w02-Lab

Conditionals and Iteration

Assembled for 204217

2015 S1

by Kittipitch Kuptavanich

Basic Program Instructions

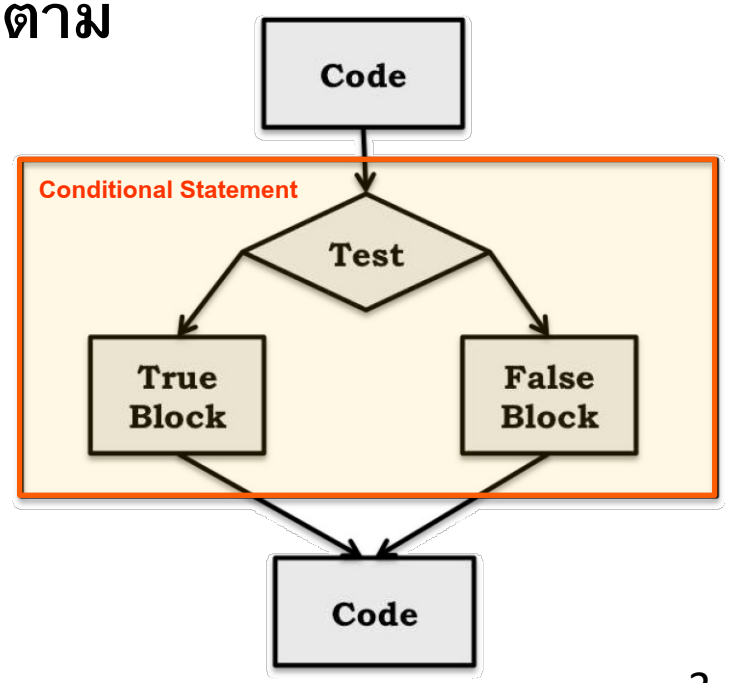
A few basic instructions appear in just about every language:

- Input
- Output
- Math
- **Conditional Execution**
- Repetition

เราสามารถพิจารณาการเขียนโปรแกรมว่าเป็นการแบ่งปัญหาที่ใหญ่และซับซ้อน ลงเป็นปัญหาย่อยที่เล็ก และซ้อบซ้อนน้อยลง จนกว่าจะสามารถแก้ปัญหาย่อย ๆ นั้น ๆ ได้ ด้วยชุดคำสั่งพื้นฐานดังกล่าว

Conditional Execution

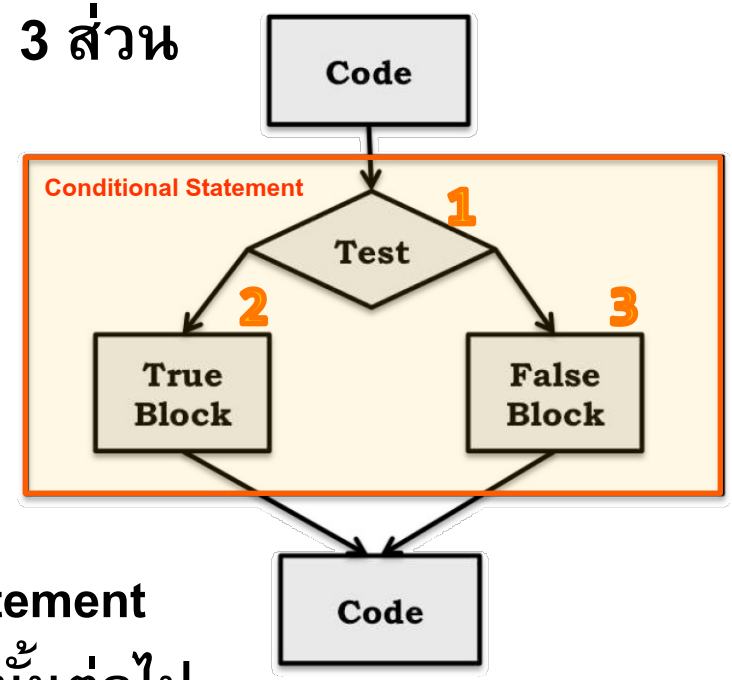
- ในการพัฒนาโปรแกรม หลาย ๆ ครั้งที่เราจำเป็นต้องมีการพิจารณาเงื่อนไขต่าง ๆ และออกแบบโปรแกรมให้ทำงานต่างกันไปตามเงื่อนไขนั้น ๆ
- รูปแบบพื้นฐานที่สุดของการทำงานตามเงื่อนไข คือ `if` **statement**



Conditional Execution [2]

- **Condition Statement ประกอบด้วย 3 ส่วน**

1. **Test**
2. **Block of Code when Test is True**
3. ***Optional* Block when Test is False**



- **หลังจากการดำเนินการ conditional statement แล้ว ก็จะดำเนินการในชุดคำสั่งหลังจากนั้นต่อไป**
- **ในภาษา Python**

  Conditional Statement จะอยู่ในรูป

```
if Boolean expression:
    block of code
else:                    optional
    block of code
```

