



# เขียนโปรแกรมภาษา Kotlin

## สำหรับผู้เริ่มต้น

# รู้จักกับ Kotlin



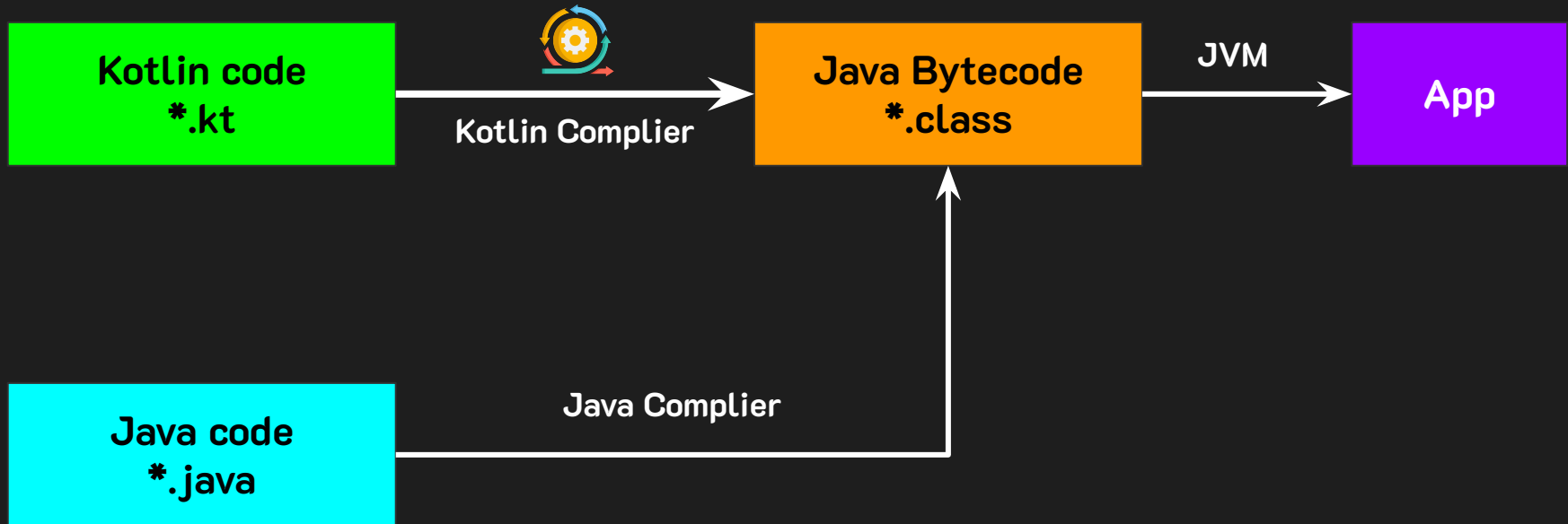
Kotlin คือ ภาษาคอมไพเตอร์ที่พัฒนาขึ้นมา เพื่อแก้ไขข้อบกพร่องบางส่วนที่มีอยู่ในภาษา Java โดยโค้ดที่เขียนด้วยภาษา Kotlin นั้น จะมีความสั้นและกระชับกว่าภาษา Java อีกทั้งยังสามารถนำภาษา Kotlin ไปพัฒนาแอปพลิเคชันบน Android ได้ด้วย

ภาษา Kotlin นั้นอาศัยตัวแปลภาษาของภาษา Java ดังนั้น การศึกษาภาษา Kotlin ควรจะเรียนรู้เกี่ยวกับพื้นฐานภาษา Java ด้วย

# ข้อดีของภาษา Kotlin

- โค้ดมีความสั้นกระชับกว่าภาษา Java
- เขียนได้ทั้งรูปแบบ Functional และ OOP
- แก้ไขข้อบกพร่องหลายอย่างในภาษา Java
- ทำงานบน JVM (Java Virtual Machine) จึงสามารถใช้งานข้าม Platform ได้
- ใช้แทนภาษา Java ในการพัฒนาแอปบน Android

# แผนภาพขั้นตอนการประมวลผลของ Kotlin



# เครื่องมือพื้นฐาน

- **JDK (Java Development Kit)** หรือชุดเครื่องมือสำหรับการเขียนโปรแกรมภาษา Java ประกอบด้วย Compiler และ Debugger
- **IntelliJ IDEA Community** คือชุดเครื่องมือสำหรับใช้ในการพัฒนาโปรแกรมที่ช่วยอำนวยความสะดวกในการเขียนโปรแกรมด้วยภาษา Kotlin

# โครงสร้างภาษา Kotlin

- ไฟล์ที่เก็บโค้ดภาษา Kotlin จะมีนามสกุลไฟล์ **.kt**
- ฟังก์ชัน **main()** คือ ฟังก์ชันพิเศษ กลุ่มคำสั่งที่อยู่ในฟังก์ชันนี้ จะทำงานโดยอัตโนมัติในตอนเริ่มต้นเสมอ

