

w03-Lab

# Conditionals

## Part I

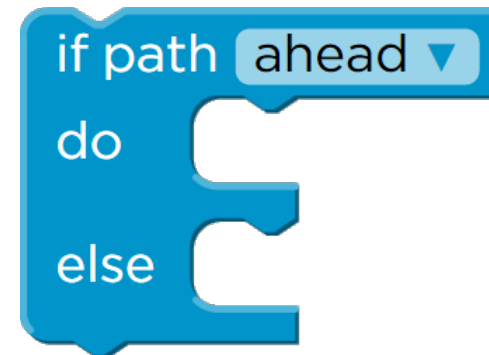
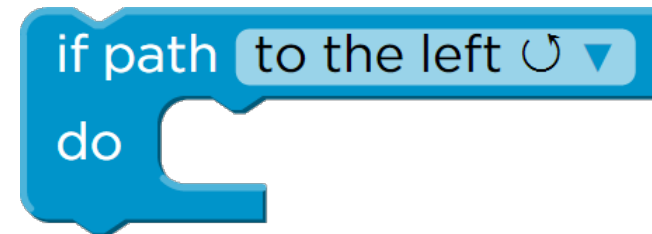
for 204111

Kittipitch Kuptavanich

# Basic Program Instructions

A few **basic instructions** appear in just about every language:

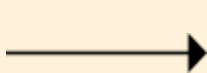



- Input
- Output
- Math
- **Conditional Execution**
- Repetition

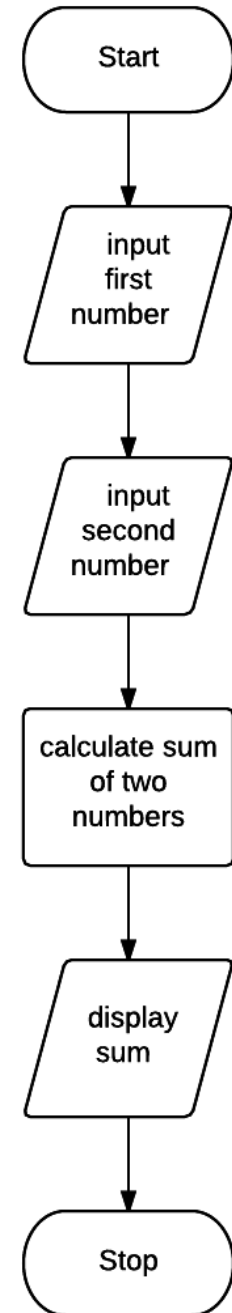


# Branching Programs

- โปรแกรมที่เราได้พัฒนาขึ้นมาก่อนหน้านี้มีลักษณะเป็น **Straight-line Programs**
  - ดำเนินการเป็นเส้นตรงจากบนลงล่าง
  - เช่นการหาผลบวกของตัวเลข 2 จำนวน
    - นอกจาก **Pseudocode** แล้วเราสามารถใช้ **Flowchart** ในการวางโครงร่างโปรแกรม

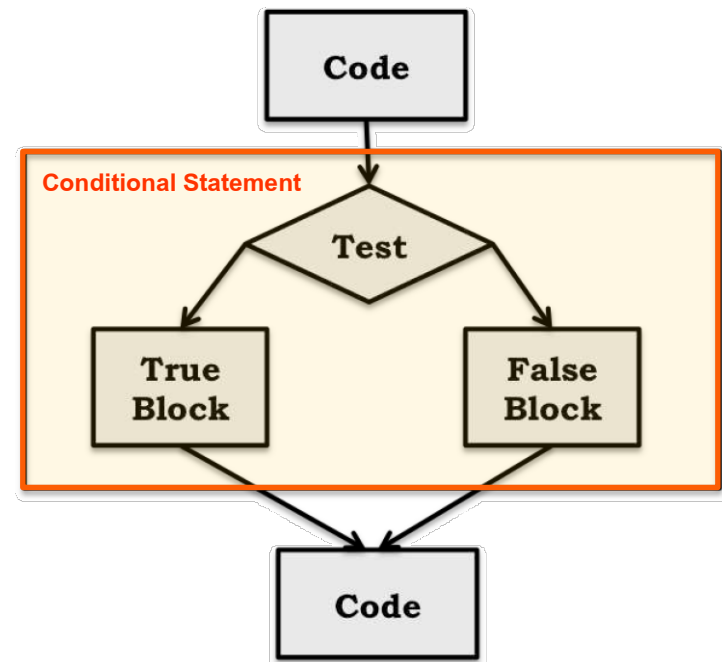
<https://en.wikipedia.org/wiki/Flowchart>

| Shape |  |  |  |  |
|-------|---|---|--|---|
| Name  | Line  | Input/Output  | Process  | Terminal  |
| Usage |   |   |  |   |



