พื้นฐานการเขียนโปรแกรมภาษาไพธอน

บทที่ 6 การทำซ้ำ

วัตถุประสงค์ของบทเรียน

เมื่อสิ้นสุดบทเรียน ผู้เรียนจะสามารถ

- อธิบายถึงหลักการการเขียนโปรแกรมแบบวนซ้ำได้
- ใช้โครงสร้างการทำซ้ำในการออกแบบโปรแกรมได้
- พัฒนาโปรแกรมที่สามารถทำซ้ำได้

ทำไมต้องใช้โครงสร้างการทำซ้ำ

- หลักของการทำงานเชิงเส้น (Sequential Structure) จะบังคับให้โปรแกรมทำงานเป็นเส้นตรง ทุกรอบของการประมวลผล
  - มีจำนวนขั้นตอนการทำงานเหมือนกันทุกครั้ง

- หลักของการทำงานเชิงเส้น (Selection Structure) จะบังคับให้โปรแกรมเลือกระหว่างทางเลือก โดยใช้เงื่อนไขในการเลือก

- ดังนั้น โปรแกรมสามารถผสมผสานโครงสร้างทั้ง 2 แบบเพื่อออกแบบการแก้ไขปัญหาได้

กรณีศึกษา #1

จงเขียนโปรแกรมที่แสดงข้อความคำว่า "Hello World" เท่ากับตัวเลขที่ผู้ใช้กรอก

กรณีศึกษา #1

- หากผู้ใช้ กรอกเลข 3 โปรแกรมจะทำการแสดง

Hello World
Hello World
Hello World

กรณีศึกษา #1

- หากผู้ใช้ กรอกเลข 5 โปรแกรมจะทำการแสดง

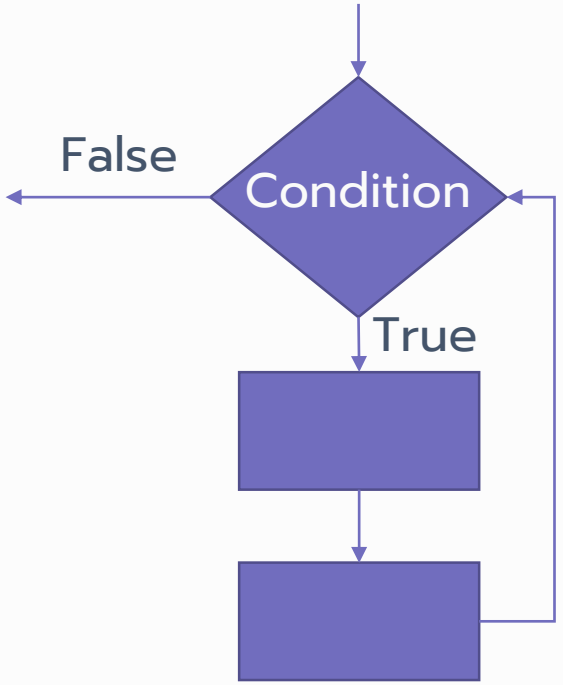
Hello World
Hello World
Hello World
Hello World
Hello World

เราจะเขียนโปรแกรมนี้อย่างไร โดยใช้ โครงสร้างเชิงเส้น และ โครงสร้างทำซ้ำ

โครงสร้างการทำซ้ำ

- โครงสร้างการทำซ้ำ คือ รูปแบบการออกแบบโปรแกรมให้สามารถที่จะทำกิจกรรม หรือ ชุดกิจกรรมหนึ่งๆ ซ้ำตามเงื่อนไขที่ถูกออกแบบไว้

- บางครั้งโครงสร้างการทำซ้ำจะถูกเรียกว่า Loop



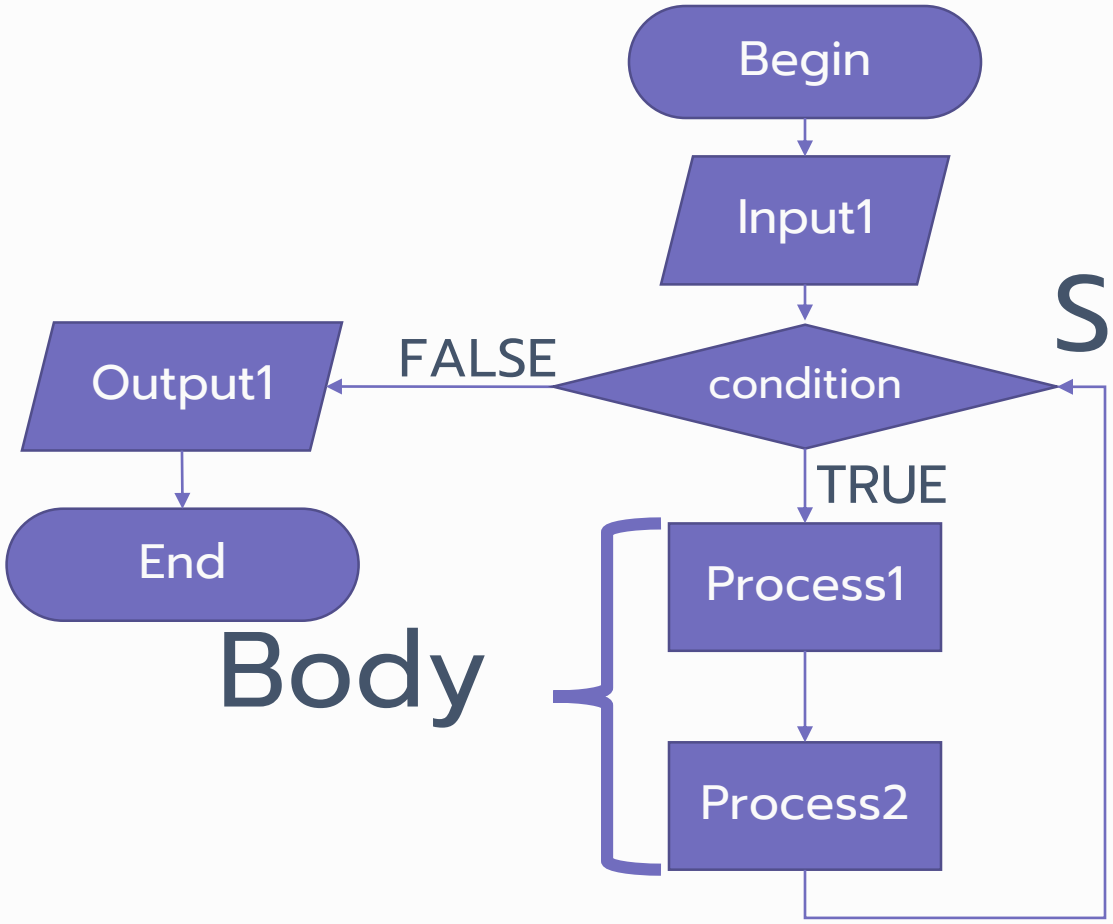
โครงสร้างการทำซ้ำ

- โครงสร้างการทำซ้ำ มี องค์ประกอบ 2 ส่วน

1) เงื่อนไขหยุด (Stop conditions)
   - คือ เงื่อนไขที่จะทำให้โครงสร้างการทำซ้ำหยุดการทำงาน
   - เงื่อนไขจะเป็นนิพจน์ตรรกะ (Boolean expression)

2) เนื้อหาการทำงาน (Statements หรือ Loop bodies)
   - คือ ขั้นตอนการทำงานที่จะทำซ้ำ เมื่อเงื่อนไขการหยุดไม่หยุดการทำงานของโครงสร้างการทำซ้ำ

Flowchart: Example



ประเภทของโครงสร้างการทำซ้ำ

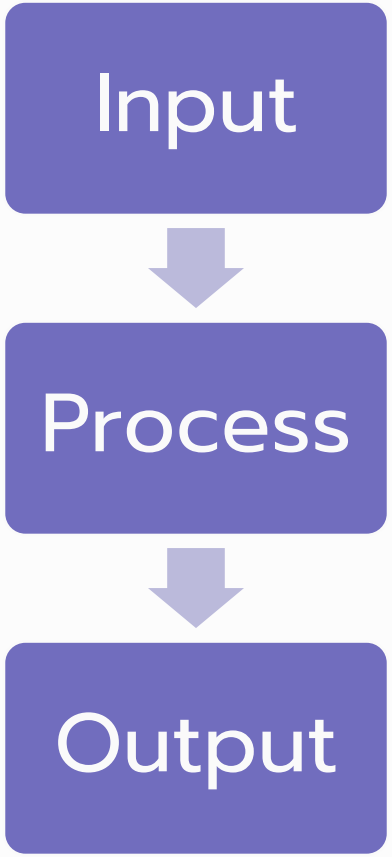
- โครงสร้างการทำซ้ำสามารถแบ่งออกได้เป็น 2 ประเภท คือ
- โครงสร้างการทำซ้ำแบบ ตรวจสอบเงื่อนไขก่อนถึงทำงาน (Pre-test loop)
- โครงสร้างการทำซ้ำแบบ ทำงานก่อนถึงตรวจสอบเงื่อนไข (Post-test loop)

กรณีศึกษา #2

จงเขียนโปรแกรมที่รับจำนวนเต็มบวกจากผู้ใช้ คำนวณค่า แฟคทอเรียล (Factorial) และ แสดงผลการคำนวณบนหน้าจอ

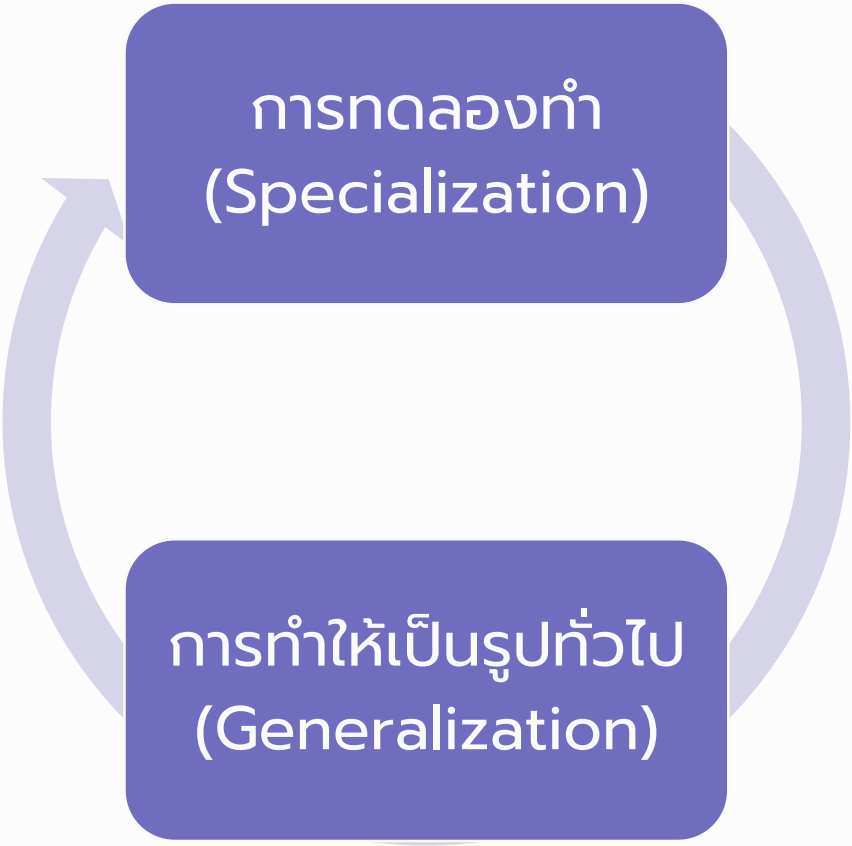
กรณีศึกษา #2 - วิเคราะห์ปัญหา

- อะไรคือข้อมูลนำเข้า? (Input)
  - จำนวนเต็มบวก 1 ค่า
- อะไรคือผลลัพธ์? (Output)
  - ค่าแฟคทอเรียลของข้อมูลนำเข้า
- อะไรคือกระบวนการ? (Process)
  - ??????????