# โครงสร้างภาษา Kotlin

- **ขอบเขต (Block)** ใช้สัญลักษณ์ {} เพื่อบอกขอบเขตการทำงานของกรุปคำสั่งว่ามีจุดเริ่มต้นและสิ้นสุดที่ตำแหน่งใด
- **เครื่องหมายสิ้นสุดคำสั่ง** ใช้สัญลักษณ์ ; (ไม่ใช่ก็ได้)
- **หมายเหตุ (Comment)** ใช้สัญลักษณ์ // หรือ /\* \*/

# แสดงผลข้อมูล (Output)

คือคำสั่งสำหรับใช้แสดงผลข้อมูลออกจากจอภาพ  
ทั้งข้อมูลที่อยู่ในรูปแบบ ตัวเลขและตัวอักษรหรือ  
ผลลัพธ์จากการประมวลผล





# โครงสร้างคำสั่ง

คำสั่ง	คำอธิบาย
print(ข้อมูล)	แสดงข้อมูลออกทาง Console
println(ข้อมูล)	แสดงข้อมูลออกทาง Console และขึ้นบรรทัดใหม่

ถ้าข้อมูลอยู่ในรูปแบบข้อความ (สตริง) ให้เขียนไว้ในเครื่องหมาย “ ”

# หมายเหตุ (Comment)

## จุดประสงค์

- อธิบายหน้าที่หรือความหมายของโค้ดที่เขียน
- ยกเลิกโค้ดชั่วคราว ส่งผลให้ตัวแปลภาษาไม่สนใจโค้ดในบรรทัดที่ถูกทำหมายเหตุ

# หมายเหตุ (Comment)

วิธีที่ 1 โดยใช้เครื่องหมาย Slash ( // ) ใช้ในการอธิบายคำสั่งสั้นๆ  
ในรูปแบบบรรทัดเดียว

วิธีที่ 2 เขียนคำอธิบายไว้ในเครื่องหมาย /\* ... \*/ ใช้ในการอธิบาย  
คำสั่งยาวๆหรือแบบหลายบรรทัด

# ชนิดข้อมูลพื้นฐาน (Primitive Data Type)

ในภาษา Kotlin สามารถแบ่งประเภทของชนิดข้อมูลพื้นฐานได้ดังนี้

- ข้อมูลประเภทตัวเลขจำนวนเต็ม (Signed)
- ข้อมูลประเภทตัวเลขจำนวนเต็มบวก (Unsigned)
- ข้อมูลประเภทตัวเลขทศนิยม (Floating Point)
- ข้อมูลประเภทตรรกศาสตร์หรือค่าความจริง (Boolean)
- ข้อมูลประเภทอักขระ (Character & String)

# ข้อมูลประเภทตัวเลขจำนวนเต็ม

ชนิดข้อมูล	ช่วงข้อมูล	ขนาด (bit)	ข้อกำหนด
Byte	-128 ถึง 127	8	
Short	-32768 ถึง 32767	16	
Int	-2,147,483,648 ถึง 2,147,483,647	32	
Long	-9,223,372,036,854,775,808 ถึง 9,223,372,036,854,775,807	64	ใส่ L เป็น อักขระต่อท้าย

# ข้อมูลประเภทตัวเลขจำนวนเต็มบวก

ชนิดข้อมูล	ช่วงข้อมูล	ขนาด (bit)	ข้อกำหนด
UByte	0 ถึง 255	8	ใส่ u หรือ U เป็นอักขระต่อท้าย
UShort	0 ถึง 65535	16	ใส่ u หรือ U เป็นอักขระต่อท้าย
UInt	0 ถึง 4,294,967,295	32	ใส่ u หรือ U เป็นอักขระต่อท้าย
ULong	0 ถึง 18,446,744,073,709,551,615	64	ใส่ ul หรือ UL เป็นอักขระต่อท้าย

## ข้อมูลประเภทตัวเลขทศนิยม

ชนิดข้อมูล	ช่วงข้อมูล	ขนาด (bit)	ข้อกำหนด
Float	1.4E-45 ถึง 3.4028235E38	32	ใส่ f หรือ F เป็นอักขระต่อท้าย
Double	4.9E-324 ถึง 1.7976931348623157E308	64	

## ข้อมูลประเภทตรรกศาสตร์หรือค่าความจริง

ชนิดข้อมูล	ช่วงข้อมูล	ขนาด (bit)	ข้อกำหนด
Boolean	true หรือ false	1	

# ข้อมูลประเภทอักขระ

ชนิดข้อมูล	ช่วงข้อมูล	ขนาด (bit)	ข้อกำหนด
Char	เก็บข้อมูลอักขระเพียง 1 ตัว	16	
String	เก็บชุดข้อความหรืออักขระมากกว่า 1 ตัว		



