# LGT3109
# Introduction to Coding for Business with Python
# (week 3)

Xiaoyu Wang

Dept. of LMS, The HK PolyU

# Summary of Week 1

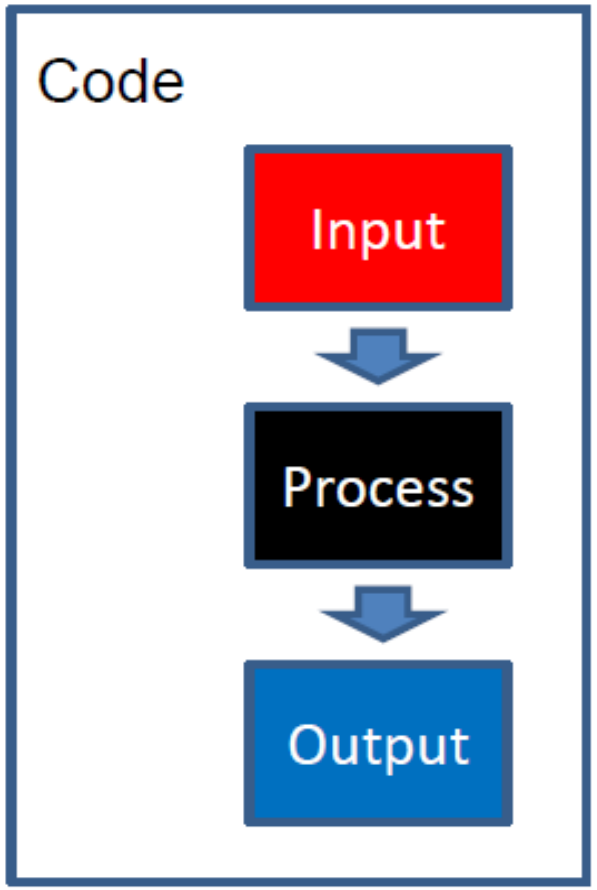
- Code is a sequence of instructions for computers to work as the tool.
- Python is a coding language for expression of the instructions

- Python code structure:

➢Input

➢Process

➢Output



# Summary of Week 1

- Python code component:
  - ***Value***, ***Variable***
  - ***Statement***
  - ***Script***

- Python code step:
  - ***Sequential steps***
  - ***Repeated steps***
  - ***Conditional steps***

# Summary of Week 2

- Value: Number (integer, float)+String
- Variable: value can be updated
- Reserved word: upper/lower case

- Statement: input statement, print statement, assignment statement
- Expression: value + variable + operators

# Conditional Execution and Flow Chart

- One-way decisions
  - ➢Flow Chart
  - ➢Coding-Boolean Expression, Operator, Indentation
- Two-way decisions
- Multiway decisions
- Nested decisions

# Foxconn Case

- Foxconn plans to ship containers from Hong Kong to San Francisco.
- A carrier quotes shipping rate (USD/container) and volume discount:
- Shipping rate = 1000 USD/container.
- If 200> shipping containers >= 100, then 10% discount is applied.
- If shipping containers >= 200, then 20% discount is applied.

| <100 | >=100 and <200 | >200 |
|---|---|---|
| Input:<br>What's the shipping quantity? 50<br>Output:<br>The total cost is 50000.0. | Input:<br>What's the shipping quantity? 150<br>Output:<br>The total cost is 135000.0. | Input:<br>What's the shipping quantity? 250<br>Output:<br>The total cost is 200000.0. |

# Foxconn Case-Python Code

- What's Foxconn's shipping cost?
- A carrier quotes shipping rate (USD/container) and volume discount:
- Shipping rate = 1000 USD/container.
- If 200> shipping containers >= 100, then 10% discount is applied.
- If shipping containers >= 200, then 20% discount is applied.