เทคนิคในการออกแบบการแก้ไขปัญหา

- การแก้ไขปัญหาสามารถทำได้หลายแนวทาง
- การทดลองทำ (Specialization) คือ การศึกษาจากตัวอย่างของโจทย์ปัญหา เพื่อทำความเข้าใจปัญหา และทดลองหาวิธีแก้ปัญหา
- การทำให้เป็นรูปทั่วไป (Generalization) คือ การสร้างรูปแบบทั่วไปของการได้มาซึ่งคำตอบ



กรณีศึกษา #2 -การทดลองทำ

- อะไรคือการคำนวณแฟคทอเรียล ?????

$$\mathrm{n}! = n * (n-1) * \cdots * 1$$

เมื่อผู้ใช้กรอก 3 $3 * 2 * 1 = 6$

เมื่อผู้ใช้กรอก 5 $5 * 4 * 3 * 2 * 1 = 120$

ในแต่ละข้อมูล การคำนวณแฟคทอเรียล**แตกต่าง และ มี**
กิจกรรมบางอย่างเกิดซ้ำกันตามข้อมูลนำเข้าจากผู้ใช้

กรณีศึกษา #2 -การทดลองทำ

$$7! = 7 * 6 * 5 * 4 * 3 * 2 * 1$$

$$10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$$

การออกแบบโครงสร้างการทำงานซ้ำ

- ขั้นตอนการออกแบบโครงสร้างการทำงานซ้ำ สามารถแบ่งได้ 2 ขั้นตอน คือ

- การระบุเงื่อนไขการหยุด
  - เป็นการออกแบบวิธีการหยุดการทำงานของโครงสร้างการทำซ้ำ
  - โดยทั่วไป มี 2 แนวทาง คือ 1) หยุดตามจำนวนรอบ และ 2) หยุดตามค่า

- การระบุเนื้อหาการทำงาน
  - เป็นการออกแบบการทำงานของโครงสร้างการทำงานซ้ำ
  - โดยทั่วไป เนื้อหาการทำงานคือ ส่วนที่เกิดซ้ำ

กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

- ตัวอย่างการคำนวณแฟคทอเรียล

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

ข้อมูลนำเข้า = 5 ; ทำงานซ้ำ 4 ครั้ง

- การระบุเงื่อนไขการหยุด
  - ศึกษาจำนวนรอบของการทำงาน
    - นับที่**กิจกรรม**ที่เกิดซ้ำ
    - สัมพันธ์กับขนาดของข้อมูลนำเข้า => หยุดตามจำนวนรอบ
    - สัมพันธ์กับบางตัวแปร => หยุดตามค่า

กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

ข้อมูลนำเข้า = 7

$$7! = 7 * 6 * 5 * 4 * 3 * 2 * 1$$

ทำงานซ้ำ 6 ครั้ง; เริ่มต้นที่ 1 ; สิ้นสุดที่ 7

ข้อมูลนำเข้า = 10

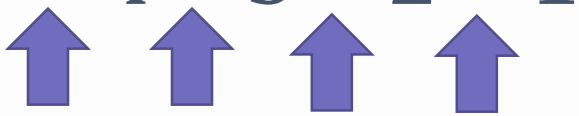
$$10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$$

ทำงานซ้ำ 9 ครั้ง ; เริ่มต้นที่ 1 ; สิ้นสุดที่ 10

กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

- ตัวอย่างการคำนวณแฟคทอเรียล

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$



ทำงานซ้ำคือการคูณ (*)

- การระบุเนื้อหาการทำงาน
  - ศึกษาสิ่งที่เกิดซ้ำ
    - กิจกรรม หรือ กลุ่มของกิจกรรม ที่เกิดขึ้น

กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

ข้อมูลนำเข้า = 7

$$7! = 7 * 6 * 5 * 4 * 3 * 2 * 1$$

ทำการคูณ (*) ซ้ำ

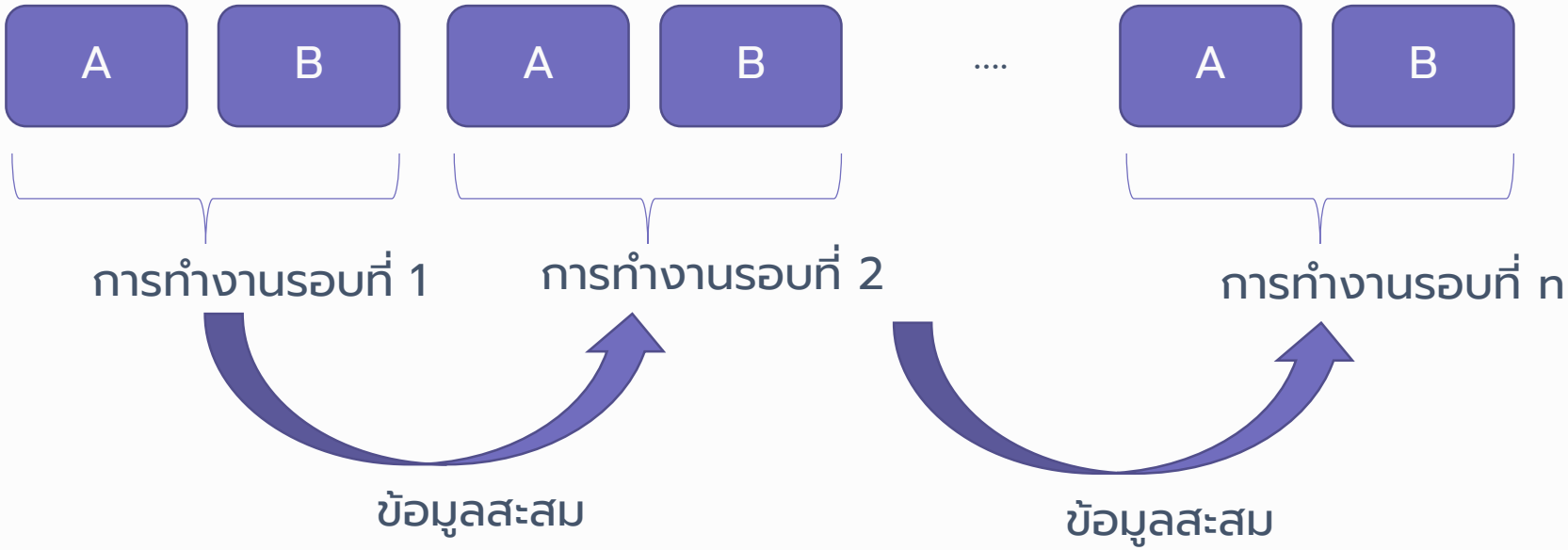
ข้อมูลนำเข้า = 10

$$10! = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$$

ทำการคูณ (*) ซ้ำ

กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

- การเชื่อมโยงระหว่างแต่ละรอบของการทำงาน
  - ต้องมีการส่งต่อข้อมูลจากรอบของการทำงานรอบก่อนหน้า ไปยังรอบถัดไป
  - การทำงานแต่ละรอบก็ถูกเชื่อมโดยการดำเนินงาน

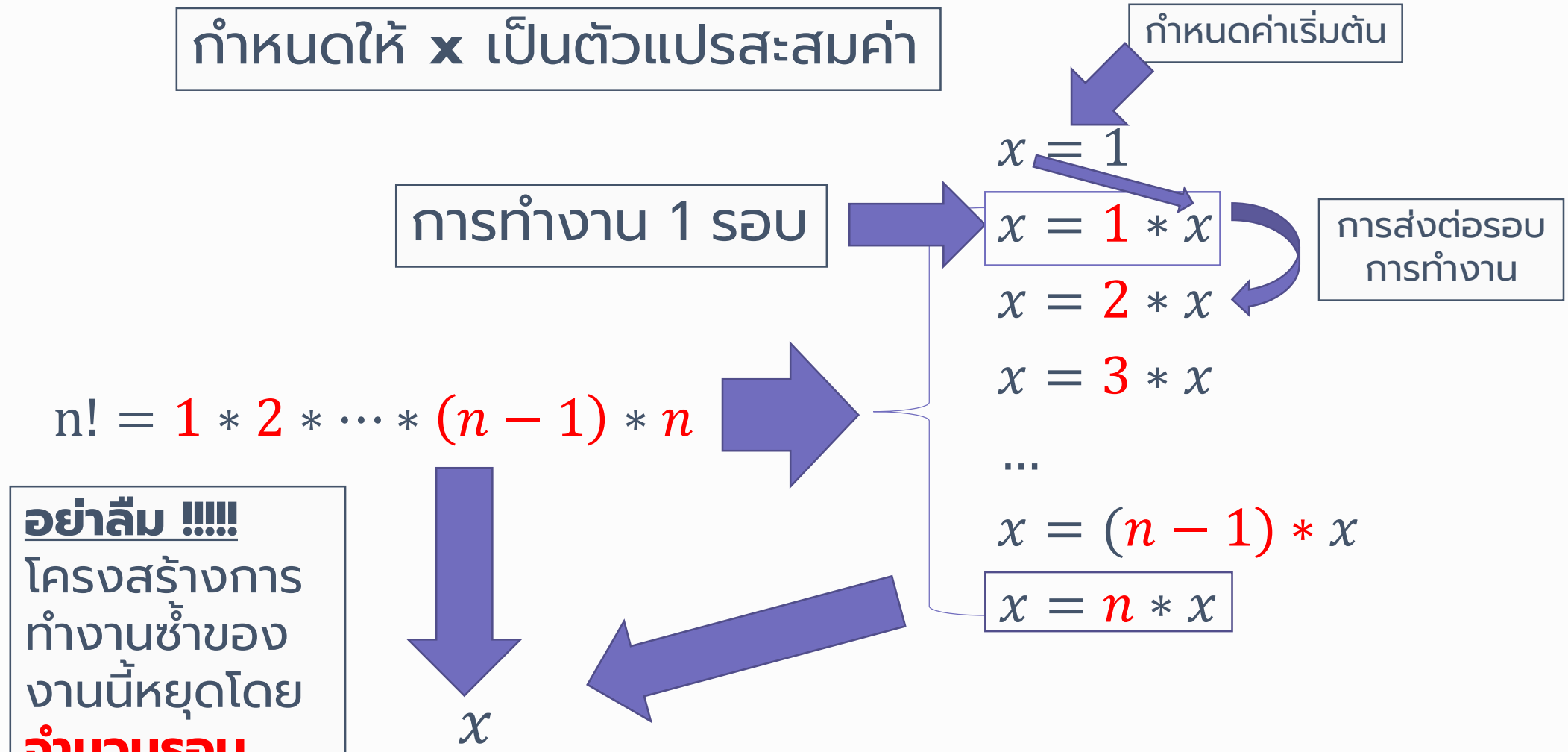


กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

- การเชื่อมโยงระหว่างแต่ละรอบของการทำงาน
  - วิธีการหนึ่ง คือ การใช้ตัวแปรเพื่อทำการสะสมค่า หรือ ตัวแปรสะสม (Cumulative variable)
  - ต้องมีการกำหนดค่าเริ่มต้น (Initial value)ที่สอดคล้องกับการดำเนินงาน
    - ถ้า การดำเนินการเป็นการคูณ * => กำหนดค่าเป็น 1
    - ถ้า การดำเนินการเป็นการบวก + => กำหนดค่าเป็น 0

กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

กำหนดให้ **x** เป็นตัวแปรสะสมค่า



<u>อย่าลืม !!!!!</u>
โครงสร้างการทำงานซ้ำของงานนี้หยุดโดย **<u>จำนวนรอบ</u>**

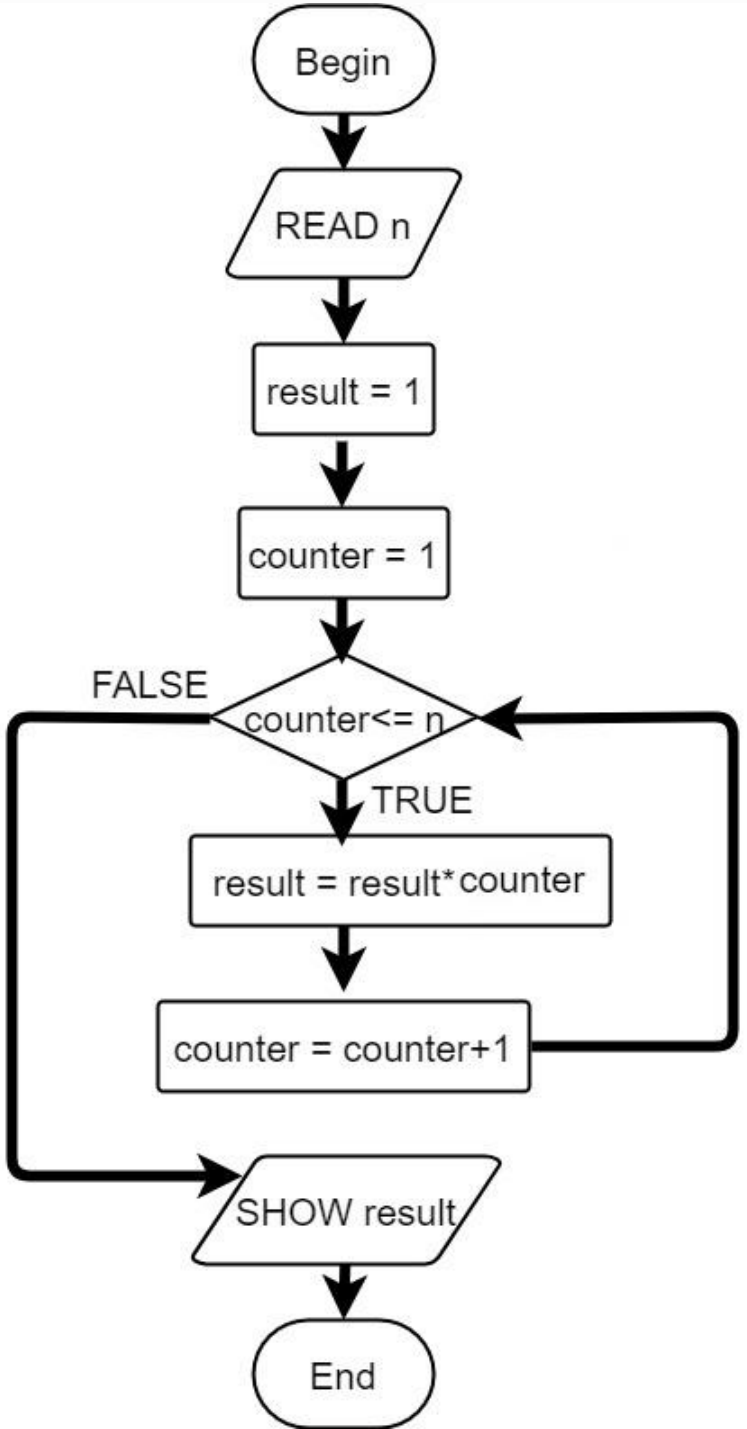
กรณีศึกษา #2 - การทำให้เป็นรูปทั่วไป

- วิธีการหนึ่ง คือ การใช้ตัวแปรเพื่อทำการสะสมค่า หรือ ตัวแปรสะสม (Cumulative variable)
- Perform the operation and store the result in the cumulative variable

รอบ	การดำเนินการ	ตัวแปรสะสม (x)
0		1
1	1*x	1
2	2*x	2
3	3*x	6
...		
n	n*x	n!