# Branching Programs [2]

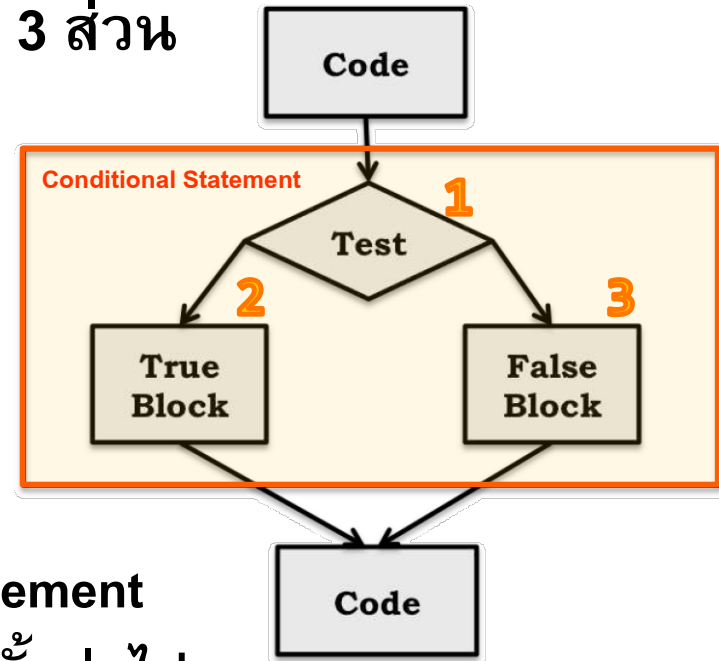
- ในการพัฒนาโปรแกรม หลาย ๆ ครั้งที่เราจำเป็นต้องมีการพิจารณาเงื่อนไขต่าง ๆ และออกแบบโปรแกรมให้ทำงานต่างกันไปตามเงื่อนไขนั้น ๆ
- เมื่อโปรแกรมทำงานมาถึงชุดคำสั่งที่มีการตรวจสอบเงื่อนไข (Test Expression) ก็จะทำการแตกการทำงาน (Branching) ออกเป็น 2 กิ่ง (หรือ 2 สาย)



# Branching Programs [3]

- 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 ก็จะทำเนิการในชุดคำสั่งหลังจากนั้นต่อไป
- ในภาษา Python Conditional Statement จะอยู่ในรูป

```

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

# Boolean Expressions

- **Boolean Expression** คือ **Expression** ที่มีค่าเป็น **True** (จริง) หรือ **False** (เท็จ)

```
>>> 5 == 5
True
>>> 5 == 6
False
```

- ค่า **True** หรือ **False** เป็นค่าเฉพาะที่มาจากชนิดข้อมูล **bool** (และไม่ใช่ **string**)

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

# Boolean Expressions [2]

- เครื่องหมาย `==` เป็นหนึ่งใน **Operator** ทางความสัมพันธ์ (Relational Operator)
- Relational Operator อื่น ๆ ได้แก่

```
x != y      # x is not equal to y
x > y       # x is greater than y
x < y       # x is less than y
x >= y      # x is greater than or equal to y
x <= y      # x is less than or equal to y
```

- หากต้องการเขียน **Expression** ข้ามบรรทัด สามารถทำได้โดยการ  
ใช้เครื่องหมาย **Backslash \** หรือวงเล็บ **()**

```
>>> x = 8
>>> (x + 4 < 10 and
      x % 2 != 1)
False
```

```
>>> x = 8
>>> x + 4 < 10 and
      x % 2 != 1
False
```

บรรทัดต่อไป  
จะถูกเก็บไว้จนกว่า  
ตัวถัดมา

# Boolean Expressions [3]

- Floating-point Number Comparisons

```
>>> print(0.1 + 0.1 == 0.2)
True                                # True, but...

>>> print(0.1 + 0.1 + 0.1 == 0.3)
False

>>> print(0.1 + 0.1 + 0.1)
0.3                                # seems ok

>>> print((0.1 + 0.1 + 0.1) - 0.3)
5.55111512313e-17                 # (tiny, but non-zero!)
```

- ค่าที่เก็บในตัวแปรชนิด **float** เป็นค่าประมาณ!!



# Floating-Point Numbers and `almost_equal()`

```
09 d1 = 0.1 + 0.1 + 0.1
10 d2 = 0.3
11 print(d1 == d2)                # still False, of course
12 epsilon = 10 ** -10
13 print(abs(d2 - d1) < epsilon)  # True!
14
15 # Once again, using an almostEqual function
16 # (that we will write)
17
18 def almost_equal(d1, d2, epsilon=10 ** -10):
19     return (abs(d2 - d1) < epsilon)
20
21 d1 = 0.1 + 0.1 + 0.1
22 d2 = 0.3
23 print(d1 == d2)                # still False, of course
24 # True, and now packaged in a handy reusable function!
25 print(almost_equal(d1, d2))
```

# math.isclose()

- Python 3.5 and up has a built-in function to do this also