Conditional Execution

- ชุดคำสั่งเงื่อนไขที่มีรูปแบบที่ง่ายที่สุด

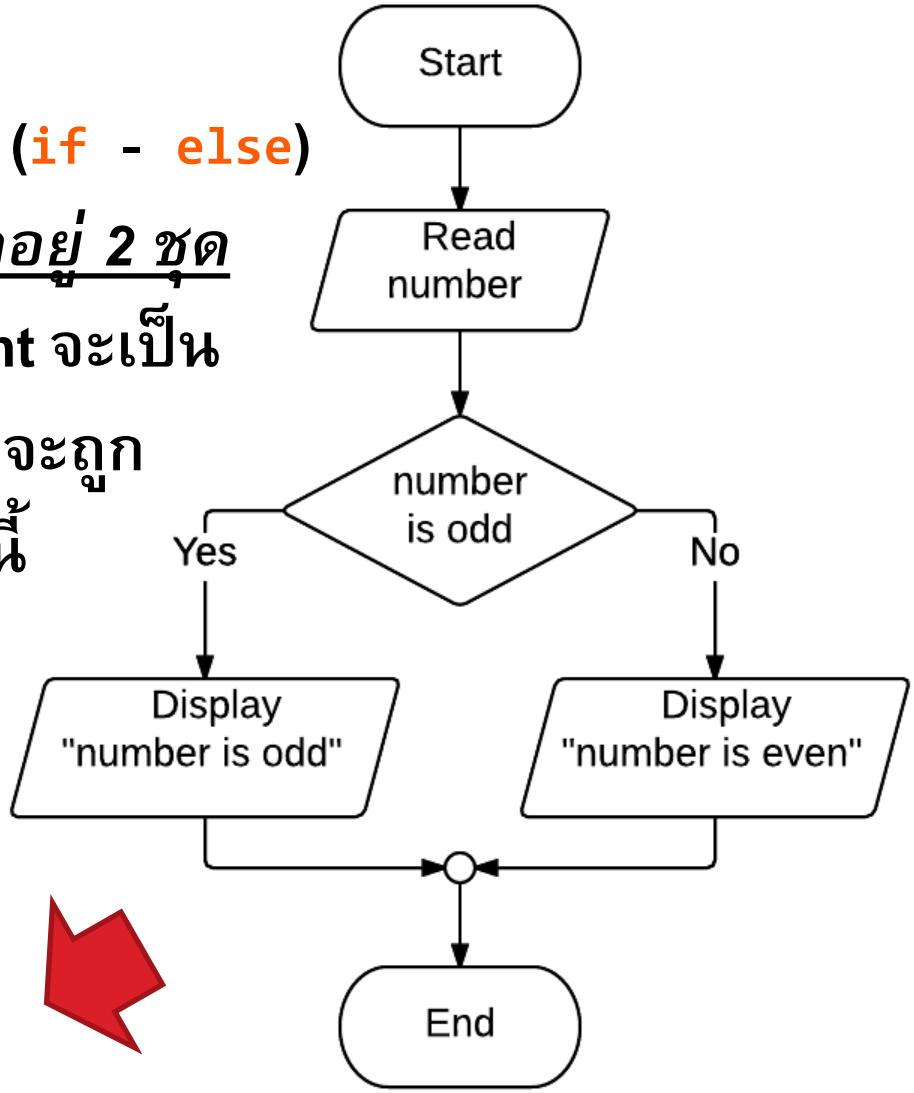
```
if Boolean expression:
    block of code
```

- เราเรียก **Boolean Expression** ในกรณีนี้ว่า **Condition**
- `if` **Statement** มีลักษณะเหมือน **Function Definition** คือ
  - มี ส่วนบรรทัดแรกเป็น **Header** (ตามด้วย **Colon** `:`) และมีส่วน **Body** ที่ต้องย่อหน้า
  - เราเรียก **Statement** ในลักษณะนี้ว่า **Compound Statement**

```
01 if x > 0:
02     print("x is positive")
```

Alternative Execution

- **`if` Statement ในรูปแบบที่ 2 (`if - else`) คือมีชุดคำสั่งที่เป็น*ทางเลือกอยู่ 2 ชุด*** **โดยที่ Conditional Statement จะเป็นตัวกำหนดว่า คำสั่งชุดไหนที่จะถูกดำเนินการ โดยมีรูปแบบดังนี้**




```
if Boolean expression:
    block of code
else:
    block of code
```

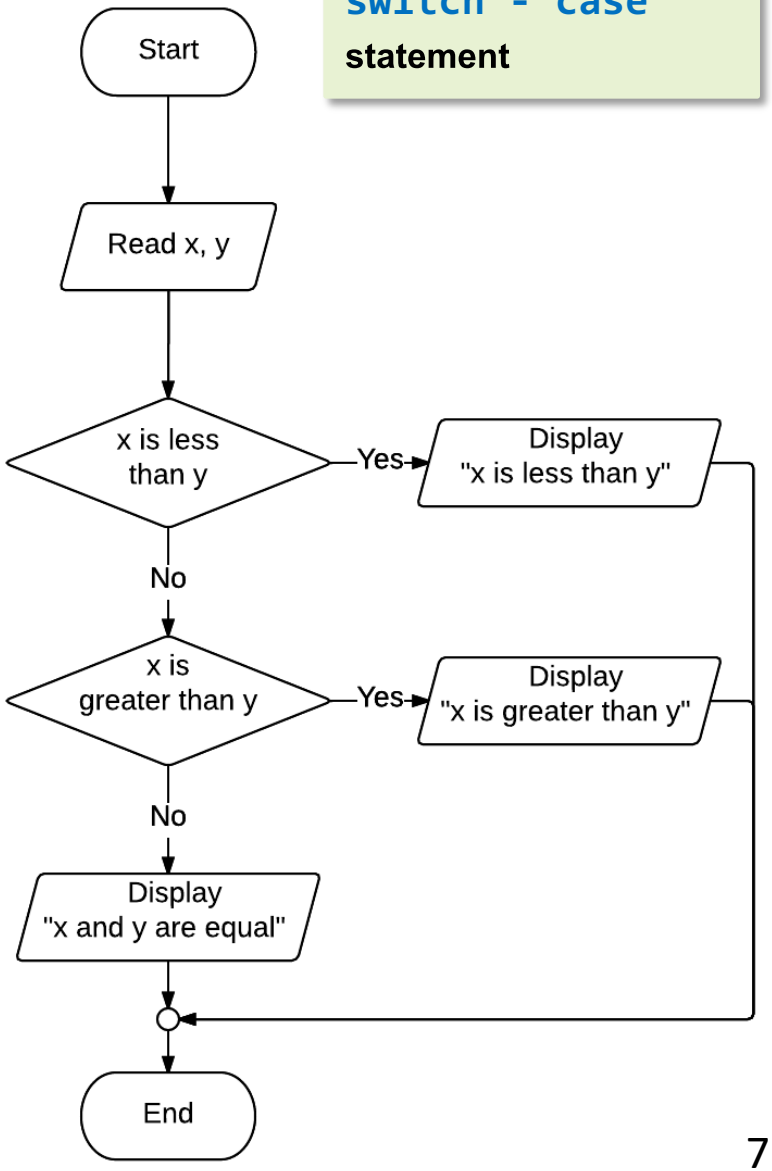
```
if _____________________:
    _____________________
else:
    _____________________
```