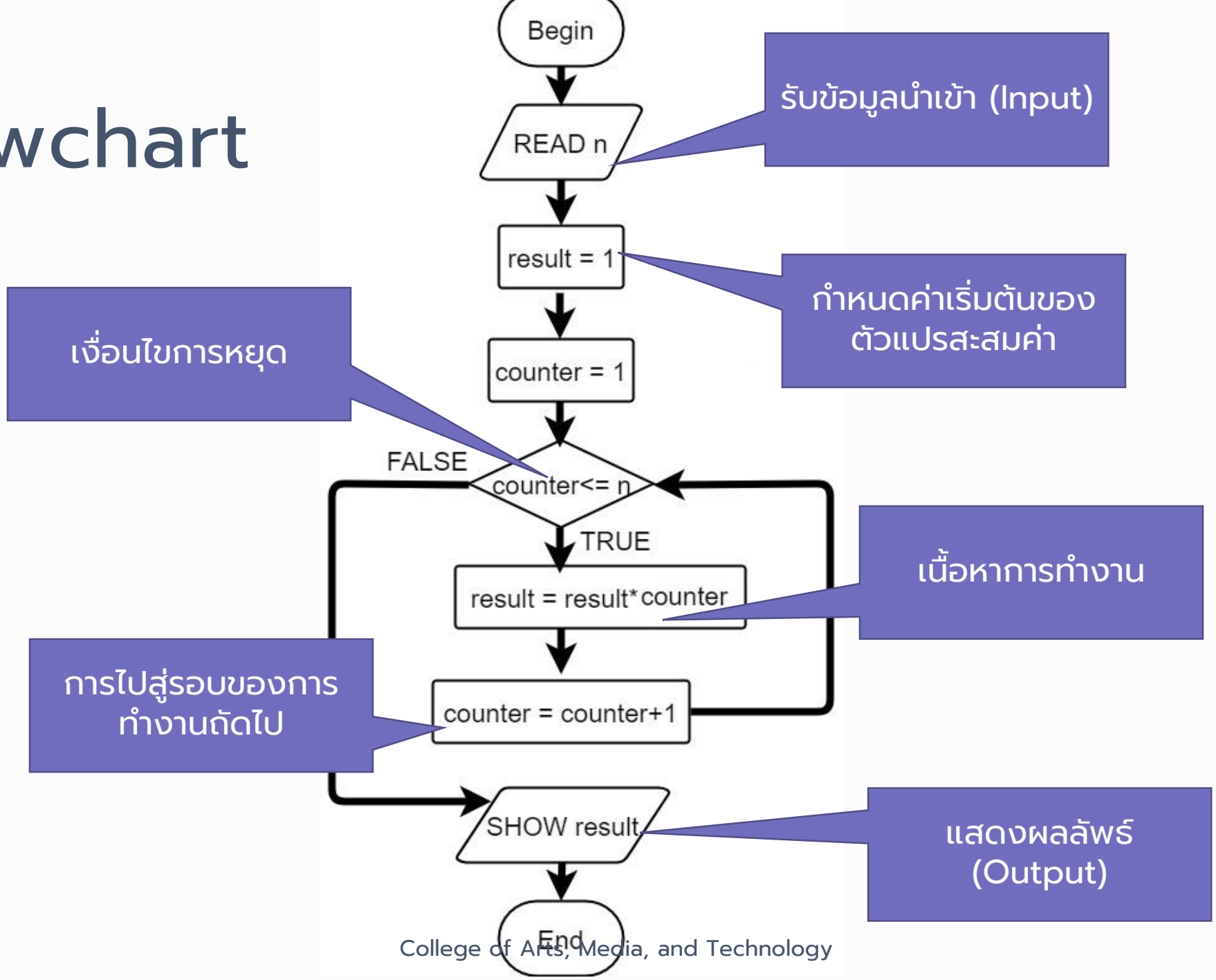
ทำไมค่าเริ่มต้นเป็น 1 ???

Flowchart

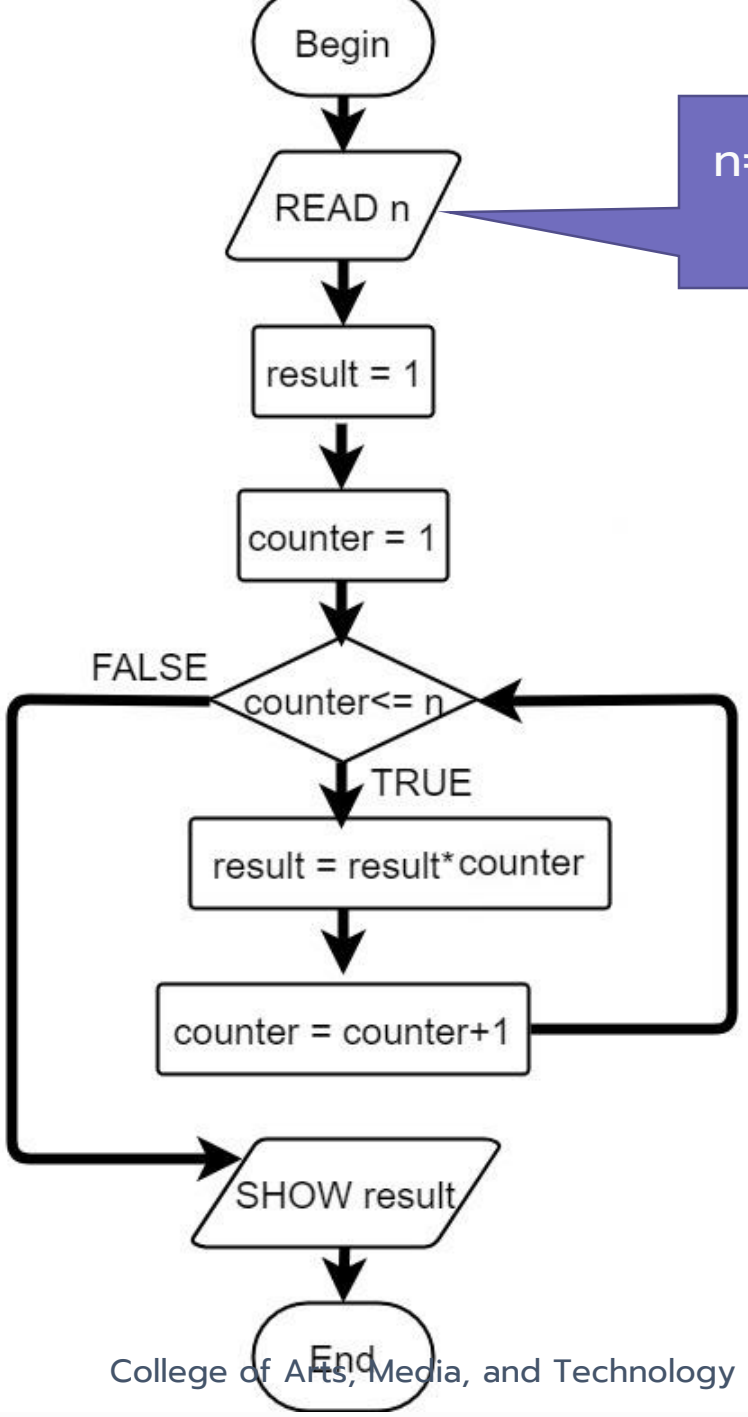

Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End

หากผู้ใช้กรอก 3 เป็นข้อมูลนำเข้า

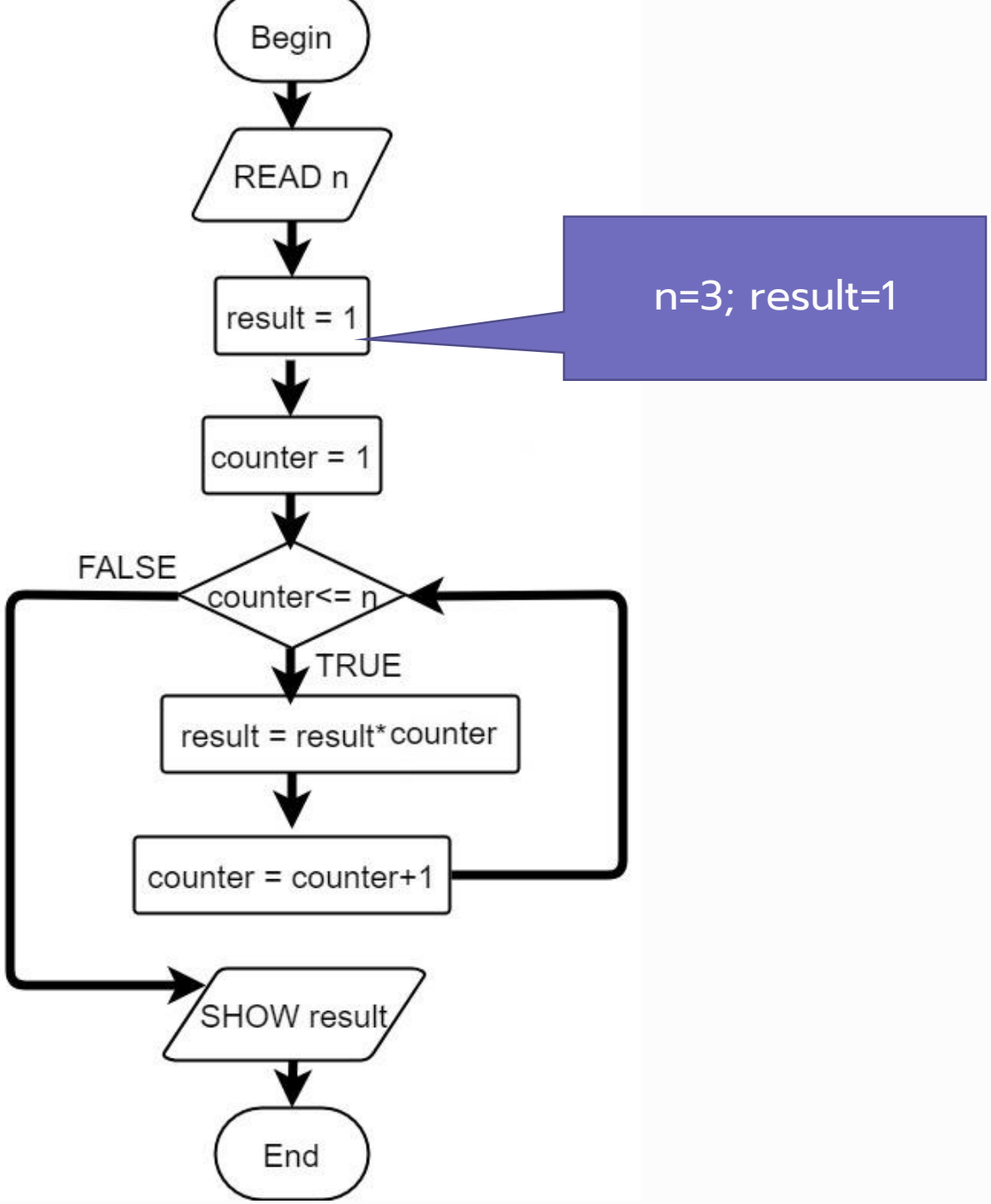
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End
รับข้อมูลนำเข้า (Input)
กำหนดค่าเริ่มต้นของ ตัวแปรสะสมค่า
เงื่อนไขการหยุด
เนื้อหาการทำงาน
การไปสู่รอบของการ ทำงานถัดไป
แสดงผลลัพธ์ (Output)

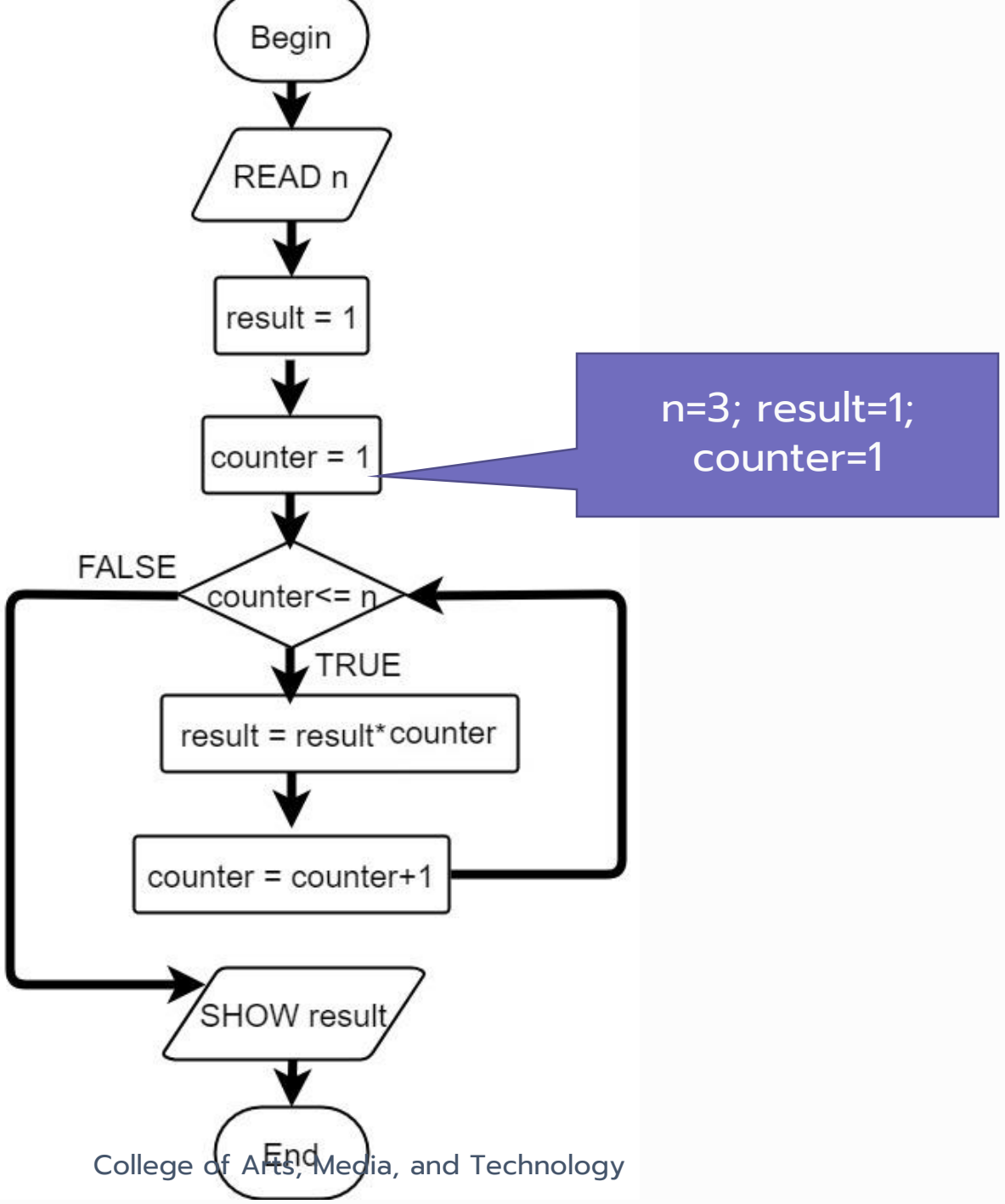
Flowchart


Begin
READ n
n=3 // User inputs 3.
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End