# ตัวแปร (Variable)

คือ ชื่อที่ถูกระบุขึ้นมาเพื่อใช้เก็บค่าข้อมูลสำหรับนำไปใช้งานในโปรแกรม โดยข้อมูลประกอบด้วย ข้อความ ตัวเลข ตัวอักษร หรือผลลัพธ์จากการประมวลผล

\*ข้อมูลที่เก็บในตัวแปรสามารถเปลี่ยนแปลงค่าได้ (Mutable)

# โครงสร้างคำสั่ง

- **var** ชื่อตัวแปร : ชนิดข้อมูล;
- **var** ชื่อตัวแปร : ชนิดข้อมูล = ค่าเริ่มต้น;

ให้นำค่าทางขวามือของเครื่องหมาย = ไปเก็บไว้ในตัวแปรที่อยู่ด้านซ้ายมือ

# กฎการตั้งชื่อ

- ขึ้นต้นด้วยตัวอักษร A-Z หรือ a-z หรือ \_ เครื่องหมายขีดเส้นใต้เท่านั้น
- อักษรตัวแรกห้ามเป็นตัวเลข
- ตัวพิมพ์เล็ก-พิมพ์ใหญ่มีความหมายต่างกัน (Case Sensitive)
- ห้ามใช้อักขระพิเศษมาประกอบเป็นชื่อตัวแปร เช่น {}, %, ^ และช่องว่าง เป็นต้น
- ไม่สามารถประกาศชื่อเดียวกัน แต่มีชนิดข้อมูล 2 ชนิดได้
- ไม่ซ้ำกับคำสงวน (keyword) ในภาษา Kotlin

# ตัวอย่างคำสงวน (Keyword)

as	else	constructor	object	try
false	interface	return	break	for
is	super	var	val	continue
if	null	when	true	do
while	package	fun	class	throw

ข้อมูลเพิ่มเติม : <https://kotlinlang.org/docs/keyword-reference.html>

# ข้อมูลนักเรียน

ข้อมูล	รูปแบบ	ข้อกำหนด
ชื่อ	ข้อความ	
อายุ	ตัวเลขจำนวนเต็ม	
เกรด	ตัวเลขที่มีจุดทศนิยม	
สถานะ	ค่าทางตรรกศาสตร์	True = อยู่ระหว่างศึกษา False = จบการศึกษา
เพศ	ตัวอักษร	M = Male (ชาย) F = Female (หญิง)

# ค่าคงที่ (Constant)

มีลักษณะการใช้งานคล้ายกับตัวแปร แต่ค่าคงที่คือค่าที่ ไม่สามารถเปลี่ยนแปลงได้ (Immutable) ตอนประกาศใช้งาน ค่าคงที่จะต้องมีการประกาศค่าเริ่มต้นเสมอ

โครงสร้างคำสั่ง

`val` ชื่อค่าคงที่ : ชนิดข้อมูล = ค่าเริ่มต้น;

# Type Inference

คือการสร้างตัวแปรหรือค่าคงที่โดย**ไม่มีการระบุชนิดข้อมูลกำกับ** ซึ่งจะต้องกำหนดค่าเริ่มต้นในตอนทีประกาศ ตัวแปรหรือค่าคงที่นั้นจะมีชนิดข้อมูลอ้างอิงตามค่าที่กำหนดให้และจะไม่สามารถกำหนดข้อมูลชนิดอื่นได้ในภายหลัง

- `var grade = 3.50 // Double`
- `val status = true // Boolean`

# Type Casting

คือกระบวนการแปลงชนิดข้อมูลในภาษา Kotlin โดยใช้ฟังก์ชัน **toXXX()** สำหรับแปลงชนิดข้อมูลมาเขียนต่อท้ายชื่อตัวแปร เพื่อให้แปลงเป็นชนิดข้อมูลที่ต้องการ (โดย XXX หมายถึงชื่อชนิดข้อมูล)

toInt()	toLong()	toFloat()
toDouble()	toBoolean()	toString()



# ตัวดำเนินการ (Operators)

กลุ่มของเครื่องหมายหรือสัญลักษณ์ที่ใช้ในการเขียนโปรแกรม

$$A+B$$

- ตัวดำเนินการ (Operator)
- ตัวถูกดำเนินการ (Operand)

# ตัวดำเนินการ (Operators)

- ตัวดำเนินการทางคณิตศาสตร์
- ตัวดำเนินการเพิ่มค่าและลดค่า
- ตัวดำเนินการกำหนดค่า
- ตัวดำเนินการเปรียบเทียบ
- ตัวดำเนินการกำหนดช่วงข้อมูล
- ตัวดำเนินการทางตรรกศาสตร์