Chained Conditionals

Note: python ไม่มี `switch - case` statement

- ในบางกรณี ทางเลือกที่เป็นไปได้อาจมีมากกว่า 2 ทาง เราสามารถใช้ chained condition (`if - elif - else`) เพื่อรองรับเงื่อนไขการตัดสินใจในลักษณะนี้
- `elif` คือตัวย่อของ "else if"
- จากตัวเลือกที่เป็นไปได้ทั้งหมด ชุดคำสั่งเพียง 1 ชุดเท่านั้นที่จะถูกดำเนินการ

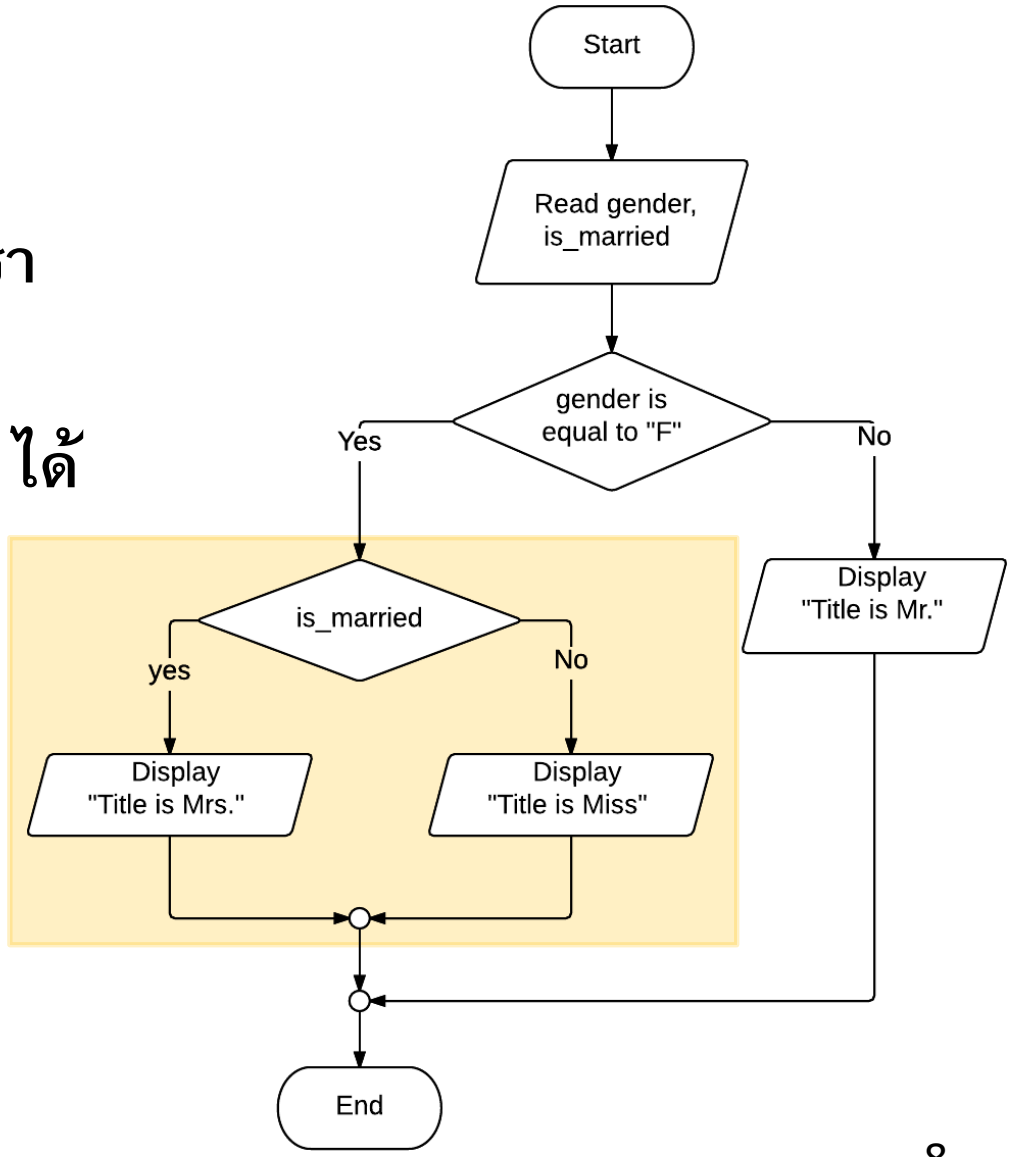
```
if Boolean expression:
    block of code
elif Boolean expression:
    block of code
else:
    block of code
```