```
05 from math import isclose
06
07 d1 = 0.1 + 0.1 + 0.1
08 d2 = 0.3
09
10 epsilon = 10 ** -10
11 print(abs(d2 - d1) < epsilon)           # True!
12
13 print(isclose(d1, d2, abs_tol=epsilon))  # True!
14
15 print(isclose(8.005, 8.450, abs_tol=0.4)) # False!
16 print(isclose(8.005, 8.450, abs_tol=0.5)) # True!
```

# `math.isclose()` [2]

- `math.isclose()` has the following signature

`math.isclose(a, b, rel_tol=1e-9, abs_tol=0.0)`

- `rel_tol` = relative tolerance
  - relative to the largest of a or b
  - Default to  $10^{-9}$
  - Relative tolerance of 5%: `rel_tol=0.05`
- `abs_tol` = absolute tolerance
  - Minimum absolute tolerance
  - useful for comparisons near zero
- Implementation:
  - `abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)`

# Logical Operator

- ในภาษา Python มี ตัวดำเนินการทางตรรกะ (Logical Operator) 3 ตัวได้แก่ **and**, **or** และ **not**

- ความหมายของ Operator ทั้งสามตัวตรงกับความหมายในภาษาอังกฤษ

เราสามารถใช่วงเล็บเพื่อช่วยทำให้เงื่อนไขอ่านง่ายขึ้น

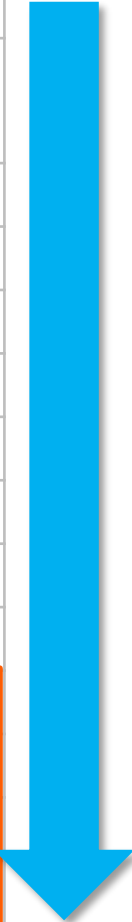
- $(x > 0) \text{ and } (x < 10)$
- $(n \% 2 == 0) \text{ or } (n \% 3 == 0)$

- ตัวอย่างเช่น

- $x > 0 \text{ and } x < 10$  จะเป็นจริงก็ต่อเมื่อ  $x$  มากกว่า 0 และ น้อยกว่า 10 (ในกรณีนี้สามารถเขียน  $0 < x < 10$  ได้ ในภาษา Python)
- $n \% 2 == 0 \text{ or } n \% 3 == 0$  จะเป็นจริงเมื่อ เงื่อนไขตัวใดตัวหนึ่ง (หรือทั้ง 2 ตัว) เป็นจริง
- โดยทั่วไปแล้ว Operands ของ Logical Operator ควรเป็น Boolean Expression แต่ในภาษา Python Logical Value ของตัวเลขใด ๆ ที่มีค่าไม่เป็น 0 จะมีค่าเป็น True

```
>>> 17 and True
True
```

# Operator Precedence [2]

| Operator  | Description   |   |
|---|---|---|
| (expressions...), [expressions...],<br>{key: value...},{expressions...} | Binding or tuple display, list display, dictionary display, set display | <div>high</div>  <div>low</div> |
| x[index], x[index:index],<br>x(arguments...), x.attribute               | Subscription, slicing, call, attribute reference                        |   |
| **  | Exponentiation  |   |
| +x, -x, ~x  | Positive, negative, bitwise NOT   |   |
| *, /, //, %   | Multiplication, division, remainder                                     |   |
| +, -  | Addition and subtraction  |   |
| <<, >>  | Shifts  |   |
| &   | Bitwise AND   |   |
| ^   | Bitwise XOR   |   |
|   | Bitwise OR  |   |
| in, not in, is, is not, <, <=, >, >=, !=, ==                            | Comparisons, including membership tests and identity tests              |   |
| not x   | Boolean NOT   |   |
| and   | Boolean AND   |   |
| or  | Boolean OR  |   |
| if – else   | Conditional expression  |   |
| lambda  | Lambda expression   | 13  |

# Short-Circuit Evaluation

- หากพิจารณา Truth Table ของ Expression **p or q**
- จะพบว่า กรณีเดียวที่ Expression จะมีค่าเป็น **False** คือกรณีที่ **p** และ **q** เป็น **False** ทั้งคู่
  - ดังนั้นหากพบว่า Operand ตัวแรก (**p**) มีค่าเป็น **True** เราสรุปได้ว่า Expression นี้จะ Evaluate เป็น **True** โดยไม่จำเป็นต้องพิจารณาค่าของ **q**
  - กรณี **p** มีค่าเป็น **False** เป็นกรณีเดียวที่ต้องพิจารณาค่า **q**
- การ Evaluate ค่าโดยพิจารณา Operand บางส่วนเท่าที่จำเป็นแล้วให้ผลลัพธ์ทันที เรียกว่า **Short Circuit Evaluation**
- ภาษา Programming หลาย ๆ ภาษา ใช้การ Evaluate ในลักษณะนี้
- เช่นเดียวกันกับการพิจารณา Expression **p and q**
  - ทำ Short Circuit Evaluation ได้ทันทีเมื่อพบว่า **p** มีค่าเป็น \_\_\_\_\_ และจะ Evaluate Expression เป็น \_\_\_\_\_

| <i>p</i> | <i>q</i> | <i>p</i> ∨ <i>q</i> |
|----------|----------|---------------------|
| T        | T        | T                   |
| T        | F        | T                   |
| F        | T        | T                   |
| F        | F        | F                   |

| <i>p</i> | <i>q</i> | <i>p</i> ∧ <i>q</i> |
|----------|----------|---------------------|
| T        | T        | T                   |
| T        | F        | F                   |
| F        | T        | F                   |
| F        | F        | F                   |

# Short-Circuit Evaluation [2]

```
>>> x = 1
>>> y = 0
>>> print((y != 0) and ((x / y) != 0))  # Works!
False
>>> print(((x / y) != 0) and (y != 0))  # Crashes!
...
ZeroDivisionError: division by zero

>>> print((y == 0) or ((x / y) == 0))  # Works!
True
>>> print(((x / y) == 0) or (y == 0))  # Crashes!
...
ZeroDivisionError: division by zero
```

# Short-Circuit Evaluation [3]