# ตัวดำเนินการทางคณิตศาสตร์

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

# ตัวดำเนินการเพิ่มค่าและลดค่า

เครื่องหมาย	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

# ตัวดำเนินการกำหนดค่า

เครื่องหมาย	รูปแบบการเขียน	ความหมาย
<code>+=</code>	<code>x+=y</code>	<code>x=x+y</code>
<code>-=</code>	<code>x-=y</code>	<code>x=x-y</code>
<code>*=</code>	<code>x*=y</code>	<code>x=x*y</code>
<code>/=</code>	<code>x/=y</code>	<code>x=x/y</code>
<code>%=</code>	<code>x%=y</code>	<code>x=x%y</code>

# ตัวดำเนินการเปรียบเทียบ

ผลลัพธ์จากการเปรียบเทียบจะได้ค่าทางตรรกศาสตร์ (True / False)

เครื่องหมาย	คำอธิบาย	ตัวอย่าง
==	เท่ากับ	$a==b$
!=	ไม่เท่ากับ	$a!=b$
>	มากกว่า	$a>b$
<	น้อยกว่า	$a<b$
>=	มากกว่าเท่ากับ	$a>=b$
<=	น้อยกว่าเท่ากับ	$a<=b$

# ตัวดำเนินการกำหนดช่วงข้อมูล

ใช้สำหรับกำหนดช่วงข้อมูลโดยระบุขอบเขตของข้อมูลให้อยู่  
ในลักษณะที่มีลำดับต่อเนื่องเช่น 1-10 หรือ a-z เป็นต้น

## รูปแบบการใช้งาน

- **.. (Range Operator)** กำหนดช่วงข้อมูลจากน้อยไปมาก โดยนับค่าสุดท้ายด้วย
- **until** ฟังก์ชันกำหนดช่วงข้อมูลจากน้อยไปมาก โดยไม่นับค่าสุดท้าย (เพิ่มค่า)
- **downTo** ฟังก์ชันกำหนดช่วงข้อมูลจากมากไปน้อย (ลดค่า)

กำหนดเพิ่มค่าหรือลดค่าผ่านคำสั่ง **step (default = 1)**

# ตรวจสอบค่าในช่วงข้อมูล

## in และ !in

- in ตรวจสอบว่าค่าที่ระบุนั้นอยู่ในช่วงข้อมูลหรือไม่
- !in ตรวจสอบว่าค่าที่ระบุนั้นไม่อยู่ในช่วงข้อมูลหรือไม่



# คำสั่งเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- When

# คำสั่งเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- When

# รูปแบบคำสั่งแบบเงื่อนไขเดียว

## If Statement

เป็นคำสั่งที่ใช้กำหนดเงื่อนไขในการตัดสินใจทำงานของโปรแกรม ถ้าเงื่อนไขเป็นจริงจะทำตามคำสั่งต่างๆที่กำหนดภายใต้เงื่อนไขนั้นๆ

# รูปแบบคำสั่งแบบเงื่อนไขเดียว

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



# โจทย์ปัญหา

คำนวณคะแนนสอบวิชาคอมพิวเตอร์ของนักเรียนในห้อง  
โดยมีคะแนนเต็ม 100 คะแนน ต้องการอยากรทราบว่านักเรียนคน  
ใดสอบผ่านบ้างใช้เกณฑ์วัดผลดังนี้

- คะแนนตั้งแต่ 50 คะแนนขึ้นไป => สอบผ่าน



# If...Else Statement

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

# โจทย์ปัญหา

คำนวณคะแนนสอบวิชาคอมพิวเตอร์ของนักเรียนในห้อง  
โดยมีคะแนนเต็ม 100 คะแนน ต้องการอยากรทราบว่านักเรียนคน  
ใดสอบผ่านบ้างใช้เกณฑ์วัดผลดังนี้

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



# รูปแบบคำสั่งแบบหลายเงื่อนไข

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



# โจทย์ปัญหา : ใช้บริการธนาคาร

ป้อนหมายเลขเพื่อใช้บริการ

- หมายเลข 1 : เปิดบัญชีใหม่ (create account)
- หมายเลข 2 : ถอนเงิน (withdraw)
- หมายเลข 3 : ฝากเงิน (deposit)
- หมายเลขอื่น : หมายเลขไม่ถูกต้อง (Invalid)