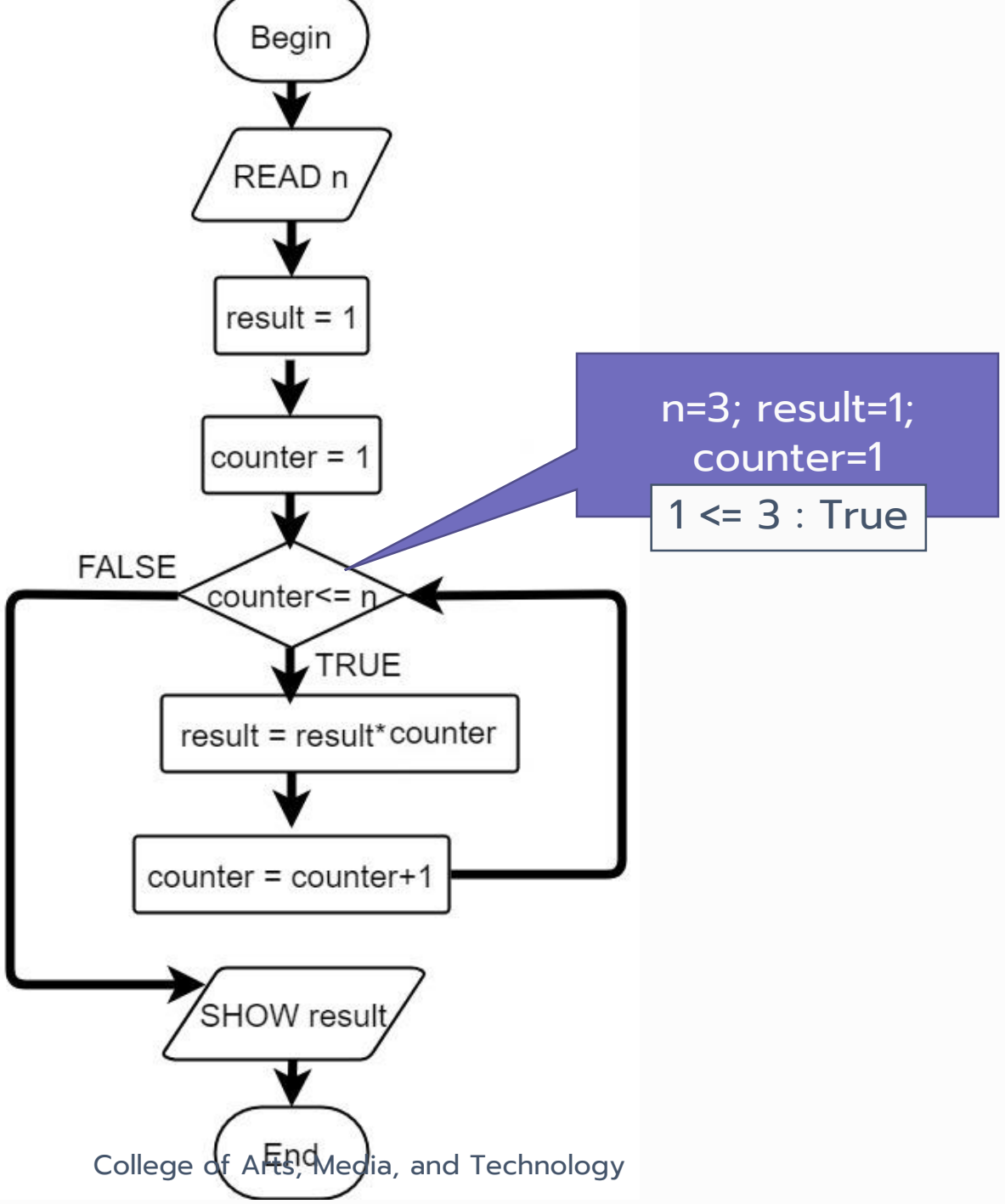
Flowchart


Begin
READ n
result = 1
n=3; result=1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End

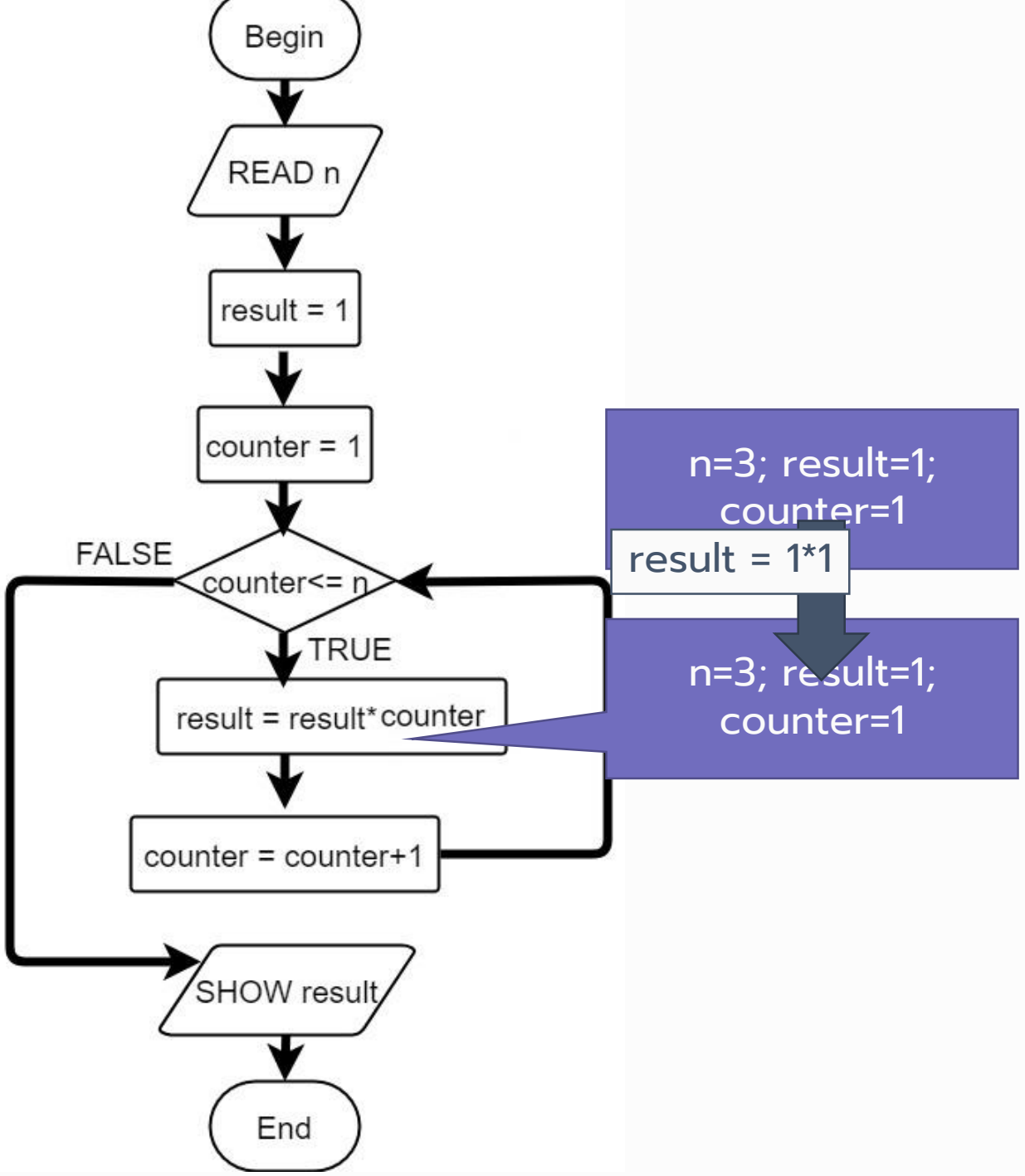
Flowchart


Begin
READ n
result = 1
counter = 1
n=3; result=1;
counter=1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End

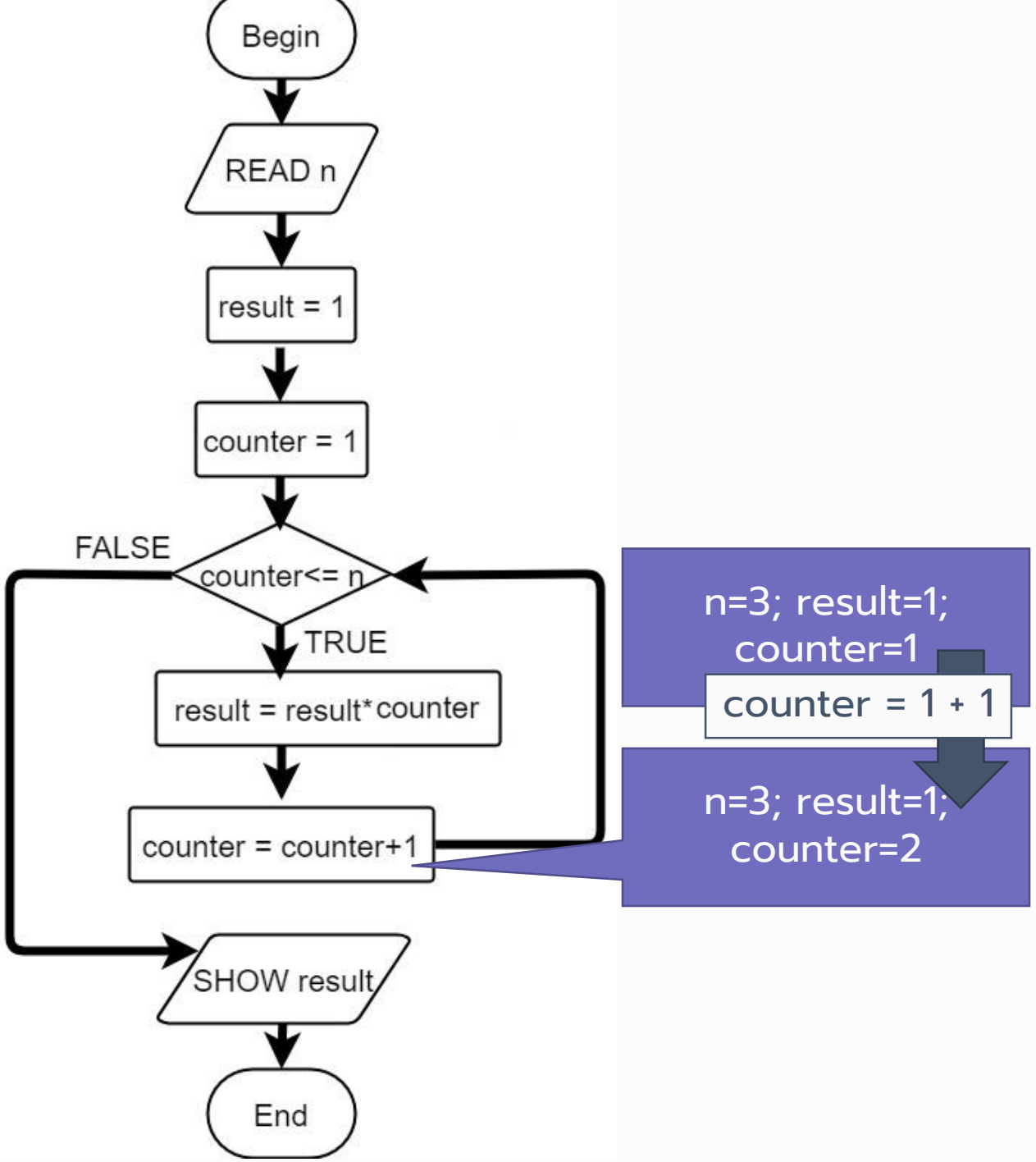
Flowchart


Begin
READ n
result = 1
counter = 1
counter<= n
FALSE
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=1;
counter=1
1 <= 3 : True

