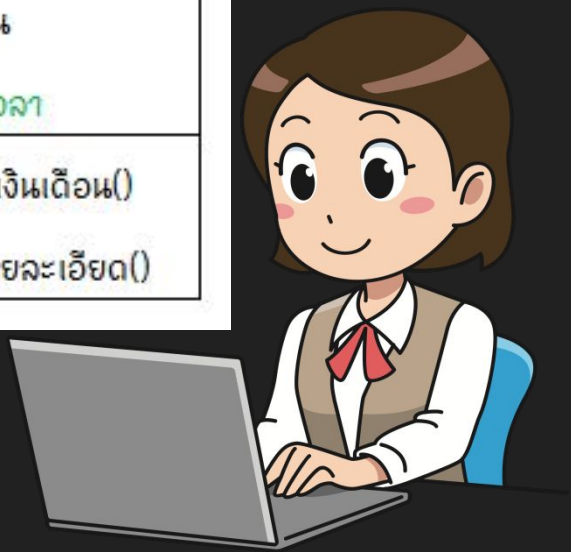


คลาสแม่ (Superclass) คลาสลูก (Subclass)



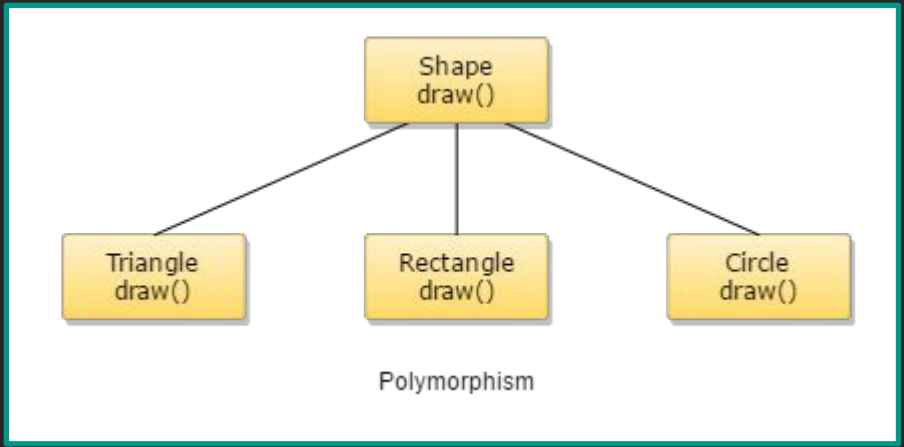
Employee

ผู้จัดการ	พนักงานขาย	พนักงานฝ่ายผลิต
- รหัสพนักงาน - ชื่อ - เงินเดือน - ที่จอดรถ	- รหัสพนักงาน - ชื่อ - เงินเดือน - ค่าคอมมิชั่น	- รหัสพนักงาน - ชื่อ - เงินเดือน - ค่าล่วงเวลา
+ คำนวณเงินเดือน() + แสดงรายละเอียด()	+ คำนวณเงินเดือน() + แสดงรายละเอียด()	+ คำนวณเงินเดือน() + แสดงรายละเอียด()



การพ้องรูป (Polymorphism)

Polymorphism เกิดจาก **poly** (หลากหลาย) + **morphology** (รูปแบบ)



ในทางโปรแกรมคือการที่เมธอดชื่อเดียวกัน สามารถรับอาร์กิวเมนต์ที่แตกต่างกันได้หลายรูปแบบ โดยเมธอดนี้จะถูกเรียกว่า overload method (เมธอดถูกโอเวอร์โหลด)

สรุปง่ายๆ

- **Class – ต้นแบบของวัตถุ**
- **Object – สิ่งที่ถูกสร้างขึ้นมาจาก Class ประกอบด้วย**
  - คุณสมบัติ (Attribute)
  - พฤติกรรม (Method)
- **คุมสมบัติของการเขียนโปรแกรมเชิงวัตถุ**
  - การห่อหุ้ม(Encapsulation)
  - การสืบทอด (Inheritance)
  - การพ้องรูป (Polymorphism)