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity < 200 and quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

Conditional Execution

Comparison and Boolean Expression

# Foxconn Case-If Condition

- We are familiar with assignment statement.
  - Variable=expression

```
#process
total_cost= 1000*quantity
```

- In the conditional statement:
  - Boolean expression: True or False
  - e.g., quantity>=200, quantity=20, 2000
  - Assignment statement, print statement…
  - If Boolean expression is True: execute!

```
#conditional steps
if quantity <200 and quantity >= 100:
    total_cost=total_cost*(1-0.1)

if quantity >=200:
    total_cost = total_cost*(1-0.2)
```

# Conditional Execution and Flow Chart

- One-way decisions
  - Flow Chart
  - Coding-Boolean Expression, Operator, Indentation
- Two-way decisions
- Multiway decisions
- Nested decisions

# Flow Chart

- We have a problem, and flow chart helps us to understand it. Especially if you need to present this in a meeting.

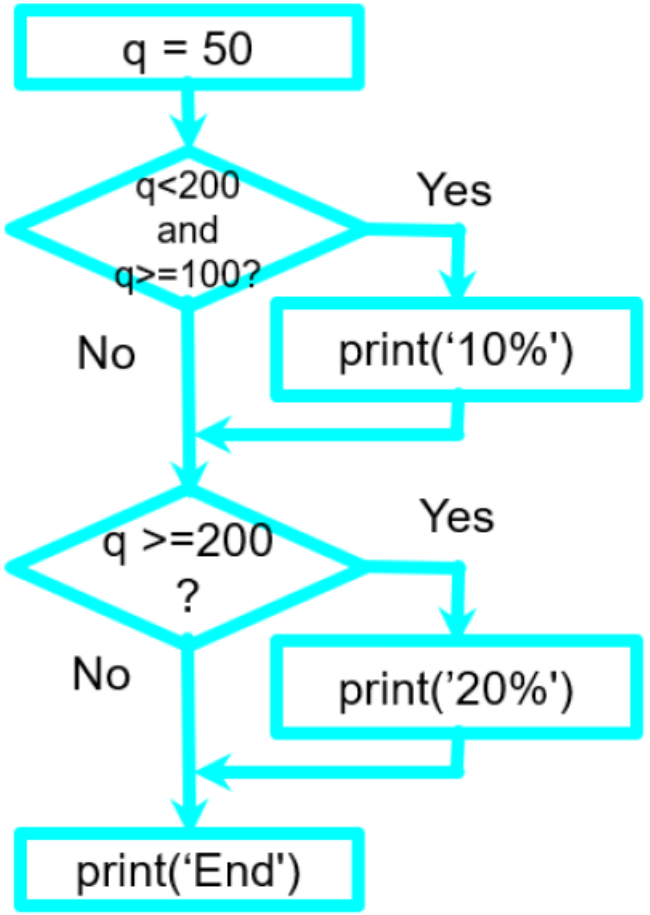

- We have already known the Foxconn problem.
- Deal with flow chart and coding.

# Flow Chart

- Flowchart: diagram representing a workflow or process.

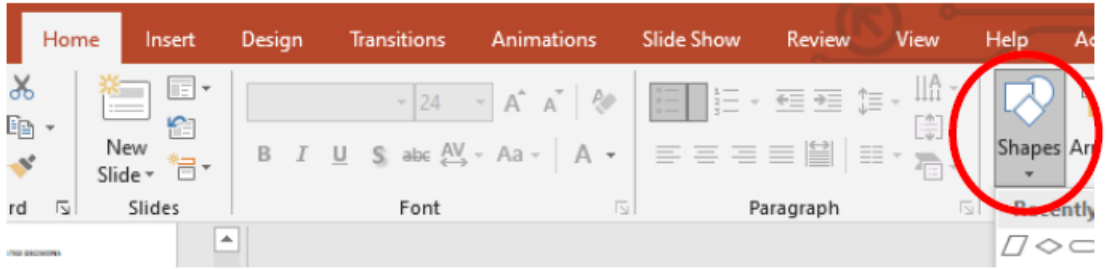
- Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

➢Interfaces between business users and software developers.



# Flow Chart-Basic Symbols

- Software to draw flowchart
- MS Word, MS PowerPoint , MS Excel.