Nested Conditionals

- ภายในกิ่งใด ๆ ของ **Conditional Statement** เราสามารถมี **Conditional Statement** ซ้อนอีก **Block** ได้

```
if Boolean expression:
    if Boolean expression:
        block of code
    else:
        block of code
else:
    block of code
```



Nested Conditionals

- ในบางกรณี เราสามารถใช้ **Basic Conditionals, Nested Conditionals** หรือ **Chained Conditionals** เพื่อแก้ปัญหาเดียวกันได้ (เช่นการหา **max-mid-min** ของเลข **3** จำนวน)
- อย่างไรก็ตามการเขียนโปรแกรม **Nested Condition** (`if` ซ้อน `if`) นั้น มีลักษณะโครงสร้างที่อ่านและเข้าใจได้ยากกว่า **Conditional Statement** ลักษณะอื่น
  - ควรหลีกเลี่ยงหากเป็นไปได้

Nested Conditionals [2]

- เราสามารถนำ **Logical Operators** มาประยุกต์ใช้เพื่อหลีกเลี่ยงการใช้ **Nested Condition** ได้เช่น

```
if 0 < x:
    if x < 10:
        print("x is a positive single digit number.")
```

- จะเห็นได้ว่าฟังก์ชัน `print()` จะถูกเรียกใช้ก็ต่อเมื่อเงื่อนไขใน `if` เป็นจริงทั้ง 2 กรณี
- เราสามารถใช้ `and` **Operator** เพื่อสร้าง **Expression** ใหม่

```
if (0 < x) and (x < 10):      # 0 < x < 10
    print("x is a positive single digit number.")
```

The "dangling-else" problem

- **The dangling else is a problem in computer programming in which an optional else clause in an if–then(–else) statement results in nested conditionals being ambiguous.**
- **In C, the statement `if (a) if (b) s; else s2;`**
  - **Can be interpreted as**

```
if (a)
{
    if (b)
        s;
    else
        s2;
}
```

OR

```
if (a)
{
    if (b)
        s;
} else
    s2;
```

The "dangling-else" problem [2]

```
if (x < 10):
    if (x > 3):
        print("a")
    else:
        print("b")
```

```
if (x < 10):
    if (x > 3):
        print("a")
else:
    print("b")
```

Logical Opposites

- ใน **Relational Operator** แต่ละตัว จะมีเครื่องหมายตรงกันข้ามอยู่ ตัวอย่างเช่น
- ในประเทศไทย ผู้ที่ต้องการทำใบขับขี่ จะต้องมีอายุมากกว่าหรือเท่ากับ 18 ปี (≥ 18)
  - ไม่สามารถทำใบขับขี่ได้หากมีอายุต่ำกว่า 18 ปี (< 18)
  - สังเกตว่า เครื่องหมายตรงกันข้ามของ ≥ คือ <

operator	logical opposite
==	!=
!=	==
<	>=
<=	>
>	<=
>=	<

Logical Opposites [2]

- พิจารณาชุดคำสั่ง

```
if not (age >= 18):
    print("You're too young to get a driving license!")
```

- สามารถ simplify ได้เป็น

```
if age < 18:                          # อ่านและเข้าใจได้ง่ายกว่า
print("You're too young to get a driving license!")
```

De Morgan's Laws

- เช่นเดียวกันกับใน **Boolean Algebra** เราสามารถนำ **De Morgan's Laws** มาใช้เพื่อ **Simplify** ประโยคเงื่อนไข **(Conditional Statements)** ได้ดังนี้

$$\sim(a \wedge b) = \sim a \vee \sim b$$
$$\sim(a \vee b) = \sim a \wedge \sim b$$

```
not (x and y) == (not x or not y)
 not (x or y) == (not x and not y)
```

De Morgan's Laws [2]

Example:

- วัยเรียน (หรือนักศึกษา) คือคนที่มีอายุในช่วง 6 ปีขึ้นไป จนถึง 21 ปี
  - `6 ≤ age ≤ 21` → `student`
  - `(age >= 6) and (age <= 21)` → `student`
- คนที่มีอายุ น้อยกว่า 6 ปี หรือ มากกว่า 75 ปี ไม่ถือเป็นวัยทำงาน
  - `age < 6` หรือ `age > 75` → not professional
  - `(age < 6) or (age > 75)` → not professional
  - `not ((age < 6) or (age > 75))` → professional

De Morgan's Laws [3]

Example (contd.):

```
if (age >= 6) and (age <= 21):
  print("student")

elif not ((age < 6) or (age > 75)):
  print("professional")
```

- เมื่อใช้ **De Morgan's Law** จะได้

```
if (age >= 6) and (age <= 21):
  print("student")

elif (age >= 6) and (age <= 75):
  print("professional")
```

Refactoring Conditions

```
if (age >= 6) and (age <= 21):
    print("student")

elif (age >= 6) and (age <= 75):
    print("professional")
```

- สามารถใช้กฎการกระจายเพื่อดึง `(age >= 6)` ออกมา

```
if age >= 6:
  if age <= 21:
    print("student")

  elif age <= 75:
    print("professional")
```

Readability?