การสร้าง Class และ Method

```
class Product{
    String name=”คีย์บอร์ด”;
    double price=1500.00;
    void show()=>print(“สินค้า”);// การสร้าง method
}
```

การสร้าง Object

```
Product product1=Product();
    product1.name;
    product1.show();
Product product2=Product();
    product2.show();
```

Access Modifier คือ ระดับในการเข้าถึง Class, Attribute, Method และอื่น ๆ ในภาษาเชิงวัตถุ โดยมีประโยชน์อย่างมากในเรื่องของการกำหนดระดับการเข้าถึง สิทธิในการเข้าใช้งาน การซ่อนข้อมูล และอื่น ๆ

Public เป็นการประกาศระดับการเข้าถึงที่เข้มงวดน้อยที่สุด หรือกล่าวได้ว่าใคร ๆ ก็สามารถเข้าถึงและเรียกใช้งานได้

Private เป็นการประกาศระดับการเข้าถึงที่เข้มงวดที่สุด กล่าวคือ จะมีแต่คลาสของตัวมันเองเท่านั้นที่มีสิทธิ์ใช้งานได้

Access Modifiers (Public / Private)

```
class Product {
    String name; // public
    String _name; // private
    void show()=>print("public method");
    void _show()=>print("private method");
}
```

คีย์เวิร์ด THIS & SUPER

- **this** คือ เมื่อต้องการเรียกใช้งาน
  คอนสตรัคเตอร์ หรือคุณลักษณะอื่นๆ ที่อยู่ภายในคลาสเดียวกัน

- **super** เมื่อต้องการเรียกคอนสตรัคเตอร์ของคลาสแม่ให้ทำงาน คีย์เวิร์ด **super** ในการเรียกใช้งานคอนสตรัคเตอร์ของคลาสแม่**จะต้องทำการเรียกที่บรรทัดแรกสุดของคอนสตรัคเตอร์นั้นๆ เท่านั้น**

Setter , Getter Method

Setter การกำหนดค่าให้ Object

```
class Product{
    String _name;
    void setName(String name){
        this._name=name;
    }
}
```

Gettter การดึงค่าจาก Object

```
class Product{
    String _name;
    String getName() => _name;
}
```

Constructor

การสร้าง Method ที่ชื่อเหมือนกับชื่อ Class ใช้สำหรับกำหนดค่าเริ่มต้นให้กับ Object

```
class Product{
        Product(){
            print(“default constructor”)
        }
}
```

Constructor

```
class Product{
        String _name; String _price;
        Product(String name,String price){
            this._name =name;
            this._price=price;
        }
    }
```

Constructor

รู้ลำดับการกำหนดค่าให้วัตถุสามารถลดรูปคำสั่งได้

```
class Product{
    String _name; String _price;
    Product(this._name,this._price);
 }
```

การสืบทอดคุณสมบัติ

```
class Product{
    ...
}
class Furniture extends Product{
    ...
}
```

การใช้งาน Super()

```
class Product{
    ...
}
class Furniture extends Product{
    super();
}
```

Overloading & Overriding Method

- **Overloading method** คือ **เมธอดที่มีชื่อเหมือนกัน** และ**อยู่ภายในคลาสเดียวกัน** สิ่งที่แยกความแตกต่างของเมธอดที่เป็น overload method คือ **พารามิเตอร์** (เป็นผลมาจากคุณสมบัติ OO คือ **polymorphism**)

  (ใช้รูปแบบ Optional Parameter แทน)

- **Overriding method** คือ **เมธอดของคลาสลูก (subclass) ที่มีชื่อเหมือนกับเมธอดของคลาสแม่ (superclass)** (เป็นผลมาจากคุณสมบัติ OO คือ **inheritance**)