```
25 def isPositive(n):
26     result = (n > 0)
27     print("isPositive(", n, ") =", result)
28     return result
29
30
31 def isEven(n):
32     result = (n % 2 == 0)
33     print("isEven(", n, ") =", result)
34     return result
35
36 print("Test 1: isEven(-4) and isPositive(-4)")
37 print(isEven(-4) and isPositive(-4)) # Calls both
38 print("-----")
39 print("Test 2: isEven(-3) and isPositive(-3)")
40 print(isEven(-3) and isPositive(-3)) # Calls only one
```



# Truth Value Testing

- **Variable หรือ Literals ทุกตัวมี Truth Value ทั้งหมด**  
(สามารถ Evaluate เป็น **True** หรือ **False** ได้)
- **Value ดังต่อไปนี้ Evaluate เป็น False (ที่เหลือเป็น True)**
  - **None**
  - **False**
  - **zero of any numeric type, for example, 0, 0.0, 0j.**
  - **any empty sequence, for example, '', (), [].**
  - **any empty mapping, for example, {}.**
  - **instances of user-defined classes, if the class defines a `__bool__()` or `__len__()` method, when that method returns the integer zero or bool value False.**

# Truth Value Testing [2]

- Operation และฟังก์ชัน Built-in ไต ๆ ที่มีการคืนค่าเป็น Boolean จะคืนค่า
  - 1 หรือ **True** ถ้าเป็นจริง และ
  - 0 หรือ **False** ถ้าเป็นเท็จ เสมอ
- ข้อยกเว้น Operator **and** และ **or** จะคืนค่าเป็น Operand ตัวใดตัวหนึ่ง (พิจารณาแบบ Short Circuit Evaluation)

```

>>> 23 and 35
35
>>> 0 and -23
0
>>> True and 5
5

```

Handwritten notes for the first box:

- For `23 and 35`: "เลือกเอาตัวสุดท้าย" (Choose the last one).
- For `0 and -23`: "False หรือ" (False or).
- For `True and 5`: "เลือกเอาตัวตัวแรก" (Choose the first one).

```

>>> False and 5
False
>>> (1 and 2) or 42
2
>>> (1 and 0) or 42
42

```

Handwritten notes for the second box:

- For `False and 5`: "ถ้าเป็น 0 หรือเลือกเอาตัวแรก" (If it's 0 or choose the first one).
- For `(1 and 2) or 42`: "เลือกเอาตัวแรก" (Choose the first one).
- For `(1 and 0) or 42`: "เลือกเอาตัวแรก" (Choose the first one).

# Boolean Arithmetic

```
02 # In numeric expressions...
03 #     True is treated as 1
04 #     False is treated as 0
05
06 # So...
07 print(5 * True) # 5
08 print(5 * False) # 0
09 print(5 + True) # 6
10 print(5 + False) # 5
```

- ไม่ควรใช้ Boolean Arithmetic เนื่องจากทำให้ Code อ่านยาก แต่หากจำเป็นควรมีการ Cast ชนิด ด้วยฟังก์ชัน `int()` ก่อน