Note: ตัวอย่างเป็นไปเพื่อการแสดงการใช้สมบัติของ **Boolean Algebra**

Refactoring Conditionals [2]

Love6 Game:

- กำหนด integer 2 ตัว `first` และ `second` โปรแกรมจะแสดงผล **True** ก็ต่อเมื่อ
  - ตัวใดตัวหนึ่งมีค่าเท่ากับ 6
  - ผลบวกของทั้งสองตัวมีค่าเท่ากับ 6
  - ผลต่างของทั้งสองตัวมีค่าเท่ากับ 6

```
if ((first == 6) or (second == 6) or
    (first + second == 6) or
    abs(first - second == 6)):
    print("True")
else:
    print("False")
```

สังเกตการใช้วงเล็บ () กรณี Boolean Expression มีความยาวมากกว่า 1 บรรทัด

Refactoring Conditionals [3]

Love6 Game Revisited (2):

ในกรณีที่เงื่อนไขมีลักษณะเป็น Mutually Exclusive (เหตุการณ์ไม่เกิดร่วม) เราสามารถเปลี่ยน `or` Statement ให้เป็น `elif` ได้

```
if first == 6:
    print("True")
elif second == 6:
    print("True")
elif first + second == 6:
    print("True")
elif abs(first - second == 6):
    print("True")
else:
    print("False")
```

Tips

- ในกรณีโครงสร้างแบบ Chain (`elif`) ที่ Boolean Expression สามารถ evaluate เป็น `True` ได้มากกว่าหนึ่งเงื่อนไข เงื่อนไขแรกที่เป็น `True` เท่านั้นที่จะถูกดำเนินการ

```
x = 360

if x % 2 == 0:      # กรณีนี้เท่านั้นที่ถูกแสดงผลเมื่อ x มีค่าเท่ากับ 360
    print("2 divides x")
elif x % 3 == 0:
    print("3 divides x")
elif x % 5 == 0:
    print("5 divides x")
else:
    print("x is not divisible by 2 3 or 5")
```

Miscellaneous

- ในบางกรณี (เช่น ในกรณีออกแบบ หรือ debug โปรแกรม) เราอาจต้องการให้ชุดคำสั่งในส่วนของ Body ยังไม่ต้องทำอะไร
  - สามารถใช้คำสั่ง `pass` ได้

```
if x < 0:
    pass
```

- **Conditional Expression**

```
01 p = 5
02 print("I saw", p, "people" if (p > 1) else "person")
03 p = 1
04 print("I saw", p, "people" if (p > 1) else "person")
```

Basic Program Instructions

A few basic instructions appear in just about every language:

- Input
- Output
- Math
- Conditional Execution
- **Repetition**

Iteration

- Repeating identical or similar tasks without making errors is something that computers do well and people do poorly.

  การทำงานที่เหมือนกันหรือคล้ายคลึงซ้ำ ๆ อย่างไม่มีข้อผิดพลาดเป็นสิ่งที่ Computer ทำได้ดีกว่ามนุษย์

http://openbookproject.net/thinkcs/python/english3e/iteration.html

Types of Iteration

- **Counter-Controlled Loops**
  - **Loop ที่ทำการวนซ้ำตามจำนวนครั้งที่กำหนด**
- **Condition-Controlled Loops**
  - **Loop ที่ทำการวนซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นเท็จ**

Simple Iteration – for Loop

- เราใช้คำสั่ง for เพื่อระบุการทำซ้ำ ในกรณีที่เรา*ทราบจำนวนครั้ง*แน่นอน เช่น

```
>>> for i in range(5):
        print("Hello")

Hello
Hello
Hello
Hello
Hello
>>>
```

- ตัวเลข 5 ใน function range() คือจำนวนครั้งที่ทำซ้ำ

for Loop – Basic Form

```
for LoopVariable in range(n):
□□□□LoopBody
```

- ***LoopVariable*** เป็นตัวแปรที่ใช้เพื่อการระบุรอบที่ดำเนินการในปัจจุบันว่าเป็นรอบที่เท่าไร โดยมากใช้ตัว **i** (ย่อมาจาก **iterator**)
- ***n*** คือ***จำนวนครั้ง***ที่ต้องทำซ้ำทั้งหมด (หาก $n \leq 0$ **loop** นี้จะถูกข้ามไป
- ***LoopBody*** คือชุดคำสั่งที่ต้องทำซ้ำ มีอย่างน้อย **1** บรรทัด
- เราเรียกการทำซ้ำแบบนี้ว่า **loop** เนื่องจากเมื่อดำเนินการในชุดคำสั่งตาม ***LoopBody*** จนถึงบรรทัดสุดท้ายแล้ว ก็จะวนไปดำเนินการที่ชุดคำสั่งในบรรทัดแรกของ ***LoopBody*** อีกจนกว่าจะครบจำนวนครั้งที่ระบุ (***n***)

for Loop – Basic Form [2]

- พิจารณาค่าของ iterator

```
>>> for i in range(5):
        print("i = {0:d}".format(i))
i = 0
i = 1
i = 2
i = 3
i = 4
```

ไม่ถึง และ ไม่เกิน n

- i มีค่าจาก 0 ถึง n - 1

for Loop – range()