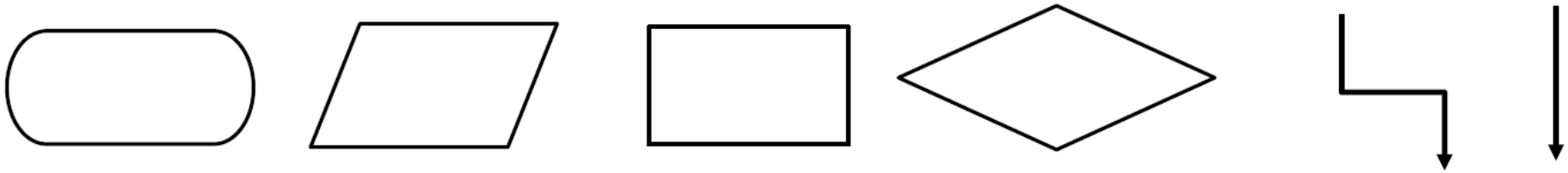


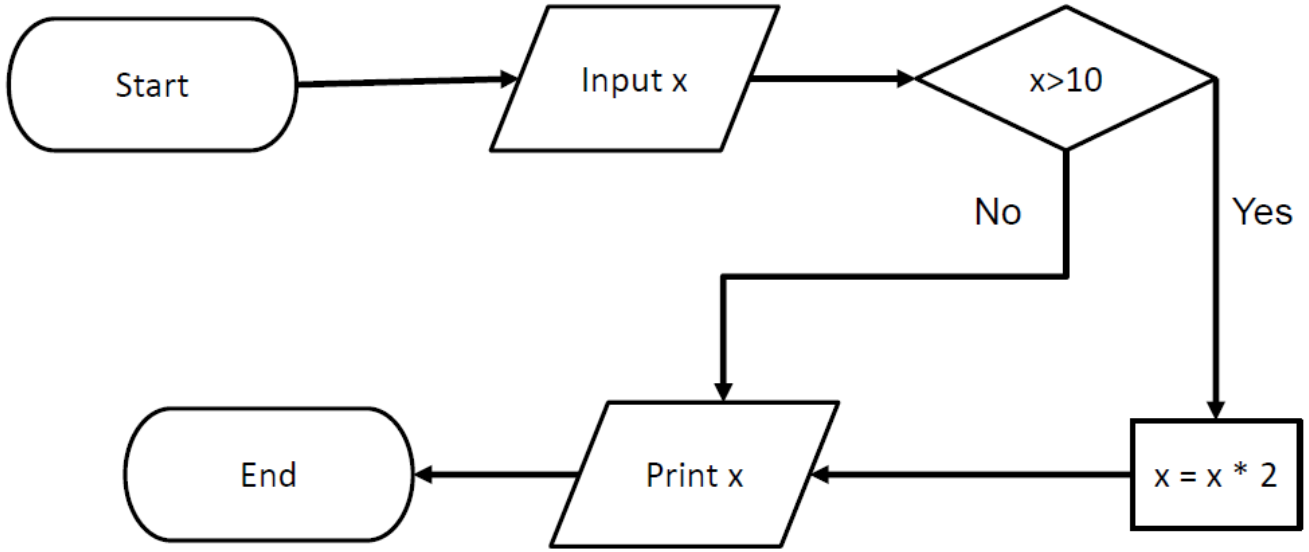


| Symbol | Name | Description |
|---|---|---|
| | Terminal | Beginning/ending of a program |
| | Process | A set of operations |
| | Decision | A conditional operation (True/False) |
| | Input/ Output | Processing input/output data |
| | Flowline (arrowhead) | Showing the order of operations |