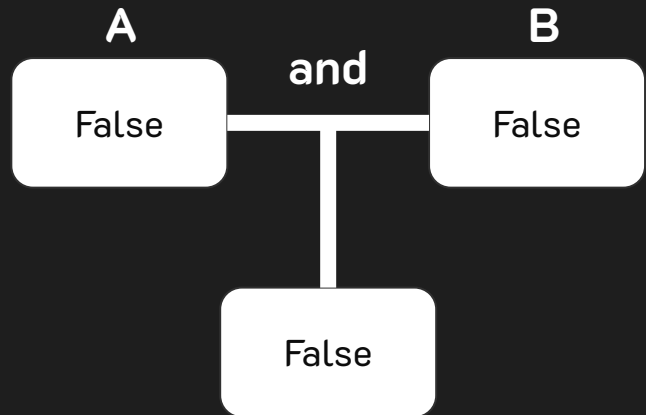
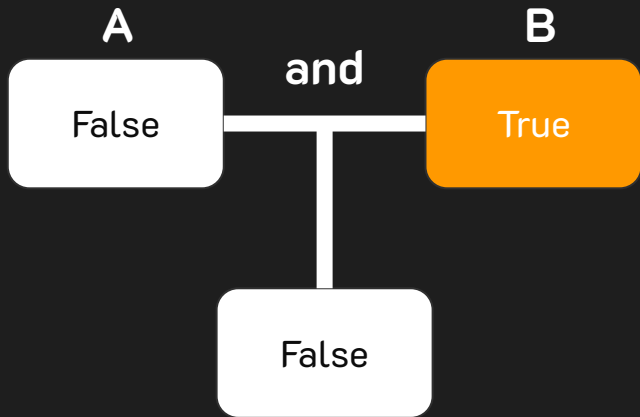
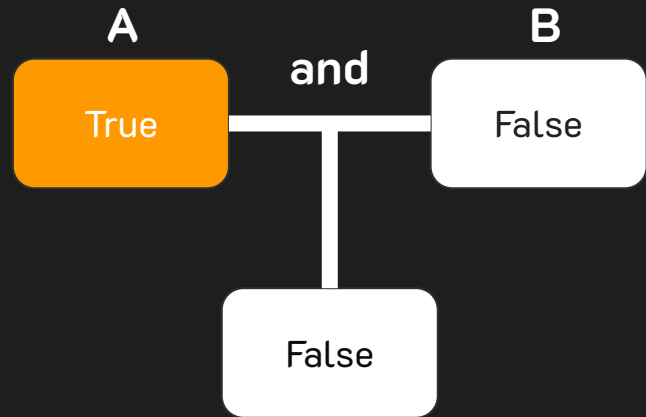
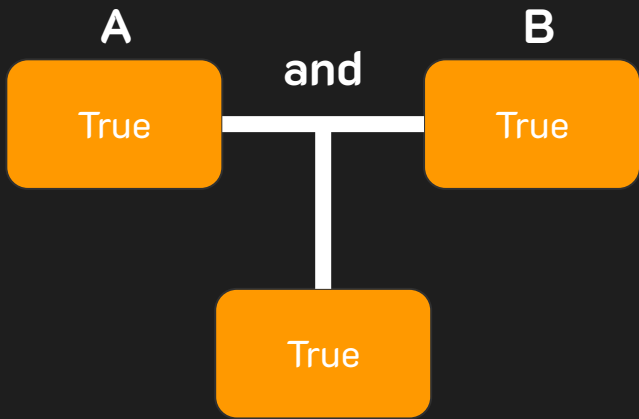
# ตัวดำเนินการทางตรรกศาสตร์