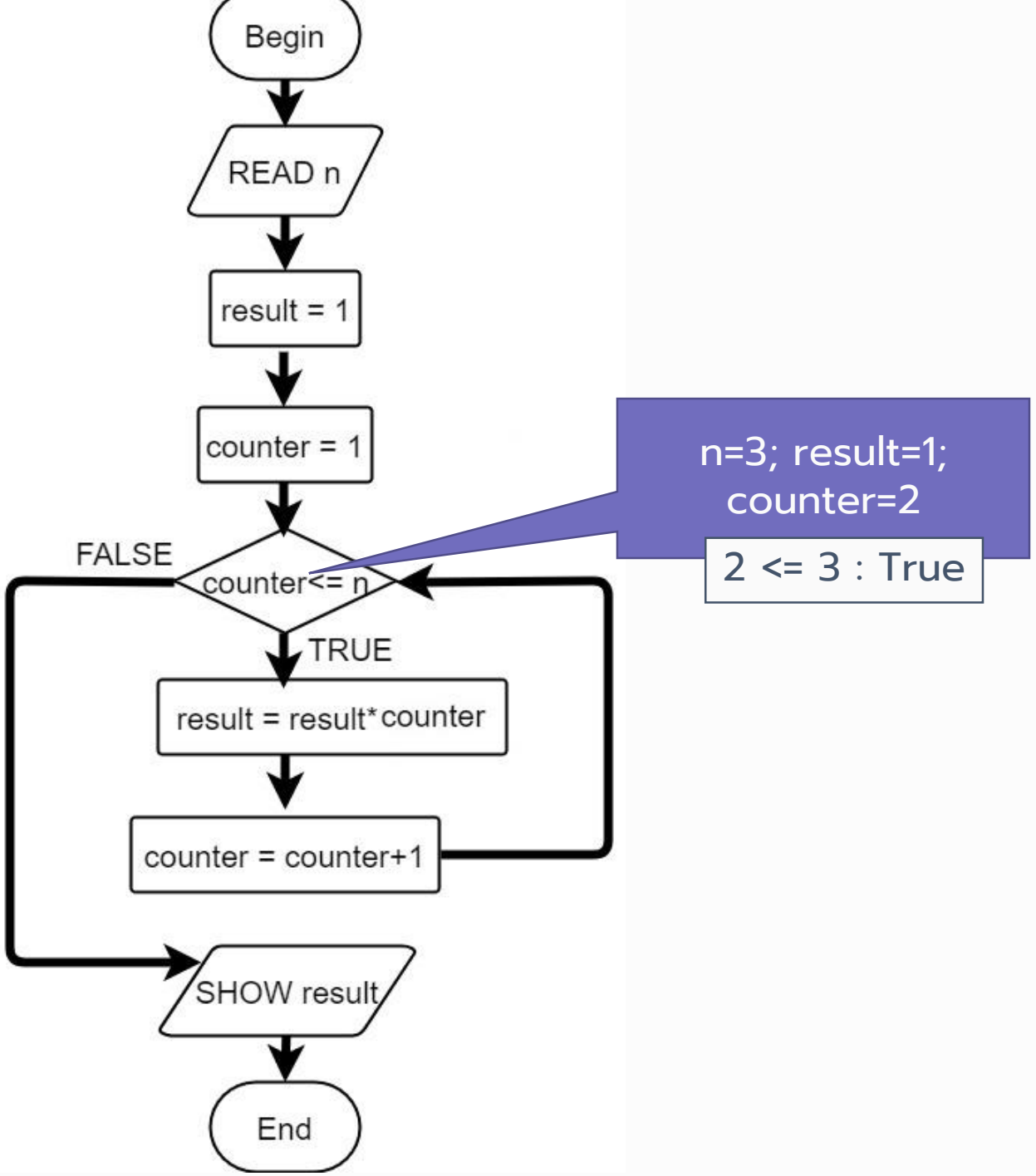
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=1; counter=1
result = 1*1
n=3; result=1; counter=1

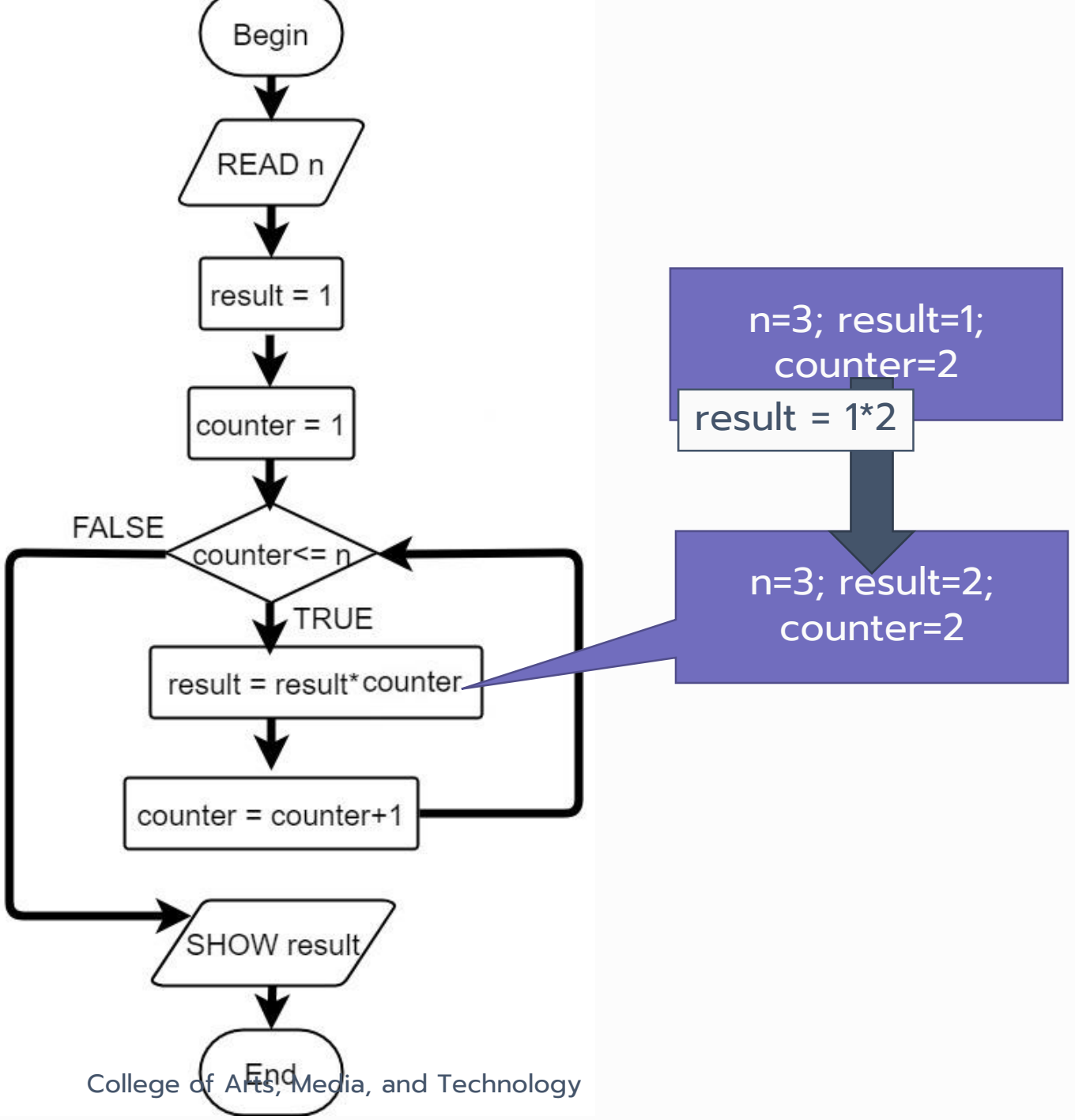
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result*counter
counter = counter+1
SHOW result
End
n=3; result=1;
counter=1
counter = 1 + 1
n=3; result=1;
counter=2

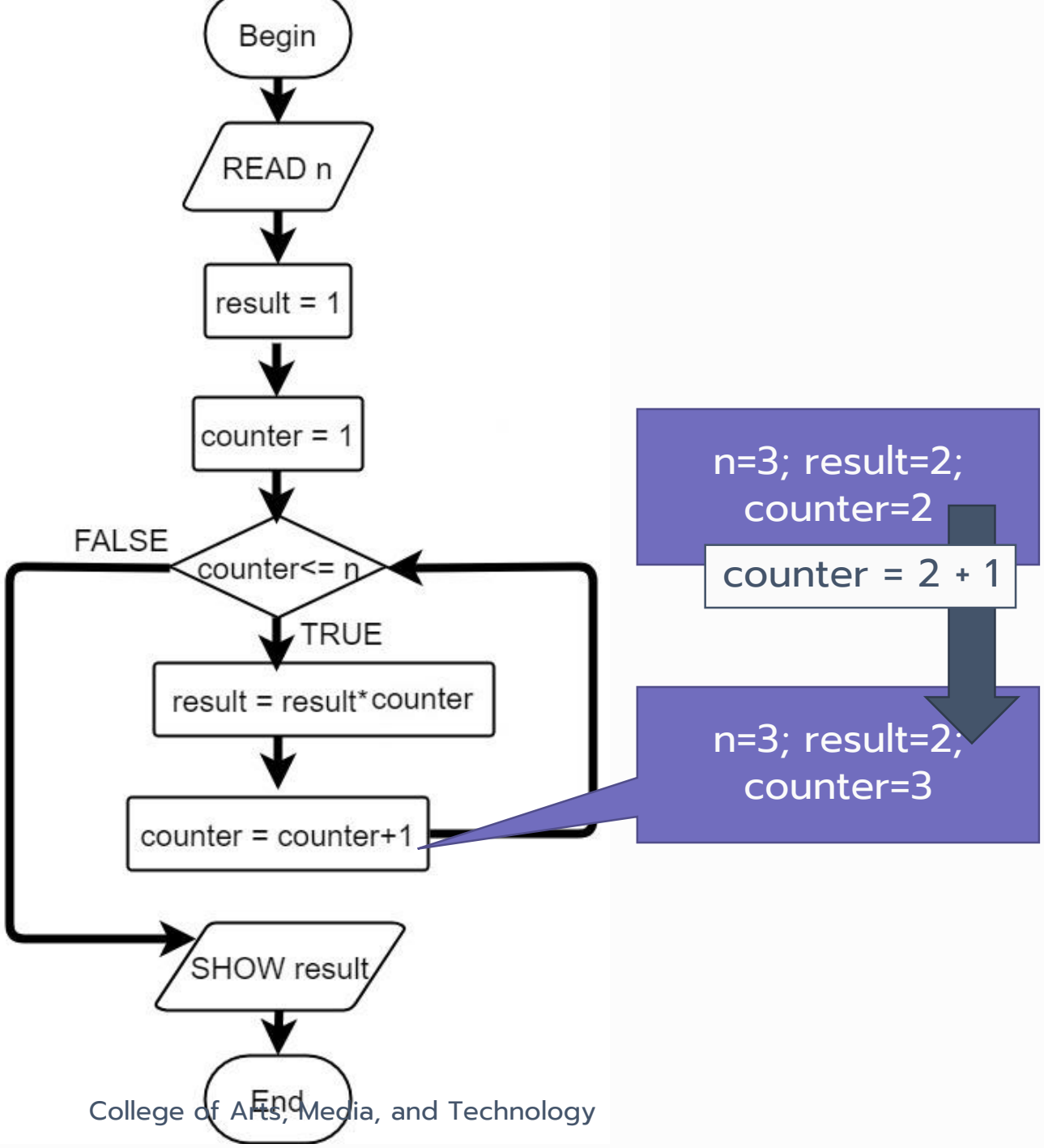
Flowchart


Begin
READ n
result = 1
counter = 1
counter<= n
FALSE
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=1;
counter=2
2 <= 3 : True

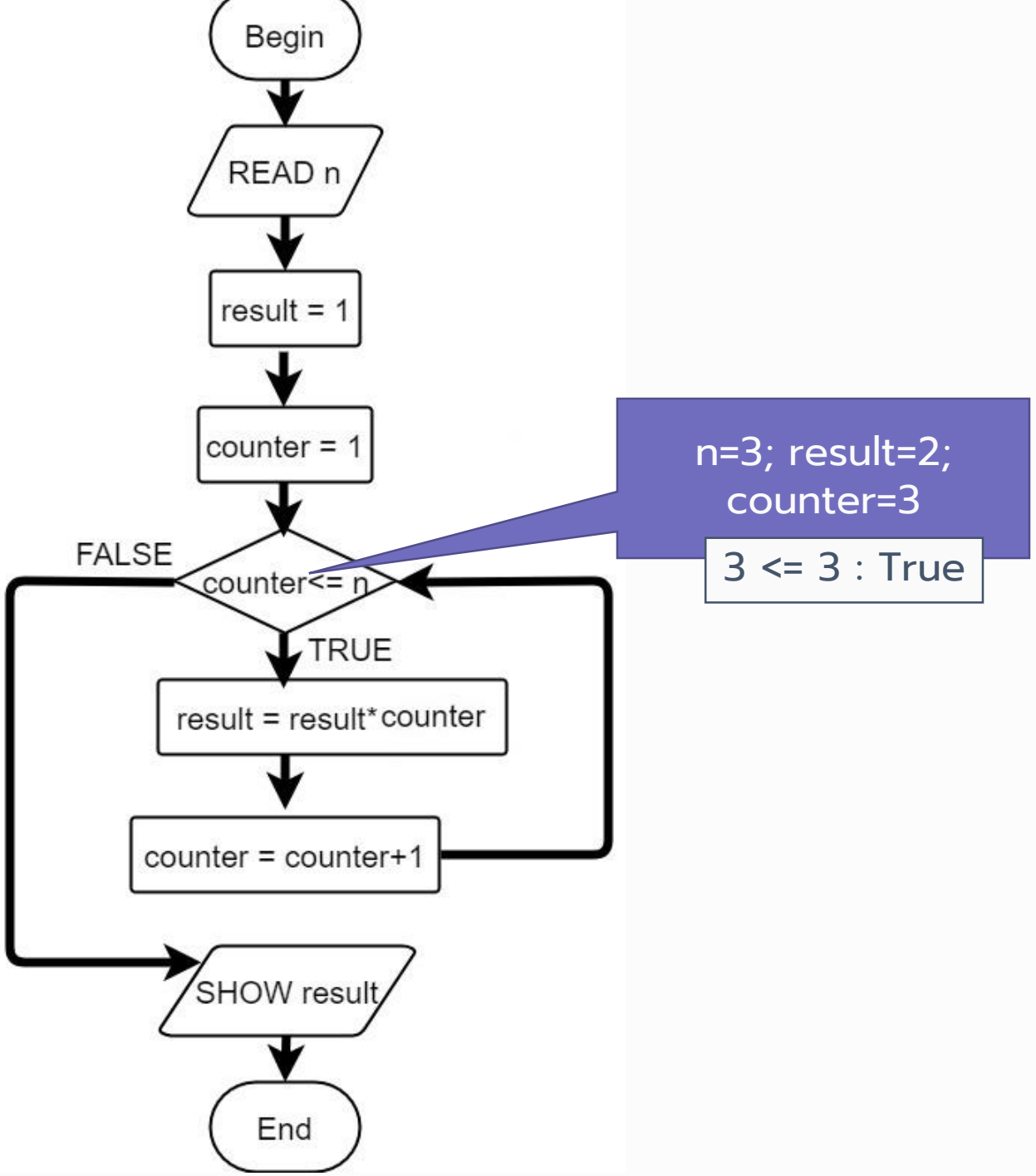
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=1;
counter=2
result = 1*2
n=3; result=2;
counter=2