# Flow Chart-Example

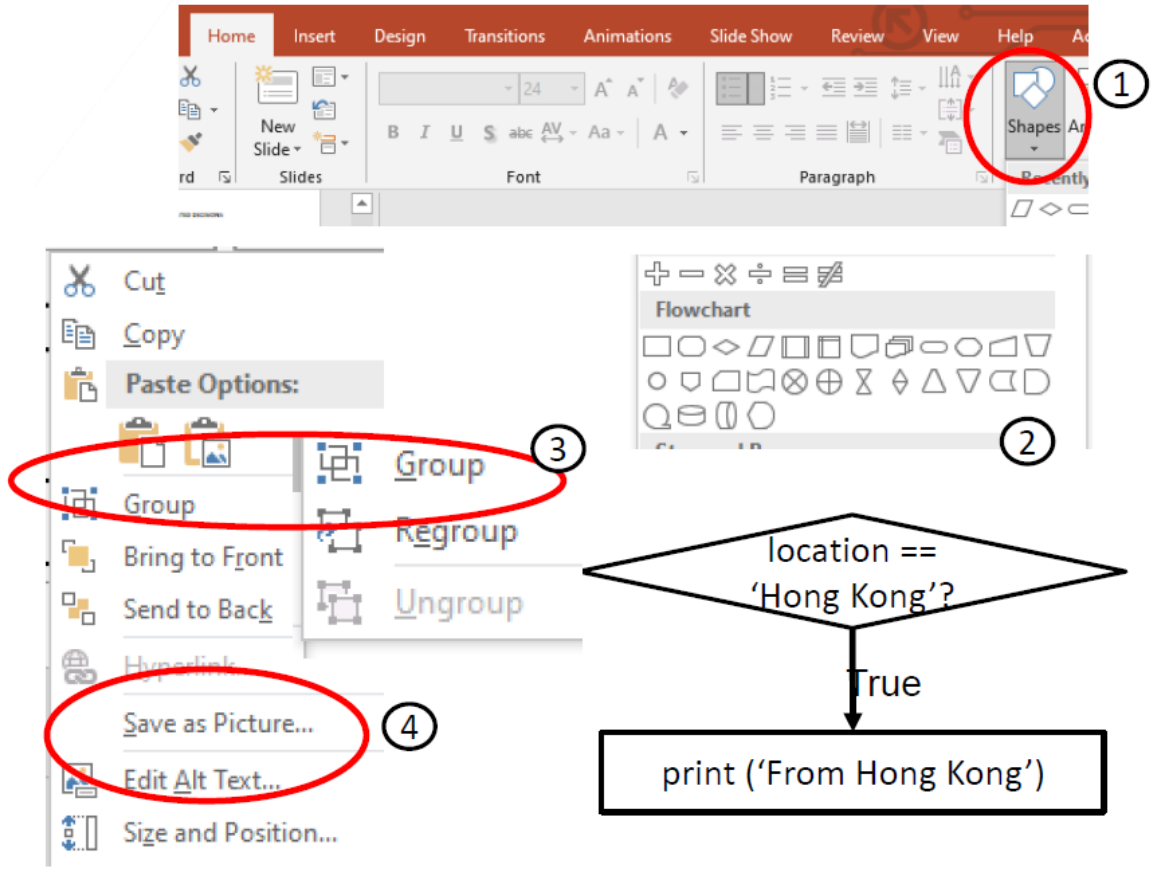
- What does this flow chart mean? Can you code it?





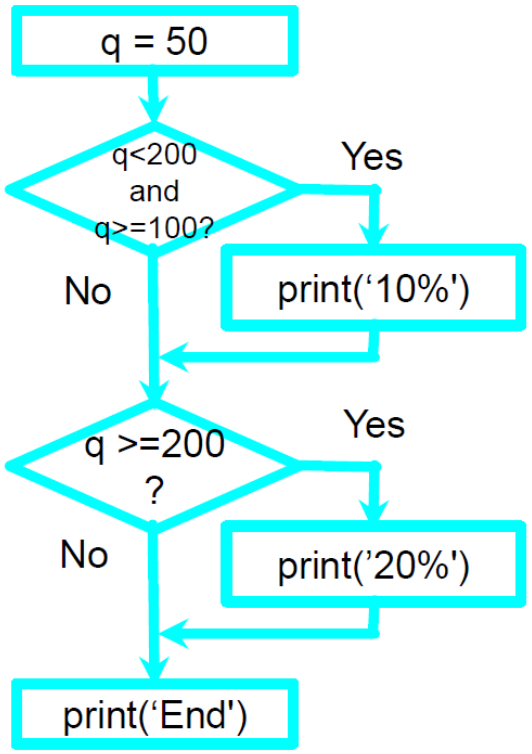
# Flow Chart-Saving Flow Chart

1. Click Shapes under “Home”.
2. Choose “Flowchart” under shapes to draw flowcharts.
3. Select all the shapes in the flowchart, right click them, and group them.
4. Right click the group of shapes, select “Save as Picture” to save the flow chart.