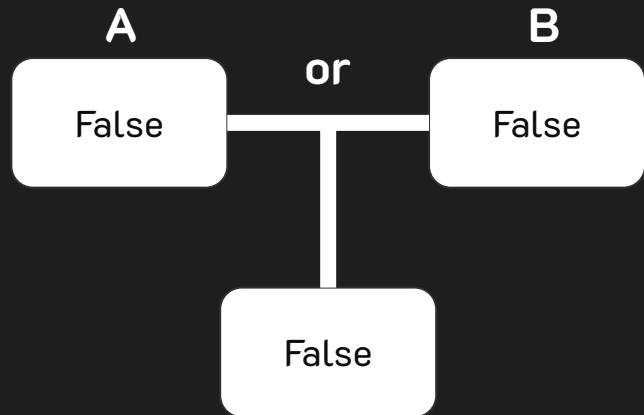
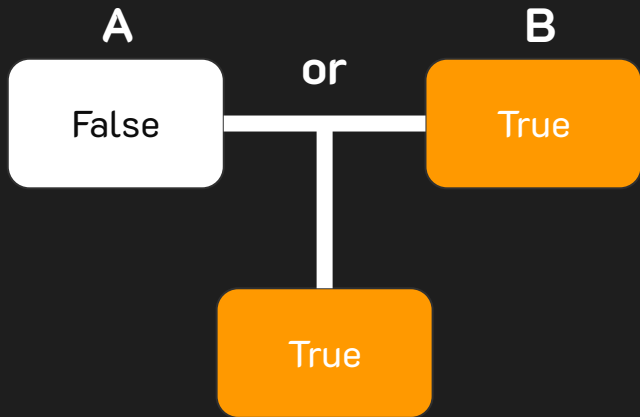
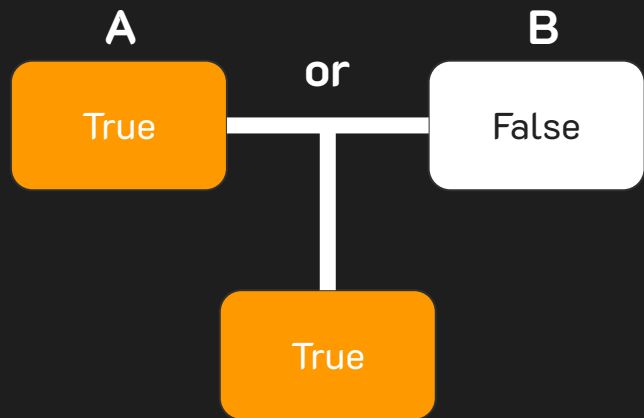
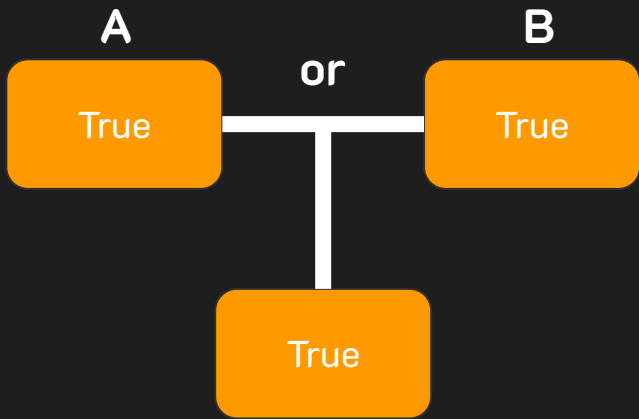
เครื่องหมาย	คำอธิบาย
&& (AND)	และ
(OR)	หรือ
! (NOT)	ไม่

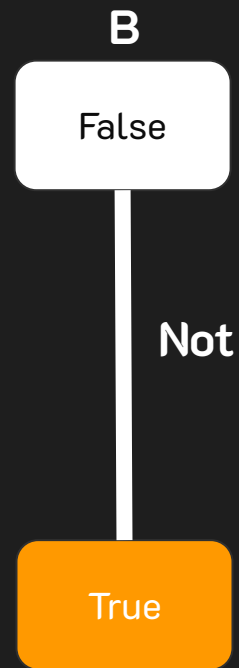
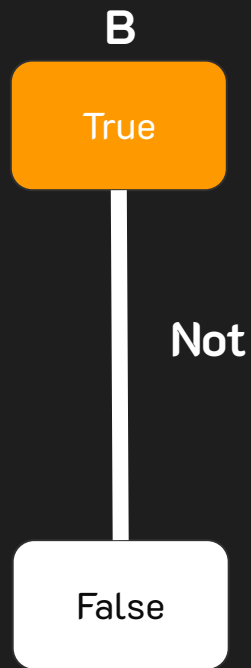
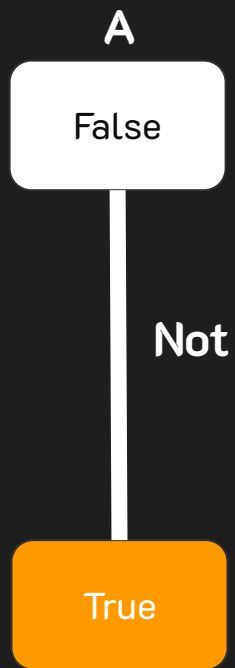
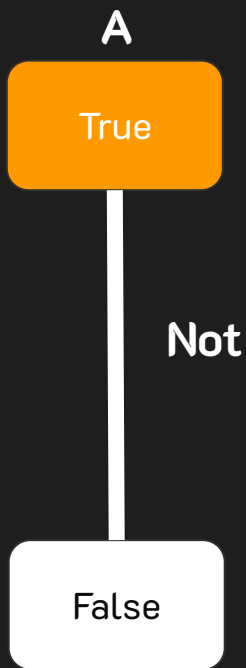
# ตัวดำเนินการทางตรรกศาสตร์

A	!A
true	false
false	true

A	B	A && B	A    B
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true







# ตัวดำเนินการทางตรรกศาสตร์

- ให้  $A$  = เงื่อนไขที่ 1 (ชื่อผู้ใช้)
- ให้  $B$  = เงื่อนไขที่ 2 (รหัสผ่าน)

# โจทย์ปัญหา : การล็อกอินเข้าสู่ระบบ

## AND (และ) , &&

ชื่อผู้ใช้เป็น admin และ รหัสผ่านเท่ากับ 1234 (เข้าสู่ระบบได้)