- `range()` เป็นชนิดข้อมูลชนิดหนึ่ง ของภาษา **python**
- หากใส่ **argument** ตัวเดียว จะเป็นการระบุจุดสิ้นสุดของ **range** คืออยู่ในรูป `range(stop)`
  - `range(5)` จะเป็นตัวเลขระหว่าง **0 – 4**
- เราสามารถระบุจุดเริ่มต้นนอกเหนือจาก **0** ได้ในรูป

```
range(start, stop)        # start และ stop เป็น int
```

```
>>> for i in range(18, 21):
        print("i = {0:d}".format(i))
i = 18
i = 19
i = 20
```

เนื่องจากเป็นการเพิ่มค่าทีละ 1

stop ต้องมากกว่า start

for Loop – range() [2]

- โดยปกติแล้ว `range()` จะเริ่มที่ค่า **start** แล้วเพิ่มค่าทีละ **1** ในแต่ละขั้น **(step)**
- เราสามารถระบุความกว้างของขั้นได้ในรูปของ

```
range(start, stop, step)        # all integers
```

```
>>> for i in range(8, 20, 3):
        print("i = {0:d}".format(i))
i = 8
i = 11
i = 14
i = 17
```

for Loop – range() [3]

- เราสามารถระบุ **step** ให้มีค่าเป็นลบ เพื่อให้ **range** มีลักษณะเรียงจาก*มากไปน้อย*ได้

```
>>> for i in range(8, 2, -2):
        print("i = {0:d}".format(i))
i = 8
i = 6
i = 4
```

- ผลลัพธ์ของ **loop** ด้านล่างคืออะไร

```
>>> for i in range(2, 8, -2):
        print("i = {0:d}".format(i))
```

Loop Variable

ข้อควรระวัง: ไม่ควร reassign ค่า หรือเปลี่ยนค่าของ loop variable (iterator)

```
>>> for i in range(5):
        print(i, end=" ")
        i = 3
0 1 2 3 4

>>> for i in range(5):
        i = 3
        print(i, end=" ")
3 3 3 3 3
```

- หากมีการ reassign ค่าของ loop variable ค่าของ variable นั้น ๆ จะถูก reset ให้เป็นค่าที่ถูกก่อนวน loop ครั้งถัดไป

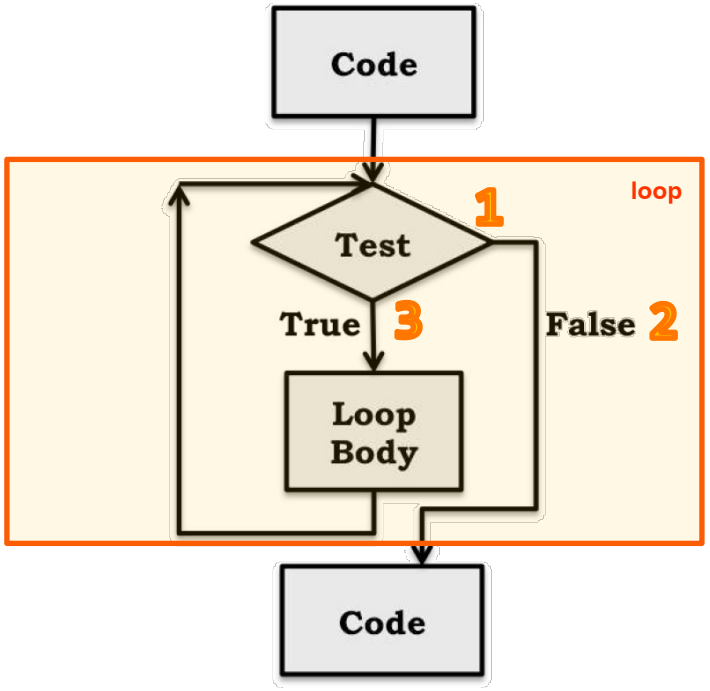
Types of Iteration

- Counter-Controlled Loops
  - Loop ที่ทำการวนซ้ำตามจำนวนครั้งที่กำหนด
- **Condition-Controlled Loops**
  - **Loop ที่ทำการวนซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นเท็จ**