# Flow Chart-More Exercise

**Flow chart**




## Conditional Steps

Program:

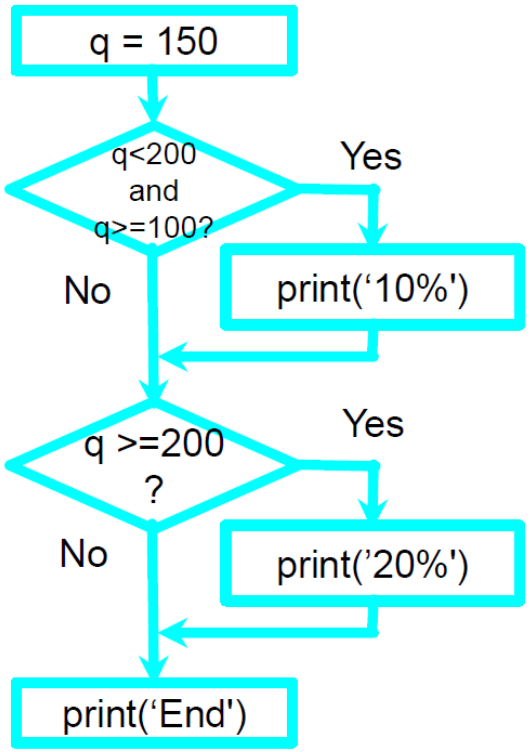
```
q = 50
if q < 200 and q >= 100:
    print('10%')
if q >= 200:
    print('20%')

print('End')
```

Output:

END

# Flow Chart-More Exercise

**Flow chart**




## Conditional Steps

Program:

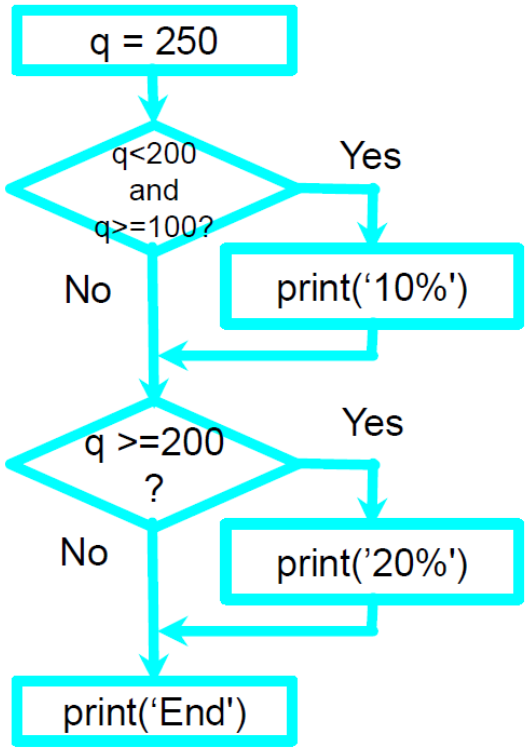
```
q = 150
if q < 200 and q >= 100:
    print('10%')
if q >= 200:
    print('20%')

print('End')
```

Output:

```
10%
END
```

# Flow Chart-More Exercise

**Flow chart**




## Conditional Steps

Program:

```
q = 250
if q < 200 and q >= 100:
    print('10%')
if q >= 200:
    print('20%')

print('End')
```

Output:

20%
END

# Conditional Execution and Flow Chart

- One-way decisions
  - ➢Flow Chart
  - ➢Coding-Boolean Expression, Operator, Indentation
- Two-way decisions
- Multiway decisions
- Nested decisions

# Coding-Composition

- Structure of conditional statement:

```
if condition :
    statements
```

```
if x > 2 :
    print('Bigger')
```

- Boolean expression
- Operator
- Indentation
- If Boolean expression is True, the statement is executed!