## OR (หรือ) , ||

ชื่อผู้ใช้เป็นค่าว่าง หรือ รหัสผ่านเป็นค่าว่าง (ข้อมูลไม่ถูกต้อง)

## NOT (ไม่) , !

ชื่อผู้ใช้ไม่เท่ากับ admin



# คำสั่งเงื่อนไข (Condition)

กลุ่มคำสั่งที่ใช้ตัดสินใจในการเลือกทำงานตามเงื่อนไขต่างๆ ภายในโปรแกรม

- if
- When

# When

เป็นคำสั้งที่ใช้กำหนดเงื่อนไขคล้ายๆกับ if แต่สามารถกำหนดเงื่อนไขหรือทางเลือกในการทำงานได้หลากหลาย

การเขียนโค้ดมีความกระชับกว่าการใช้งาน if โดยนำค่าในตัวแปรมากำหนดเป็นทางเลือกในการทำงาน

# รูปแบบคำสั่ง

**when** (ตัวแปร) {

เงื่อนไขที่ 1 -> คำสั่งเมื่อเงื่อนไขที่ 1 เป็นจริง

เงื่อนไขที่ N -> คำสั่งเมื่อเงื่อนไขที่ N เป็นจริง

else -> คำสั่งเมื่อทุกเงื่อนไขเป็นเท็จ

}

# โจทย์ปัญหา : ใช้บริการธนาคาร

ป้อนหมายเลขเพื่อใช้บริการ

- หมายเลข 1 : เปิดบัญชีใหม่ (create account)
- หมายเลข 2 : ถอนเงิน (withdraw)
- หมายเลข 3 : ฝากเงิน (deposit)
- หมายเลขอื่น : หมายเลขไม่ถูกต้อง (Invalid)

# โจทย์ปัญหา : โปรแกรมตัดเกรดอย่างง่าย

คำนวณเกรดจากคะแนนสอบของนักเรียน คะแนนเต็ม 100 คะแนน โดยมีเกณฑ์วัดผลดังนี้ คือ

- คะแนน 80 - 100 ได้เกรด A
- คะแนน 60 - 79 ได้เกรด B
- คะแนน 0 - 59 คะแนน ได้เกรด F
- ป้อนค่าอื่น ให้แจ้งข้อผิดพลาด (Invalid)



# คำสั่งทำซ้ำ (Loop)

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

# คำสั่งทำซ้ำ (Loop)

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

# While Loop

จะทำงานตามคำสั่งภายใน while ไปเรื่อยๆเมื่อเงื่อนไขที่กำหนดเป็นจริง

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



```
while(condition){  
    //statement  
}
```

## Output

แสดงข้อความ “Hello Kotlin”  
จำนวน 3 ครั้ง

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

Output

```
var count = 1;
```

```
while(count<=3){
```

```
    println("Hello Kotlin");
```

```
    count++;
```

```
}
```

Output

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

Output

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

Output

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2



```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , **count=2**

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , **count=2**

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2

```
var count = 1;
while(count<=3){
    println("Hello Kotlin");
    count++;
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , **count=3**

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3



```
var count = 1;
while(count<=3){
    println("Hello Kotlin");
    count++;
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3

```
var count = 1;
while(count<=3){
    println("Hello Kotlin");
    count++;
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin

```
var count = 1;
while(count<=3){
    println("Hello Kotlin");
    count++;
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin , count=4

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin , **count=4**

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin , count=4

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin , count=4

```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}  
//จบโปรแกรม
```

## Output

- Hello Kotlin , count=2
- Hello Kotlin , count=3
- Hello Kotlin , count=4



```
var count = 1;  
while(count<=3){  
    println("Hello Kotlin");  
    count++;  
}
```

//จบโปรแกรม

## Output

- Hello Kotlin
- Hello Kotlin
- Hello Kotlin

# คำสั่งทำซ้ำ (Loop)

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

# For-Loop

เป็นรูปแบบการซ้ำให้ครบตามจำนวนครั้งที่กำหนดโดยมีการกำหนดการทำงานโดยใช้คำสั่ง **for-in** ซึ่งจะใช้งานร่วมกับ

- Range Operator (..)
- ฟังก์ชัน Until
- ฟังก์ชัน DownTo

# For-Loop

```
for(ตัวแปร in ค่าแรก..ค่าสุดท้าย) { //Range Operator  
    คำสั่งที่ต้องการให้ทำซ้ำ  
}
```

# For-Loop

```
for(ตัวแปร in ค่าแรก until ค่าสุดท้าย) { // น้อยไปมาก  
    คำสั่งที่ต้องการให้ทำซ้ำ  
}
```

```
for(ตัวแปร in ค่าแรก downTo ค่าสุดท้าย) { // มากไปน้อย  
    คำสั่งที่ต้องการให้ทำซ้ำ  
}
```

# Repeat

เป็นรูปแบบการซ้ำให้ครบตามจำนวนครั้งที่กำหนดโดยไม่ได้  
นำค่าตัวแปรของ Loop มาใช้งานสามารถใช้ฟังก์ชัน repeat แทน  
for-loop ได้