```
07 print(5 * int(True)) # 5
08 print(5 * int(False)) # 0
09 print(5 + int(True)) # 6
10 print(5 + int(False)) # 5
```

# 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 :**) และมี **ส่วน Body** ที่ต้องย่อหน้า
  - เราเรียก **Statement** ในลักษณะนี้ว่า **Compound Statement**
  - ในบางกรณี (เช่น ในกรณีออกแบบ หรือ debug โปรแกรม) เราอาจต้องการให้ชุดคำสั่งในส่วนของ **Body** ยังไม่ต้องทำอะไร
    - สามารถใช้คำสั่ง **pass** ได้

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

# Conditional Execution [2]

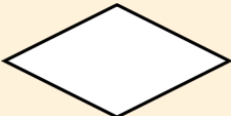

## Example 1

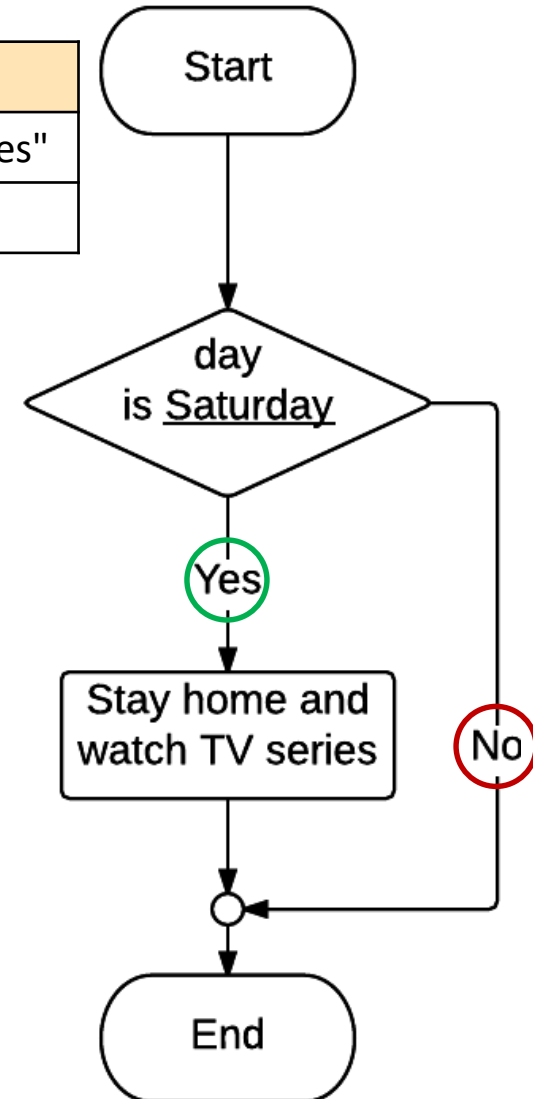
| today is Sat. | Statement                       |
|---------------|---------------------------------|
| YES           | "Stay home and watch TV series" |
| NO            |                                 |

## Statement

- ถ้าวันนี้เป็นวันเสาร์
- อยู่บ้านดู Series

<https://en.wikipedia.org/wiki/Flowchart>

| Shape |  |  |
|-------|--|--|
| Name  | Decision   | Connector  |
| Usage |  |  |



# Conditional Execution [3]

- เราสามารถมีการตัดสินใจเงื่อนไขต่อกันเป็นลำดับได้ เช่น

## Pseudocode

เปิดตู้เย็น

if นมหมด then

    เพิ่มนมในรายการจ่ายตลาด

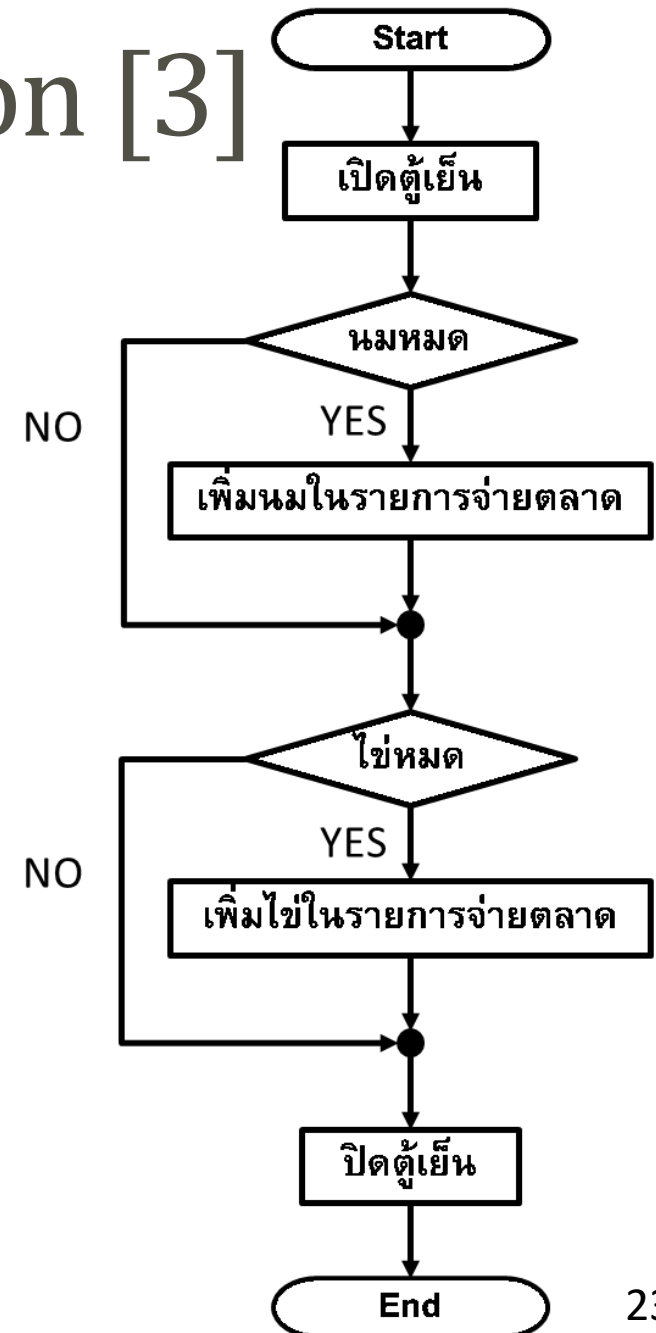
endif

if ไข่หมด then

    เพิ่มไข่ในรายการจ่ายตลาด

endif

ปิดตู้เย็น

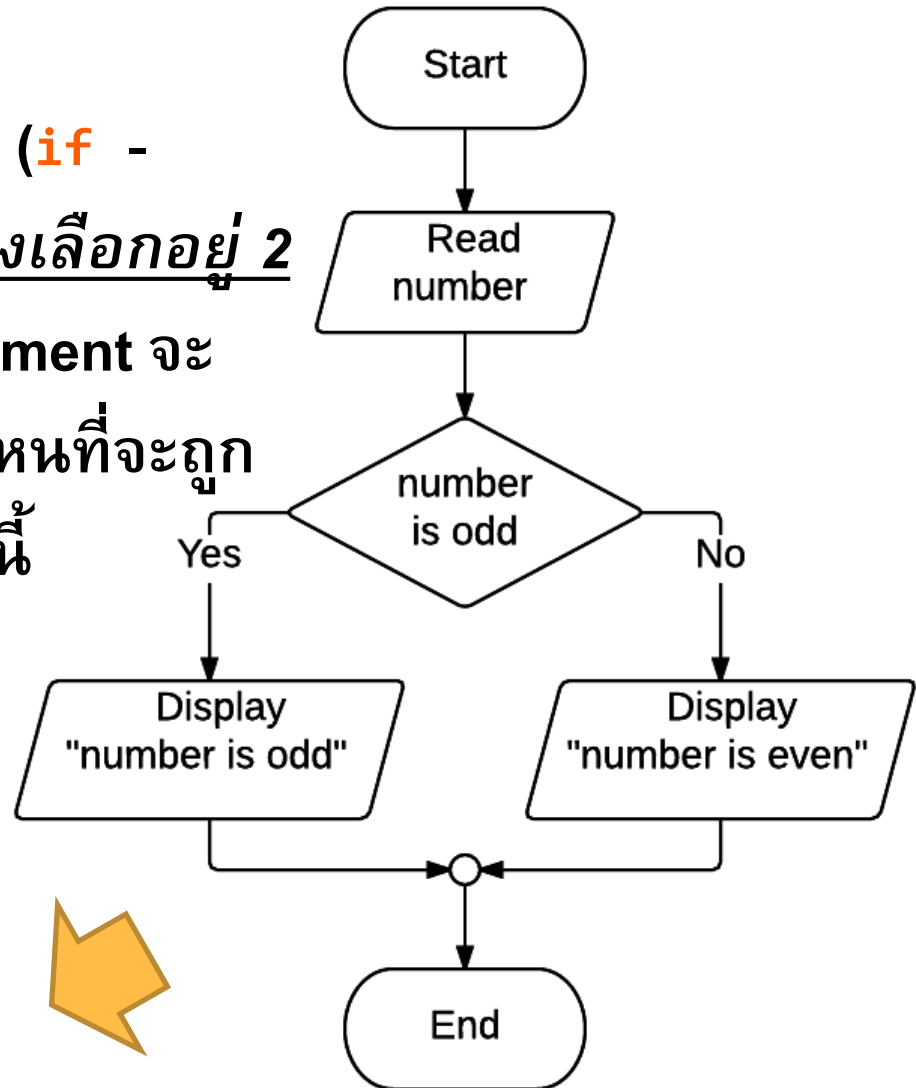


# Alternative Execution

- **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:
    _____
```