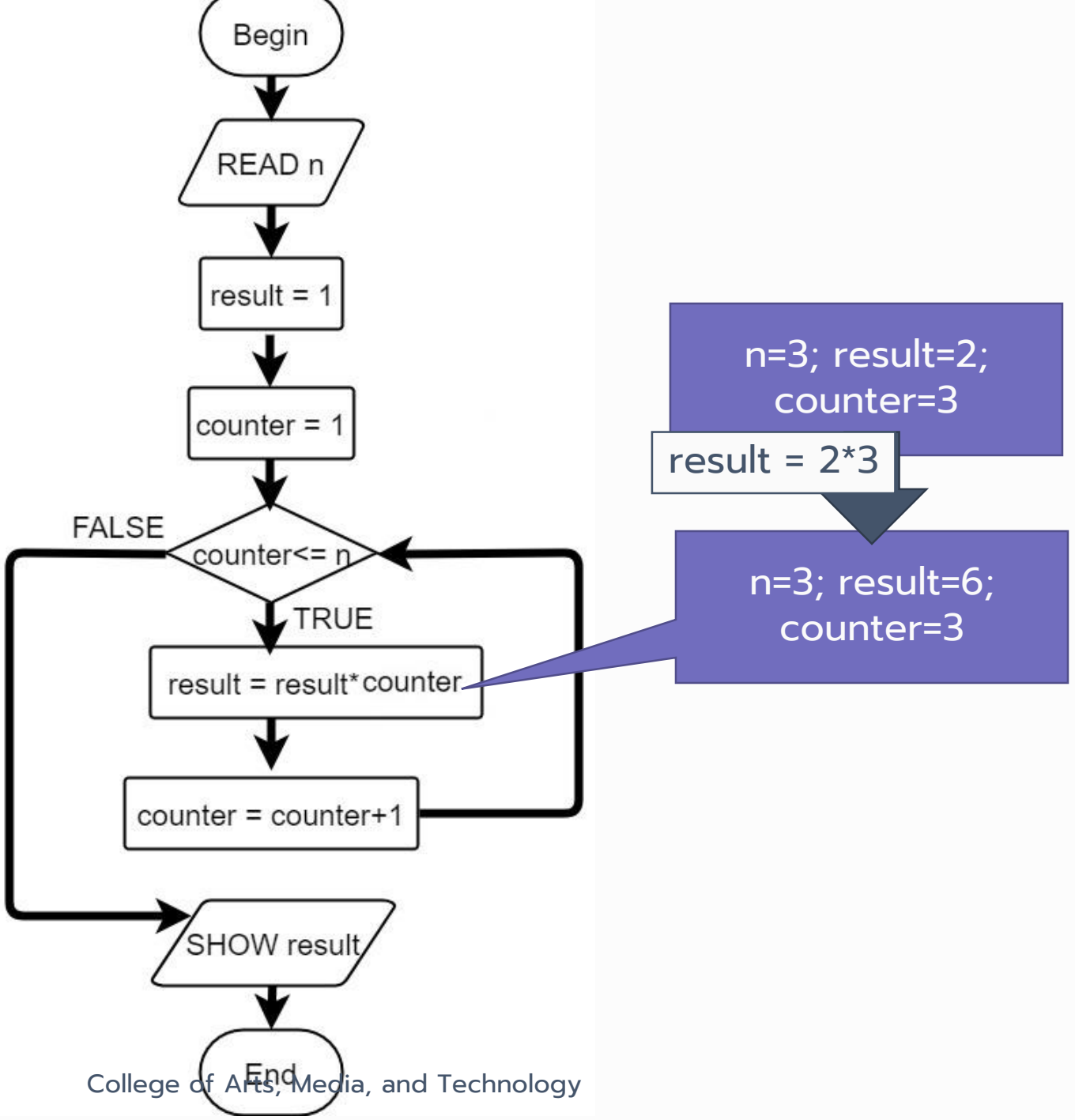
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=2;
counter=2
counter = 2 + 1
n=3; result=2;
counter=3

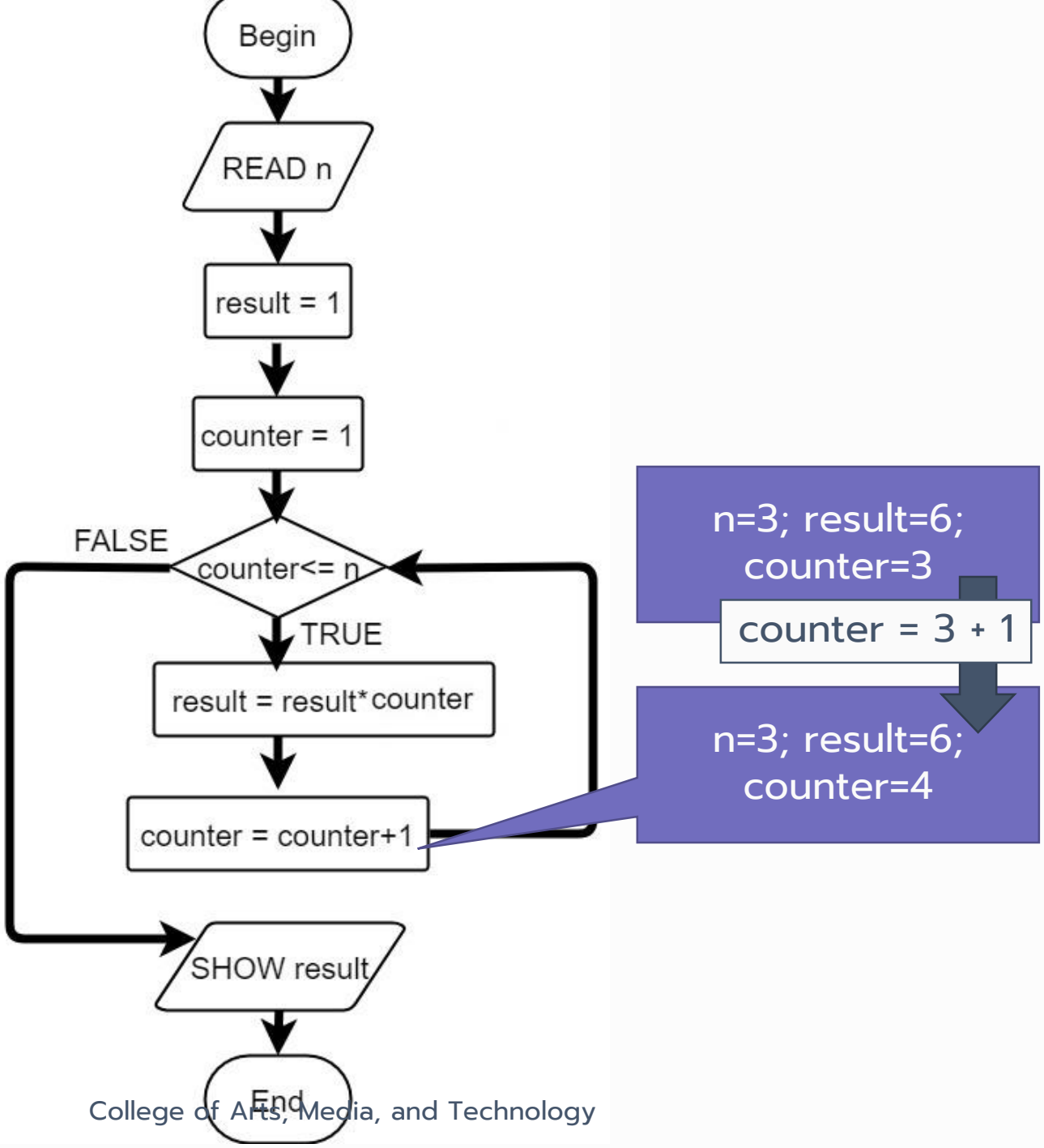
Flowchart


Begin
READ n
result = 1
counter = 1
counter<= n
FALSE
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=2;
counter=3
3 <= 3 : True

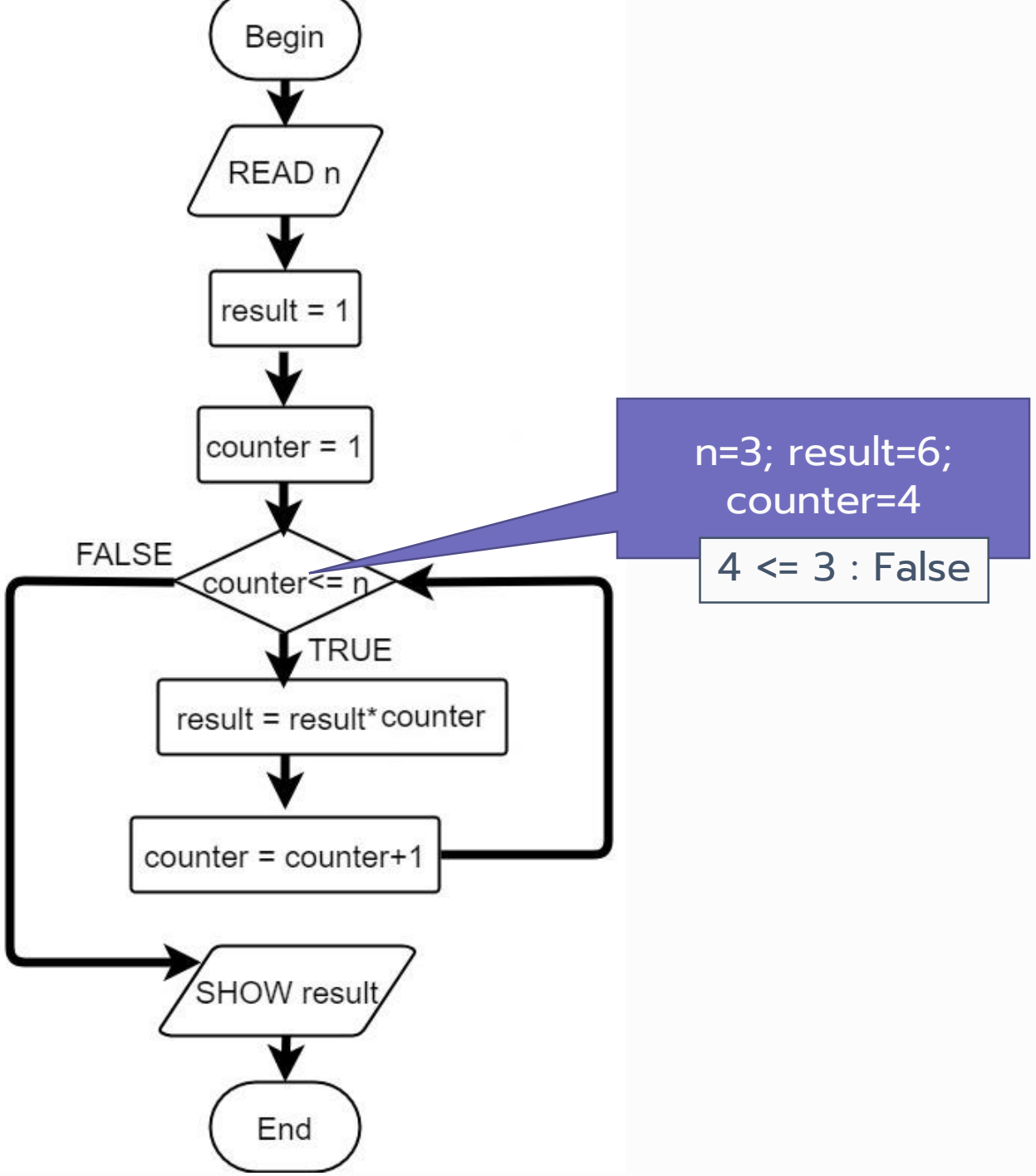
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=2;
counter=3
result = 2*3
n=3; result=6;
counter=3