```
repeat(จำนวนครั้ง){  
    คำสั่งที่ต้องการให้ทำซ้ำ  
}
```

# คำสั่งทำซ้ำ (Loop)

กลุ่มคำสั่งที่ใช้ในการวนรอบ (Loop) โปรแกรมจะทำงานไปเรื่อยๆจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ จึงจะหยุดทำงาน

- While
- For
- Do..While

# Do...While Loop

โปรแกรมจะทำงานตามคำสั่งอย่างน้อย 1 รอบ เมื่อทำงานเสร็จจะ  
มาตรวจสอบเงื่อนไขที่คำสั่ง while ถ้าเงื่อนไขเป็นจริงจะวนกลับขึ้นไปทำ  
งานที่คำสั่งใหม่อีกรอบ แต่ถ้าเป็นเท็จจะหลุดออกจากลูป

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



# คำสั่งที่เกี่ยวข้องกับ Loop

- **break** ถ้าโปรแกรมพบคำสั่งนี้จะหลุดจากการทำงานในลูปทันที เพื่อไปทำคำสั่งอื่นที่อยู่นอกลูป
- **continue** คำสั่งนี้จะทำให้หยุดการทำงานแล้วย้อนกลับไปเริ่มต้นการทำงานที่ต้นลูปใหม่

# Pair

เป็นการเก็บชุดข้อมูลที่ประกอบด้วยสมาชิก 2 ตัวหรือเก็บข้อมูลแบบคู่

โครงสร้างคำสั่ง

Pair(สมาชิกตัวที่ 1 , สมาชิกตัวที่ 2)

การเข้าถึงข้อมูล

first = สมาชิกตัวที่ 1 , second = สมาชิกตัวที่ 2

# Triple

เป็นการเก็บชุดข้อมูลที่ประกอบด้วยสมาชิก 3 ตัว

โครงสร้างคำสั่ง

Triple(สมาชิกตัวที่ 1 , สมาชิกตัวที่ 2 , สมาชิกตัวที่ 3)

การเข้าถึงข้อมูล

first = สมาชิกตัวที่ 1 , second = สมาชิกตัวที่ 2 ,

third = สมาชิกตัวที่ 3

# เจาะลึกการใช้งาน String

- การเข้าถึงตัวอักษรใน String
- การเชื่อมต่อ String (Concatenation)
- การกำหนดค่าใน String แบบหลายบรรทัด
- การแทรกข้อมูลใน String  
(String Templates / Interpolation)

# ฟังก์ชันเกี่ยวกับ String

ชื่อฟังก์ชัน	คำอธิบาย
length หรือ count()	ความยาวหรือจำนวนตัวอักษรใน String
equals()	เปรียบเทียบ String (true/false)

# ฟังก์ชันเกี่ยวกับ String

ชื่อฟังก์ชัน	คำอธิบาย
<code>indexOf(string)</code>	ลำดับตัวอักษรหรือ String ที่เจอครั้งแรก
<code>lastIndexOf(string)</code>	ลำดับตัวอักษรหรือ String ที่เจอครั้งสุดท้าย
<code>startsWith(string)</code>	เริ่มต้นด้วย String ที่ระบุหรือไม่
<code>endsWith(string)</code>	ลงท้ายด้วย String ที่ระบุหรือไม่

# ฟังก์ชันเกี่ยวกับ String

ชื่อฟังก์ชัน	คำอธิบาย
<code>replaceFirst(old , string)</code>	แทนที่ String ที่พบครั้งแรกด้วย String ใหม่
<code>replace(old , string)</code>	แทนที่ String ที่พบทั้งหมดด้วย String ใหม่

# ฟังก์ชันเกี่ยวกับ String

ชื่อฟังก์ชัน	คำอธิบาย
take(จำนวน)	ตัด String ตามจำนวนที่ระบุ โดยเริ่มจากตัวแรก
trim()	ตัดช่องว่างซ้ายขวาออกจาก String



# ฟังก์ชันเกี่ยวกับ String

ชื่อฟังก์ชัน	คำอธิบาย
subString(ตำแหน่งเริ่มต้น)	หา String ย่อยจากตำแหน่งเริ่มต้น จนถึงสิ้นสุด String
subString(ตำแหน่งเริ่มต้น , ตำแหน่งสุดท้าย)	หา String ย่อยจากตำแหน่งเริ่มต้น จนถึง ก่อน ตำแหน่งสุดท้ายString
subString(ตำแหน่งเริ่มต้น..ตำแหน่งสุดท้าย)	หา String ย่อยจากตำแหน่งที่ระบุ โดยกำหนดช่วงผ่าน range