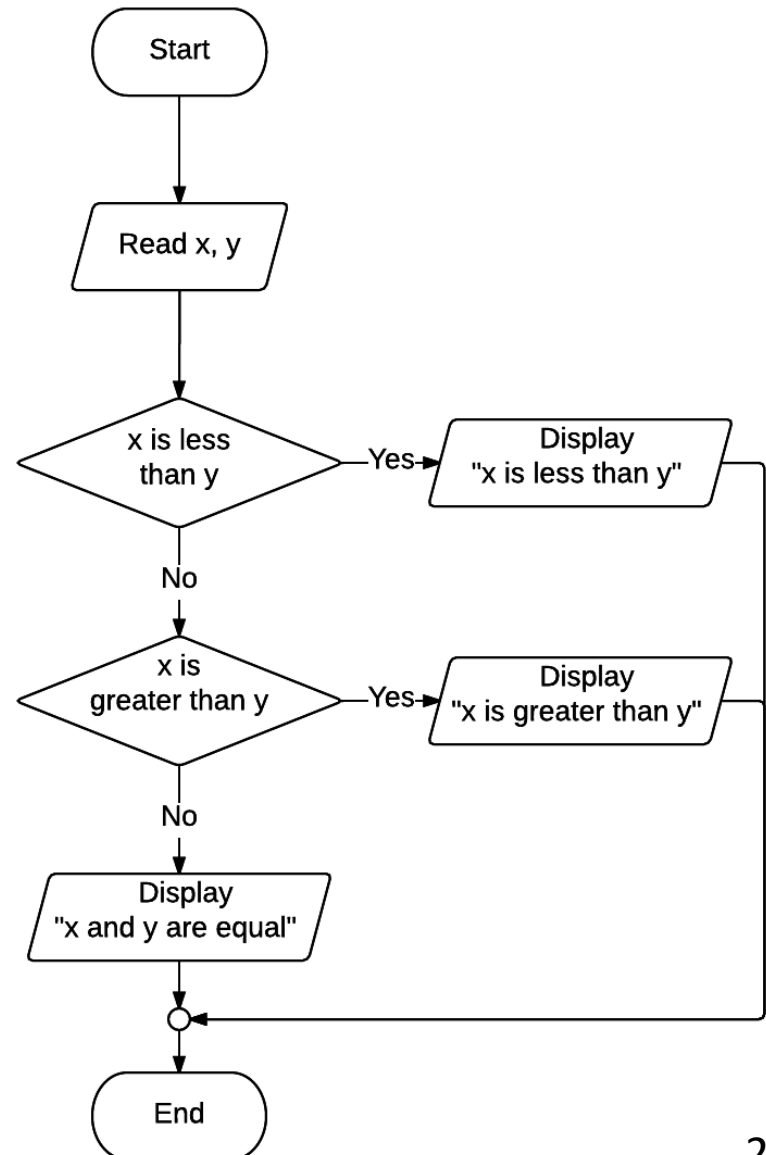


# Chained Conditionals

- ในบางกรณี ทางเลือกที่เป็นไปได้ อาจมีมากกว่า 2 ทาง เราสามารถใช้ **chained condition** (**if** - **elif** - **else**) เพื่อรองรับเงื่อนไขการตัดสินใจในลักษณะนี้
- **elif** คือตัวย่อของ "else if"
- จากตัวเลือกที่เป็นไปได้ ทั้งหมด ชุดคำสั่งเพียง 1 ชุดเท่านั้น ที่จะถูกดำเนินการ