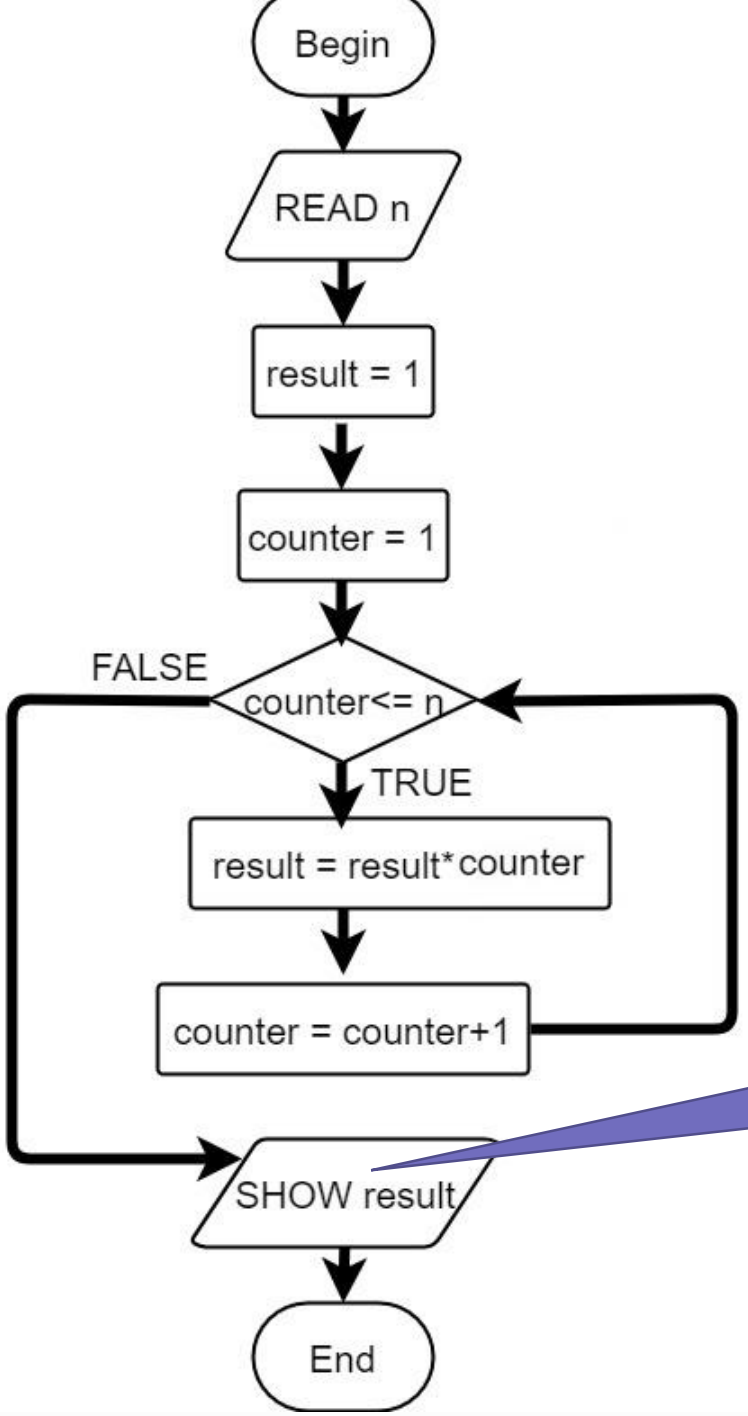
Flowchart


Begin
READ n
result = 1
counter = 1
FALSE
counter<= n
TRUE
result = result*counter
counter = counter+1
SHOW result
End
n=3; result=6;
counter=3
counter = 3 + 1
n=3; result=6;
counter=4

Flowchart

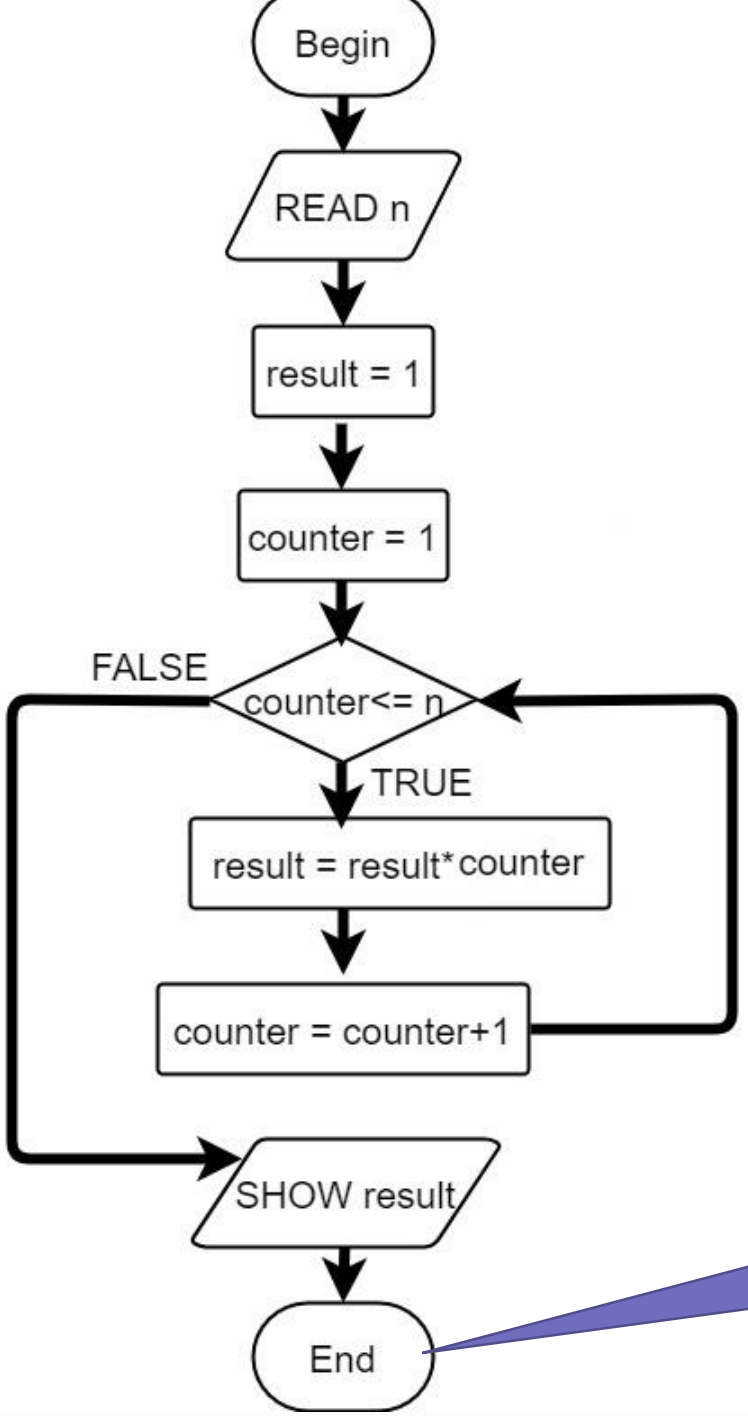

Begin
READ n
result = 1
counter = 1
counter<= n
FALSE
TRUE
result = result* counter
counter = counter+1
SHOW result
End
n=3; result=6;
counter=4
4 <= 3 : False

Flowchart



n=3; result=6; counter=4
Shows 6 to the user

Flowchart



n=3; result=6;
counter=4
// Program ends

พื้นฐานการเขียนโปรแกรมภาษาไพธอน

บทที่ 6 การทำซ้ำ

วัตถุประสงค์ของบทเรียน

เมื่อสิ้นสุดบทเรียน ผู้เรียนจะสามารถ

- อธิบายถึงหลักการการเขียนโปรแกรมแบบวนซ้ำได้
- ใช้โครงสร้างการทำซ้ำในการออกแบบโปรแกรมได้
- พัฒนาโปรแกรมที่สามารถทำซ้ำได้

กรณีศึกษา #1

จงเขียนโปรแกรมที่รับค่าจำนวนเต็มจากผู้ใช้ จำนวน 2 ค่า และโปรแกรมจะทำการหาผลรวมของตัวเลข ตั้งแต่ตัวเลขตัวแรก จนถึงตัวเลขหลัง เช่น

กรณีศึกษาที่ 1 : ผู้ใช้กรอก 1 และ 10 โปรแกรมจะแสดง 55

กรณีศึกษาที่ 2 : ผู้ใช้กรอก -10 และ 10 โปรแกรมจะแสดง 0

หมายเหตุ : เพื่อความง่าย สมมุติให้ข้อมูลตัวแรกมีค่าน้อยกว่าข้อมูลตัวหลังเสมอ


Begin
GET num1,num2
temp = num1
sum = 0
temp<=num2
End
TRUE
sum = sum+temp
temp = temp + 1

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

temp = 0
sum = 0

while (num1+temp)1 <= num2 :
  sum = sum+(num1+temp)
  temp =temp +1

print("Sum is ", sum)
```

```
Please enter the 1st number : 1
Please enter the 2nd number : 10
Sum is  55
```

ตัวแปรนับรอบ (Counter)

- ตัวแปรนับรอบ
  - คือ ตัวแปรที่ใช้ในการติดตามจำนวนรอบของการทำงาน
  - ค่าที่เก็บในตัวแปรชนิดนี้ จะเพิ่มขึ้น 1 ในทุกรอบของการทำงาน
    - ในบางกรณี อาจจะเป็นค่าอื่น
  - ในตัวอย่าง ตัวแปรนับรอบคือ temp
    - หน้าที่ของ temp คือการนับจำนวนรอบที่ทำงาน

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

temp = 0
sum = 0

while (num1+temp) <= num2 :
  sum = sum+(num1+temp)
  temp =temp +1

print("Sum is ", sum)
```

ตัวแปรนับรอบ (Counter)

- ตัวแปรนับรอบ
  - กระบวนการนับรอบจะประกอบไปด้วย 2ส่วน

1) การกำหนดค่าเริ่มต้น
   - ขั้นตอนนี้จะเป็นการกำหนดจุดเริ่มต้นของการนับรอบ
   - โดยทั่วไปแล้วจะมีค่า 0, 1 หรือ ค่าเริ่มต้น
2) การเพิ่มค่า (Update Statement)
   - ขั้นตอนนี้จะเป็นการนับรอบที่ทำงานจนเสร็จ
   - โดยทั่วไปการเพิ่มค่า จะเพิ่มทีละ 1

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

temp = 0
sum = 0

while (num1+temp) <= num2 :
  sum = sum+(num1+temp)
  temp =temp +1

print("Sum is ", sum)
```

จะเกิดอะไรขึ้นหากไม่มีการเพิ่มค่าของตัวแปรจำนวนรอบ

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

temp = 0
sum = 0

while (num1+temp)1 <= num2 :
  sum = sum+(num1+temp)
```

```
print("Sum is ", sum)
```

การทำซ้ำแบบไม่หยุด Infinite Loop

การทำซ้ำแบบจำนวนรอบชัดเจน (Fixed Iteration Loop)

- การทำซ้ำแบบจำนวนรอบชัดเจน
  - เป็นโครงสร้างการทำงานแบบทำซ้ำที่มีการใช้รอบของการทำงานในการควบคุม
  - ใช้กลไกของตัวแปรนับรอบ และใช้ตัวแปรนับรอบในเงื่อนไขการหยุดการทำงาน
- ในการเขียนโปรแกรม การทำซ้ำแบบจำนวนรอบชัดเจนถูกเรียกว่า For-Loop

การทำซ้ำแบบจำนวนรอบชัดเจน (Fixed Iteration Loop)

- คำสั่ง For-loop ในภาษา Python

```
for [counter] in [ranges] :
    statement 1
    statement 2
    statement 3
    ...
```

การทำซ้ำแบบจำนวนรอบชัดเจน (Fixed Iteration Loop)

- คำสั่ง range()
  - คำสั่ง range เป็นคำสั่งที่ใช้ในการสร้างรายการของตัวเลข เพื่อใช้ในการนับรอบ
  - การเรียกใช้คำสั่ง range มี 2 รูปแบบหลัก คือ
    - range(n) – การเรียกใช้คำสั่งในรูปแบบนี้ คือการสร้างรายการจาก 0,1,2,... จนถึง n-1 เช่น

range(5) = 0,1,2,3,4

    - range(n,m,l) - การเรียกใช้คำสั่งในรูปแบบนี้ คือการสร้างรายการจาก n,(n+l),(n+2*l),... จนถึง m-1 เช่น

range(2,6,2) = 2,4

การทำซ้ำแบบจำนวนรอบชัดเจน (Fixed Iteration Loop)

ในแต่ละรอบของการทำงาน การทำซ้ำแบบจำนวนรอบชัดเจนจะมีขั้นตอนดังนี้

1) กระบวนการกำหนดค่าเริ่มต้น
   - ค่าตัวแปรนับรอบจะถูกกำหนดเป็นค่าเริ่มต้นโดยอัตโนมัติ เป็นส่วนหนึ่งของคำสั่ง For-loop
   - ขั้นตอนนี้ถูกทำ 1 ครั้งก่อนเริ่มกระบวนการการทำซ้ำ

2) กระบวนการทำซ้ำ
   - เงื่อนไขการหยุดจะถูกกำหนดโดยใช้ตัวแปรนับรอบเป็นส่วนประกอบ
   - กระบวนการทำซ้ำจะเป็นการทำเนื้อหาของการทำซ้ำ (Loop Body)
   - การทำซ้ำจะถูกทำเมื่อ เงื่อนไขการหยุดเป็นจริง และหยุดการทำงานเมื่อ เงื่อนไขการหยุดเป็นเท็จ
     - เงื่อนไขของ For-loop คือ รายการที่ใช้นับรอบถูกใช้จนหมด

การทำซ้ำแบบจำนวนรอบชัดเจน (Fixed Iteration Loop)

Round (temp)	sum
- (before loop execution)	0
0	0 + (0+1) = 1
1	1 + (1+1) = 3
2	3 + (2+1) = 6
3	6 + (3+1) = 10
4	10 + (4+1) = 15
Exit Loop	15

num1 : 1

num2 : 5

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

sum = 0

for temp in range(num2-num1+1):
  sum = sum + (temp+num1)

print("Sum is ", sum)
```

การทำซ้ำแบบจำนวนรอบชัดเจน (Fixed Iteration Loop)

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

temp = 0
sum = 0

while (num1+temp) <= num2 :
  sum = sum+(num1+temp)
  temp =temp +1

print("Sum is ", sum)
```

```
num1 = (int)(input("Please enter the 1st number : "))
num2 = (int)(input("Please enter the 2nd number : "))

sum = 0


for temp in range(num2-num1+1):
  sum = sum + (temp+num1)


print("Sum is ", sum)
```