# Coding-Boolean Expression

- A Boolean expression is either True or False.
- True and False are special values that belong to the class (or type) bool.
- A Boolean expression consists of comparison operators and logical operators.

➢quantity < 200 and quantity >= 100

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

# Coding-Comparison Operators

- Boolean expressions

➢Asking a question and producing a Yes/No (True/False) result which we use to control program flow.

➢Using comparison operators to evaluate True/False or Yes/No.

- Comparison operators

➢Look at values of variables but do not change the values of variables.

| Python | Meaning |
|---|---|
| < | Less than |
| <= | Less than or Equal to |
| == | Equal to |
| >= | Greater than or Equal to |
| > | Greater than |
| != | Not equal |

Remember: "=" is used for assignment.

# Coding-Comparison Operators

```
x = 5
if x == 5 :
    print('Equals 5')
if x > 4 :
   print('Greater than 4')
if  x >= 5 :
    print('Greater than or Equals 5')
if x < 6 : print('Less than 6')
if x <= 5 :
    print('Less than or Equals 5')
if x != 6 :
    print('Not equal 6')
```

Equals 5

Greater than 4

Greater than or Equals 5

Less than 6

Less than or Equals 5

Not equal 6

# Coding-Logic Operators

- There are three logical operators.

➢and

➢or

➢not

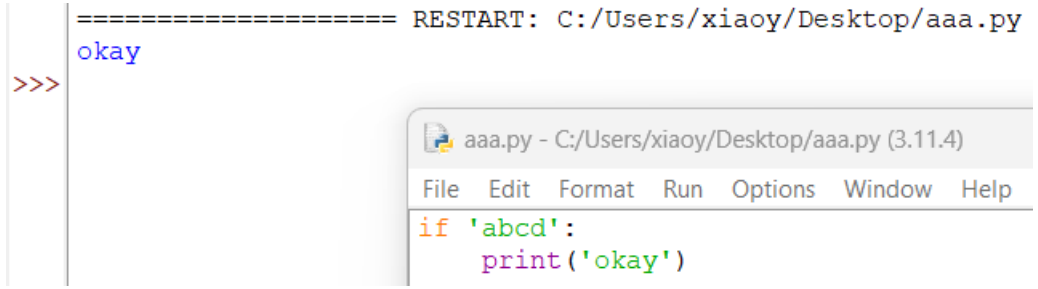
| and | Operands 1 | Operands 2 | Result |
|---|---|---|---|
| | True | True | True |
| | True | False | False |
| | False | True | False |
| | False | False | False |

| or | Operands 1 | Operands 2 | Result |
|---|---|---|---|
| | True | True | True |
| | True | False | True |
| | False | True | True |
| | False | False | False |

| not | Operands | Result |
|---|---|---|
| | True | False |
| | False | True |

# Coding-Logic Operators

- In python, any nonzero number can be interpreted as True.




| and | Operands 1 | Operands 2 | Result |
|---|---|---|---|
| | True | True | True |
| | True | False | False |
| | False | True | False |
| | False | False | False |

| or | Operands 1 | Operands 2 | Result |
|---|---|---|---|
| | True | True | True |
| | True | False | True |
| | False | True | True |
| | False | False | False |

| not | Operands | Result |
|---|---|---|
| | True | False |
| | False | True |

# Coding-Logic Operators

- Some examples:

```
>>>
========== RESTART: C:/Users/xiaoy/Desktop/aaa.py ==========
between 4 and 6
either smaller than 3 or larger than 4
>>>
```

aaa.py - C:/Users/xiaoy/Desktop/aaa.py (3.11.4)

File Edit Format Run Options Window Help

```
x=5

if x>4 and x<6:
    print('between 4 and 6')
if x>5 and x<6:
    print('between 5 and 6')
if x<3 or x>4:
    print('either smaller than 3 or larger than 4')
if x<3 or x>6:
    print('either smaller than 3 or larger than 6')
```