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

คือใช่ if  
 แสดงเป็น true  
 ที่ข้อว่า if  
 เรียงว่าเงื่อนไขเลือกได้



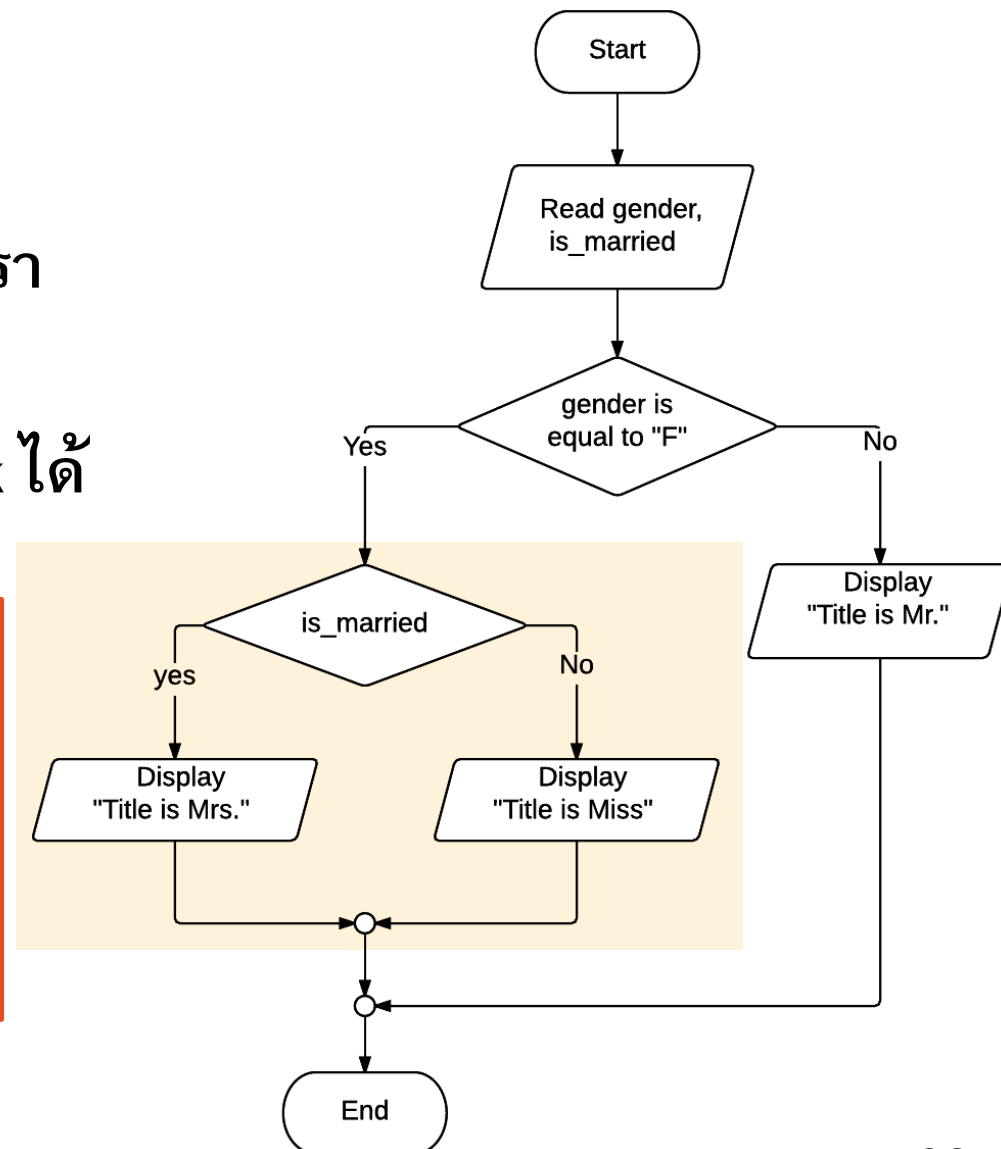


# Nested Conditionals

- 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
  
```



# The `return` Statement

- ในฟังก์ชันที่มีการคืนค่า
  - `return` Statement มีหน้าที่ระบุให้ฟังก์ชันคืนค่าทันที ด้วย ค่าของ Expression ที่ตามหลัง `return`
- พิจารณาการเขียนฟังก์ชันเพื่อคำนวณพื้นที่วงกลม

```
def area_v1(radius):
    temp = math.pi * radius**2
    return temp
```

VS

```
def area_v2(radius):
    return math.pi * radius**2
```

- ตัวแปร `temp` ในฟังก์ชันทางซ้าย ช่วยให้เรา `debug` ได้ง่ายขึ้น (เช่น ใช้ฟังก์ชัน `print()` แสดงค่าที่คำนวณได้ก่อนที่จะ `return`)

# The `return` Statement [2]

- ในบางกรณี เราอาจมี `return` Statement มากกว่าหนึ่ง
  - เช่นในแต่ละกิ่ง (Branch) ของ Conditional

```
17 def absolute_value(x):  
18     if x < 0:  
19         return -x  
20     else:  
21         return x
```

- สังเกตว่า `return` Statement อยู่ภายใต้กิ่งที่แยกกันของ Conditionals ในกรณีนี้ จะมี Statement เดียวเท่านั้นที่ถูกดำเนินการ

# The `return` Statement [3]

- เมื่อฟังก์ชันทำงานมาถึง **Return Statement** ฟังก์ชันจะหยุดดำเนินการ โดยไม่พิจารณา **Statement** ใด ๆ หลังจากบรรทัดนั้น
- เราเรียก **Code** หรือ **Statement** ใด ๆ ในฟังก์ชันหลังจากบรรทัดที่มี **return Statement** หรือในที่อื่น ๆ ที่จะไม่ถูกดำเนินการในกรณีใด ๆ ว่า **Dead Code**