While loop และ For loop สามารถใช้แทนกันได้

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Off-by-one Error

- หรือถูกเรียกว่า OBOE, OBOB, OB1

- ปัญหานี้คือ เหตุการณ์ที่จำนวนรอบของการทำงานของโครงสร้างการทำซ้ำไม่เป็นไปตามที่ออกแบบ โดยที่อาจจะทำงาน**เกิน 1 รอบ** หรือ **น้อยกว่า 1 รอบ** จากที่คาดหวัง

- ที่มาของปัญหา
  - การกำหนดค่าเริ่มต้นไม่ถูกต้อง
  - ใช้เงื่อนไขในการหยุดการทำซ้ำไม่สอดคล้องกับค่าเริ่มต้น

- ตัวอย่าง – จงเขียนโปรแกรมเพื่อแสดงตัวเลขจำนวน 10 ตัว บนหน้าจอ

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Off-by-one Error

- กรณีที่ทำงาน**ถูกต้อง**

Round No.	Counter
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9

10 รอบ

counter เริ่มต้นที่ 0

```
for counter in range(0,10,1):
    print(counter)
```

10 - 1 = 9

counter สิ้นสุดที่ 9

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Off-by-one Error

- กรณีที่ทำงาน **off-by-one (เกิน 1 รอบ)**

Round No.	Counter
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10

11 รอบ

counter เริ่มต้นที่ 0

```
for counter in range(0,11,1):
    print(counter)
```

11 - 1 = 10

counter สิ้นสุดที่ 10

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Off-by-one Error

- กรณีที่ทำงาน **off-by-one (น้อยไป 1 รอบ)**

Round No.	Counter
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

counter เริ่มต้นที่ 1

9 รอบ

```
for counter in range(1,10,1):
    print(counter)
```

10 - 1 = 9

counter สิ้นสุดที่ 9

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ – Infinite Loop

- เป็นปัญหาที่พบได้บ่อยที่สุดในการใช้โครงสร้างการทำซ้ำ
- ปัญหานี้คือ เหตุการณ์ที่เงื่อนไขการหยุดของโครงสร้างการทำซ้ำไม่มีทางเป็นเท็จ ส่งผลให้โครงสร้างการทำซ้ำไม่หยุดทำงาน – มักจะเกิดกับการใช้ while
- ที่มาของปัญหา
  - การกำหนดเงื่อนไขการหยุดที่ไม่ถูกต้อง
  - การเปลี่ยนแปลงการนับรอบไม่ถูกต้อง / ไม่มี
- ตัวอย่าง – จงเขียนโปรแกรมเพื่อแสดงตัวเลขจำนวน 10 ตัว บนหน้าจอ

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Infinite Loop

- กรณีที่ทำงาน**ถูกต้อง**

Round No.	Counter
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

10 รอบ

counter เริ่มต้นที่ 1

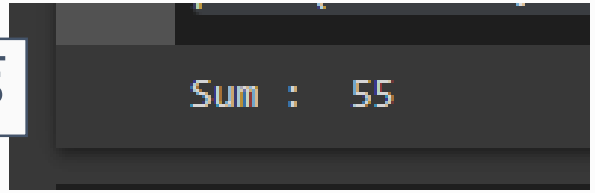
counter สิ้นสุดที่ 10

```
counter = 1
sum = 0

while counter <= 10 :
  sum = sum + counter
  counter = counter + 1

print("Sum : ",sum)
```

ผลลัพธ์



ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Infinite Loop

- กรณีที่ทำงาน **infinite loop (เงื่อนไขหยุดผิด)**

Round No.	Counter
1	1
2	2
3	3
4	4
5	5
...	...

counter เริ่มต้นที่ 1

infinite รอบ

```
counter = 1
sum = 0

while counter > 0 :
  sum = sum + counter
  counter = counter + 1

print("Sum : ",sum)
```

counter เพิ่มขึ้นไปยัง infinity

ปัญหาที่พบได้ทั่วไปของการใช้โครงสร้างการทำซ้ำ - Infinite Loop

- กรณีที่ทำงาน **infinite loop (ไม่มีการเพิ่มตัวแปรนับรอบ)**

Round No.	Counter
1	1
2	1
3	1
4	1
5	1
...	...

counter เริ่มต้นที่ 1

```
counter = 1
sum = 0

while counter <= 10 :
  sum = sum + counter
  counter = counter + 1

print("Sum : ",sum)
```

infinite รอบ

counter คงที่ มีค่าเป็น 1

การติดตามการเปลี่ยนแปลงค่า (Program Tracing)

การติดตามการเปลี่ยนแปลงค่า

- เป็นวิธีการในการติดตามการเปลี่ยนแปลงของตัวแปร ในการถูกเรียกใช้ทีละบรรทัด เพื่อ
  1) ทำความเข้าใจในการทำงาน
  2) ติดตามหา error

- ทำการจำลองการทำงานของโปรแกรมที่ออกแบบขึ้นมา
  - ทำการจำลองการทำงานในแต่ละบรรทัด ของโปรแกรม
  - มุ่งเน้นที่การเปลี่ยนแปลงของค่าที่ถูกเก็บในตัวแปร

การติดตามการเปลี่ยนแปลงค่า

Step	Variable 1	Variable 2	...	Variable n

แต่ละแถว แสดงถึงการทำงานในแต่ละคำสั่ง

แต่คอลัมน์ แสดงถึงการเปลี่ยนแปลงของตัวแปรแต่ละตัว

คอลัมน์แสดงคำสั่งการทำงาน

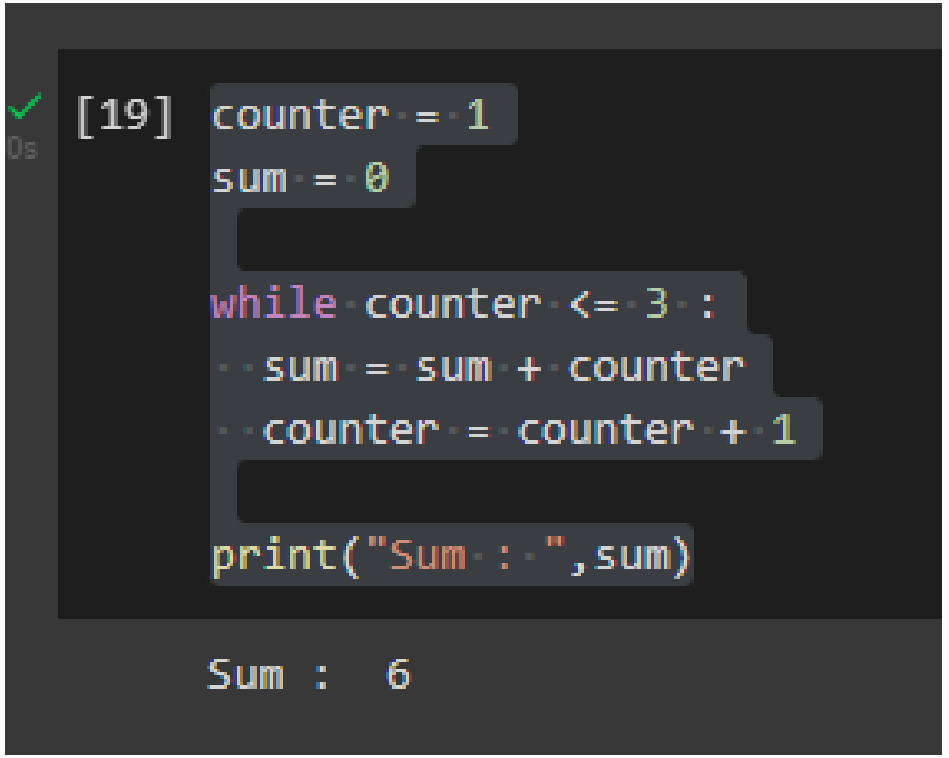
การติดตามการเปลี่ยนแปลงค่า

- กรณีที่ทำงาน**<u>ถูกต้อง</u>**

```
counter = 1
sum = 0

while counter <= 3 :
  sum = sum + counter
  counter = counter + 1

print("Sum : ",sum)
```



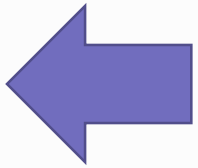
การติดตามการเปลี่ยนแปลงค่า

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 3 :
[3]     sum = sum + counter
[4]     counter = counter + 1

[5]   print("Sum : ",sum)
```



Step	counter	sum
1	1	-

การกำหนดค่าเริ่มต้น

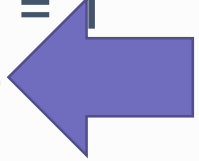
การติดตามการเปลี่ยนแปลงค่า

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 3 :
[3]     sum = sum + counter
[4]     counter = counter + 1

[5]   print("Sum : ",sum)
```



Step	counter	sum
1	1	-
2	1	0

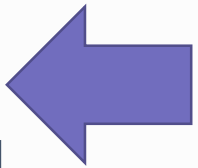
การติดตามการเปลี่ยนแปลงค่า

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 3 :
[3]     sum = sum + counter
[4]     counter = counter + 1

[5]   print("Sum : ",sum)
```



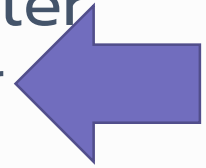
Step	counter	sum
1	1	-
2	1	0
3	1	1

ทำการตรวจสอบเงื่อนไขของ while แต่ไม่ได้ทำการบันทึกในตาราง เพราะไม่ทำให้ค่าของตัวแปรเปลี่ยนแปลง

การติดตามการเปลี่ยนแปลงค่า

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 3 :
[3]     sum = sum + counter
[4]     counter = counter


[5]   print("Sum : ",sum)
```

Step	counter	sum
1	1	-
2	1	0
3	1	1
4	2	1

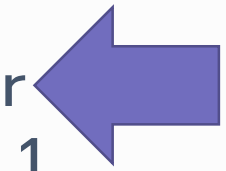
การติดตามการเปลี่ยนแปลงค่า

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 3 :
[3]     sum = sum + counter
[4]     counter = counter + 1

[5]   print("Sum : ",sum)
```



Step	counter	sum
1	1	-
2	1	0
3	1	1
4	2	1
3	2	3

ทำการทำซ้ำ เนื่องจากเงื่อนไขเป็นจริง

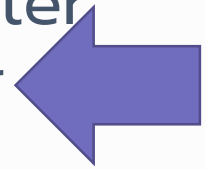
การติดตามการเปลี่ยนแปลงค่า

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 3 :
[3]     sum = sum + counter
[4]     counter = counter

[5]   print("Sum : ",sum)
```



Step	counter	sum
1	1	-
2	1	0
3	1	1
4	2	1
3	2	3
4	3	3
...	...	...

ทำจนกระทั่งโปรแกรมสิ้นสุด

การติดตามการเปลี่ยนแปลงค่า

- กรณีที่ทำงาน **off-by-one (เกิน 1 รอบ)**

Line no.

```
[1]   counter = 0
[2]   sum = 0

      while counter <= 10 :
[3]     sum = sum + counter
[4]     counter = counter + 1

[5]   print("Sum : ",sum)
```

Step	counter	sum
1	0	-
2	0	0
3	0	0
4	1	0
3	1	1
4	2	1
...	...	...
3	10	55
4	11	55
5	11	55

การติดตามการเปลี่ยนแปลงค่า

- กรณีที่ทำงาน **off-by-one (น้อยไป 1 รอบ)**

Line no.

```
[1]   counter = 1
[2]   sum = 0

      while counter <= 10 :
[3]     sum = sum + counter
[4]     counter = counter + 1

[5]   print("Sum : ",sum)
```

Step	counter	sum
1	1	-
2	1	0
3	1	1
4	2	1
3	2	3
4	3	3
...	...	...
3	10	45
4	10	55
5	11	55

พื้นฐานการเขียนโปรแกรมภาษาไพธอน

โครงสร้างเชิงซ้อน

โครงสร้างเชิงซ้อน

- โครงสร้างเชิงซ้อน คือ การเรียกใช้โครงสร้างการเขียนโปรแกรม ซ้อนในโครงสร้างการเขียนโปรแกรมอีกโครงสร้างหนึ่ง

- หลักการทำงานของ โครงสร้างเชิงซ้อน คือ 1 รอบการทำงานของโครงสร้างอยู่ภายนอกกว่า มีหมายถึงการที่โครงสร้างอยู่ภายในกว่าถูกทำจนเสร็จสิ้น

- โครงสร้างที่นำมาซ้อนกันอาจจะเป็น โครงสร้างตัดสินใจ หรือ โครงสร้างการทำซ้ำ ซ้อนทับในโครงสร้างตัดสินใจ หรือ โครงสร้างการทำซ้ำ ได้

โครงสร้างเชิงซ้อน

```
counter_1 = 0
while counter_1 <= 3 :
  for counter_2 in range(3):
    print("Counter 1 :",counter_1," Counter 2 :",counter_2)
  counter_1 = counter_1 +1
```

```
Counter 1 : 0  Counter 2 : 0
Counter 1 : 0  Counter 2 : 1
Counter 1 : 0  Counter 2 : 2
Counter 1 : 1  Counter 2 : 0
Counter 1 : 1  Counter 2 : 1
Counter 1 : 1  Counter 2 : 2
Counter 1 : 2  Counter 2 : 0
Counter 1 : 2  Counter 2 : 1
Counter 1 : 2  Counter 2 : 2
Counter 1 : 3  Counter 2 : 0
Counter 1 : 3  Counter 2 : 1
Counter 1 : 3  Counter 2 : 2
```

โครงสร้างเชิงซ้อน

```
for counter_1 in range(3):
    if counter_1 % 2 == 0:
      print("Counter : ",counter_1)
```