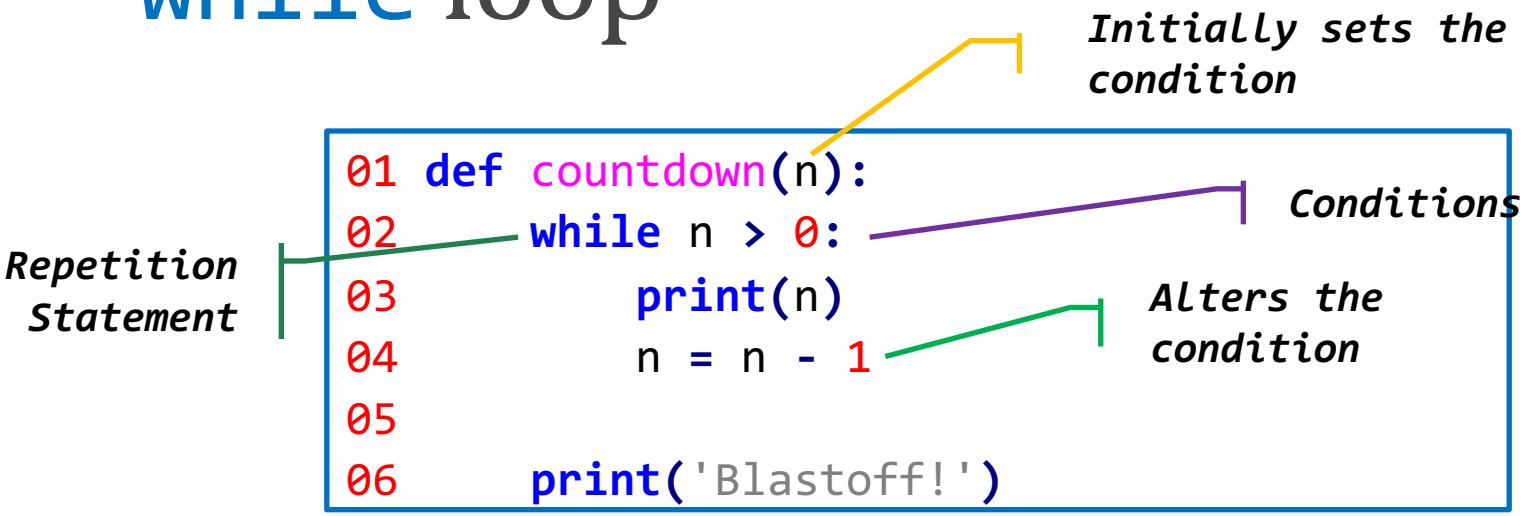
Generic Loop Structure

- **Loop structure ทั้งสองชนิดมีลักษณะคล้ายคลึงกันดังนี้**

1. **Test – ได้ผลลัพธ์เป็น True หรือ False**
2. **ถ้าผลลัพธ์เป็น False - ออกจาก loop**
3. **ถ้าผลลัพธ์เป็น True. ดำเนินการชุดคำสั่งใน Loop Body หนึ่งครั้ง แล้วกลับไปที่ข้อ 1**



while loop



- การทำงานของฟังก์ชันสามารถพิจารณาได้เหมือนการตีความประโยคภาษาอังกฤษปกติ
  - "ในขณะที่ n (ยัง) มากกว่า 0 แสดงค่า n แล้วลดค่า n ลง 1"
- ชุดคำสั่งในส่วน Loop Body ควรมีการเปลี่ยนแปลงค่าตัวแปรเพื่อที่จะส่งผลให้ Boolean Expression มีค่าเป็น False ในที่สุด
  - เพื่อที่ loop จะได้หยุดการทำงาน

```
while Boolean_expression:
□□□□loopBody
```

`while` loop [2]

- ในการออกแบบ **algorithm Loop** ทุก **loop** จะต้องหยุดการทำงาน **(terminate)** ในที่สุด
- **Loop** ที่ทำงานไปเรื่อยโดยไม่หยุดเรียกว่า **infinite loop**
- **Classic Example** ของ **infinite loop**
  - วิธีใช้ **shampoo**
    - **Lather, Rise, Repeat**

while loop [3]

ในกรณีใดบ้างที่โปรแกรม จะ terminate?

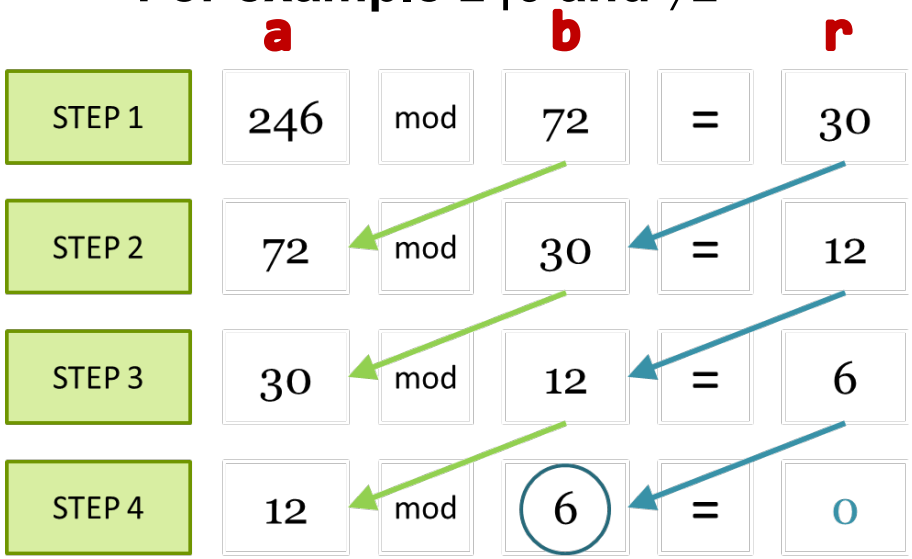
```
06 x = 3
07 ans = 0
08 itersLeft = x
09
10 while (itersLeft != 0):        # Square an integer, the hard way
11     ans = ans + x
12     itersLeft = itersLeft - 1
13
14 print(str(x) + '*' + str(x) + ' = ' + str(ans))
```

- **เราสามารถจำลองการ run ของ loop ได้โดยการเขียน**

test#	x	ans	itersLeft
1	3	0	3
2	3	3	2
3	_____	_____	_____
4	_____	_____	_____

Example 1: The Euclidean Algorithm

- **For example** 246 **and** 72




```
01 def gcd(a, b):
02     r = _____
03     while _____:
04         _____
05         _____
06         _____
07     return _____
```

- เราจะเปลี่ยน **algorithm** นี้เป็น **loop** ได้อย่างไร?
  - ตั้งชื่อให้แต่ละ **column** (นี่คือชื่อ **variable**)
  - **Loop terminate** เมื่อไร _____
  - ผลลัพธ์ที่ต้องการอยู่ใน **variable** ชื่ออะไร _____