```
04 def main():  
05     print("line 5")  
06     print("line 6")  
07  
08     return  
09  
10     print("line 10")  
11     print("line 11")  
12
```

Dead  
Code

# The `return` Statement [4]

- ในฟังก์ชันที่มีการคืนค่าผลลัพธ์ (Fruitful Function) ควรมีการตรวจสอบให้แน่ใจว่า ทุกกิ่ง (Branch) หรือ Path ภายในฟังก์ชัน จบที่ `return` Statement
  - พิจารณาฟังก์ชัน

```
13 def absolute_value(x):  
14     if x < 0:  
15         return -x  
16     if x > 0:  
17         return x
```

- ฟังก์ชันด้านบน ทำงานไม่ถูกต้อง เนื่องจากหาก  $x$  มีค่าเป็น 0 ฟังก์ชันจะจบการทำงานโดยไม่ผ่าน `return` Statement ทั้ง 2 จุด
  - ดังนั้นเมื่อทำงานจบฟังก์ชันจะคืนค่า `None` (`None` เป็น ค่าที่ถูก `return` โดย Default ของทุกฟังก์ชัน) ทั้งที่คำตอบควรเป็น 0

# Practice 1: 12 Hour Time Format

- ให้เขียนฟังก์ชัน `twelve_hr_time(hour, min)` ที่รับข้อมูลเลขจำนวนเต็ม *hour* ( $0 \leq hour \leq 23$ ) และ *min* ( $0 \leq min \leq 59$ ) แล้วแสดงผลเวลาในรูปแบบ 12 ชั่วโมง โดยให้เขียน `function` ทดสอบเอง ภายในเงื่อนไข `if __name__ == '__main__':`

| <u>Input</u> | <u>Output</u> |
|--------------|---------------|
| 8<br>30      | 8:30 am       |
| 20<br>30     | 8:30 pm       |

# The `datetime` Module

- เราสามารถใช้ Module `datetime` เพื่ออ่านข้อมูลวันที่และเวลาปัจจุบันได้

```
>>> from datetime import date, datetime
>>> date.today().day
1
>>> date.today().month
9
>>> date.today().year
2022
>>> datetime.now().hour
18
>>> # Try also .minute .second .microsecond (1/1,000,000)
```

# Practice 2: 12 Hour Time Format

- ให้เขียนฟังก์ชัน `twelve_hr_time()` ที่ดึงข้อมูลเวลาจากระบบ (ไม่รับ input ผ่านทาง parameter) แล้ว แสดงผลเวลา ในรูปแบบ 12 ชั่วโมง โดยให้เขียน function ทดสอบเอง ภายในเงื่อนไข `if __name__ == '__main__':`

| <u>Input</u> | <u>Output</u> |
|--------------|---------------|
| 8<br>30      | 8:30 am       |
| 20<br>30     | 8:30 pm       |



# Practice 3: Love6 Game

## Love6 Game:

- ให้เขียนฟังก์ชัน `love6(first, second)` โดย `first` และ `second` เป็นจำนวนเต็มทั้งคู่
  - ฟังก์ชันจะ คืนค่า `True` ก็ต่อเมื่อ
    - ตัวใดตัวหนึ่งมีค่าเท่ากับ 6
    - ผลบวกของทั้งสองตัวมีค่าเท่ากับ 6
    - ผลต่างของทั้งสองตัวมีค่าเท่ากับ 6
  - นอกจากนั้นจะคืนค่าเป็น `False`

# Incorrect Usage

- Negated Condition (with "else" clause)

No

```
b = True
if (not b):
    print("no")
else:
    print("yes")
```

Yes

```
b = True
if (b):
    print("yes")
else:
    print("no")
```

# Incorrect Usage [2]

- Empty "if" clause

No

```
b = False
if (b):
    pass
else:
    print("no")
```

Yes

```
b = False
if (not b):
    print("no")
```

# Incorrect Usage [3]

- Using "if" instead of "and"

No

```
b1 = True
b2 = True
if (b1):
    if (b2):
        print("both!")
```

Yes

```
b1 = True
b2 = True
if (b1 and b2):
    print "both!"
```

- Avoiding "else"

No

```
b = True
if (b):
    print("yes")
if (not b):
    print("no")
```

Yes

```
b = True
if (b):
    print("yes")
else:
    print("no")
```

# Incorrect Usage [4]

- Using **Boolean logic** instead of **"if"**

No

```
x = 42
y = ((x > 0) and 99)
```

# Or:

```
x = 42
y = (((x > 0) and 99) or
      ((x < 0) and 88) or
      77)
```

Yes

```
x = 42
if (x > 0):
    y = 99
```

# Or:

```
x = 42
if (x > 0):
    y = 99
elif (x < 0):
    y = 88
else:
    y = 77
```

# Incorrect Usage [5]

- Using **Boolean arithmetic** instead of **"if"**

No

```
x = 42
y = ((x > 0) * 99)
```

Yes

```
x = 42
if (x > 0):
    y = 99
else:
    y = 0
```

# Or:

```
y = 99 if (x > 0) else 0
```

# References

- <http://www.cs.cmu.edu/~112/notes/notes-data-and-exprs.html>
- <http://www.kosbie.net/cmu/summer-12/15-112/handouts/notes-conditionals.html>
- Gutttag, John V. *Introduction to Computation and Programming Using Python, Revised*