Example 2: Number Guessing

- **Problem Statement:**
  - **ต้องการเขียนโปรแกรมเพื่อให้ user เล่นเกมทายเลขระหว่าง 1 – 20 โดยจะทายได้ทั้งหมด 5 ครั้ง หายเลขที่ทายต่ำไป หรือสูงไปจะมีคำใบ้บอก**

```
$ python number_guess.py
Input number: 3
3 is too low
Input number: 7
7 is too high
Input number: 6
5 is correct!
```

Example 2: Number Guessing [2]

- เนื่องจากจำนวนครั้งที่วน loop มีค่าคงที่คือไม่เกิน 5
  - พิจารณาใช้ for loop

```
05 def guess_num():
06     key = 6
07     for i in range(5):
08         n = int(input("Input number: "))
09         if n == key:
10             print(n, "is correct")
11             ____________________
12         elif n > key:
13             print(n, "is too high")
14         else:
15             print(n, "is too low")
```

- กรณีทายถูก (บรรทัดที่ 09) โปรแกรมจะต้องออกจาก loop
  - ใช้คำสั่ง **break**

The `break` Statement

- เราสามารถใช้คำสั่ง `break` เพื่อออกจาก loop (ชั้นปัจจุบัน) ได้ตามเงื่อนไขที่ระบุ โดยคำสั่งอื่น ๆ ภายใน loop หลังจาก `break` จะถูกข้ามไป
- ใช้ได้กับคำสั่ง `for`, `while`

Example 3: Score Average

- **Problem Statement: ต้องการเขียนฟังก์ชันเพื่อรับคะแนน ระหว่าง 0 – 100 ของ นักเรียน 30 คนเพื่อหาค่าเฉลี่ย โดยไม่พิจารณาคะแนนในช่วงที่ไม่ถูกต้อง (น้อยกว่า 0 หรือ มากกว่า 100)**

```
09 def score_average():
10     total_count = 30
11     score_count = 0
12     total = 0
13
14     while score_count < total_count:
15         score = float(input("Enter score: "))
16
17         if score < 0 or score > 100:
18             ____________________
19
20         total = total + score
21         score_count = score_count + 1       # += 1
22
23     return total / score_count
```

ทำไมในกรณีนี้ จึงไม่ควรใช้ for loop?

The `continue` Statement

- คำสั่ง `continue` ใช้ได้กับ loop เท่านั้นโดยจะข้ามคำสั่งที่เหลือภายใน loop หลังจากคำสั่ง `continue` เพื่อไปวน loop ในรอบถัดไป
- ใช้ได้กับคำสั่ง `for`, `while`

Basic Loop Structures

การสร้าง Loop ประกอบด้วย 4 องค์ประกอบหลักคือ

1. **Repetition statement: คำสั่งที่ใช้สำหรับการทำซ้ำ**
   - ☑ `while statement`
   - ☑ `for statement`
2. **Condition: เงื่อนไขของการทำซ้ำ**
3. **A statement that initially sets the condition being tested: การตั้งค่าเริ่มต้นให้กับตัวแปรที่ใช้ในการควบคุมเงื่อนไข**
4. **A statement within the repeating section of code that alters the condition so that it eventually becomes false: การเปลี่ยนแปลงค่าตัวแปรที่ใช้ในการควบคุมเงื่อนไข ในการวนแต่ละครั้ง เพื่อให้เงื่อนไขเป็นเท็จในที่สุด**

Sentinels

- การระบุจำนวนครั้งที่ทำซ้ำไว้เป็นค่าคงที่ ในบางครั้งอาจไม่ยืดหยุ่นเพียงพอกับปัญหาที่ต้องการแก้
  - เช่นกรณีต้องการหาค่าเฉลี่ยของคะแนนของนักเรียนในชั้น แต่ไม่ทราบจำนวนนักเรียนล่วงหน้า
- สามารถแก้ปัญหาได้โดย
  - ให้ user ระบุจำนวนครั้งที่ต้องการทำซ้ำ ผ่าน input
  - หรือ อ่าน input จาก user จนเจอค่าที่ตกลงกันไว้ว่าใช้แสดงจุดสิ้นสุดของข้อมูล เช่น -1 หรือ EOF (End of File)
    - ค่าที่ใช้แสดงจุดเริ่มหรือสิ้นสุดของข้อมูลในลักษณะนี้เรียกว่า ***sentinels***
    - ควรเลือกค่า sentinels ให้ไม่ทับซ้อนกับค่าของข้อมูลที่เป็นไปได้

Sentinels [2]

```
09 def score_average():
10     SENTINEL = -1
11     print("Please enter students' score one for each line")
12     print("and %d for termination: " % SENTINEL)
13     total = 0.0
14     count = 0
15
16     while (True):
17         score = float(input(""))
18
19         if score == SENTINEL:
20             break
21         total += score
22         count += 1
23
24     if count > 0:        #why do we need this?
25         average = total / count
26     else
27         average = 0
28
29     print("The average of the %d numbers is %8.4f" %(count, average))
```

References

- **Guttag, John V. *Introduction to Computation and Programming Using Python, Revised***
- **http://en.wikibooks.org/wiki/Python_Programming/Conditional_Statements**
- **https://docs.python.org/3/tutorial/controlflow.html**
- **http://www.kosbie.net/cmu/spring-14/15-112/**
- **https://docs.python.org/3/tutorial/controlflow.html**
- **http://www.greenteapress.com/thinkpython/thinkCSpy/html/chap04.html**