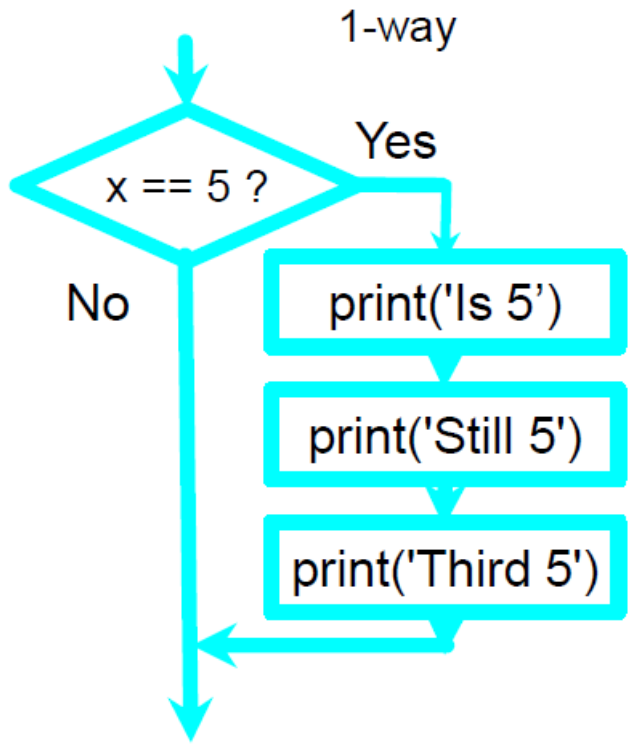
| | Operands 1 | Operands 2 | Result |
|---|---|---|---|
| and | True | True | True |
| | True | False | False |
| | False | True | False |
| | False | False | False |

| | Operands 1 | Operands 2 | Result |
|---|---|---|---|
| or | True | True | True |
| | True | False | True |
| | False | True | True |
| | False | False | False |

| | Operands | Result |
|---|---|---|
| not | True | False |
| | False | True |

# Coding-One-Way Decisions

- We look at the Boolean expression:
- If True, execute!
- One way means we only care about True case.

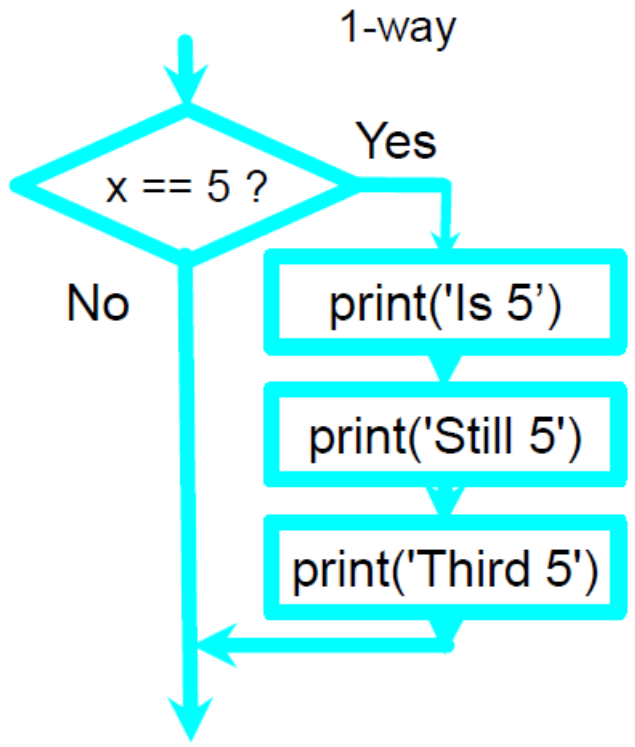


# Coding-One-Way Decisions

```
x = 5
print('Before 5')
if  x == 5 :
    print('Is 5')
    print('Is Still 5')
    print('Third 5')
print('Afterwards 5')
```

Before 5

Is 5
Is Still 5
Third 5
Afterwards 5



# Coding-One-Way Decisions-code

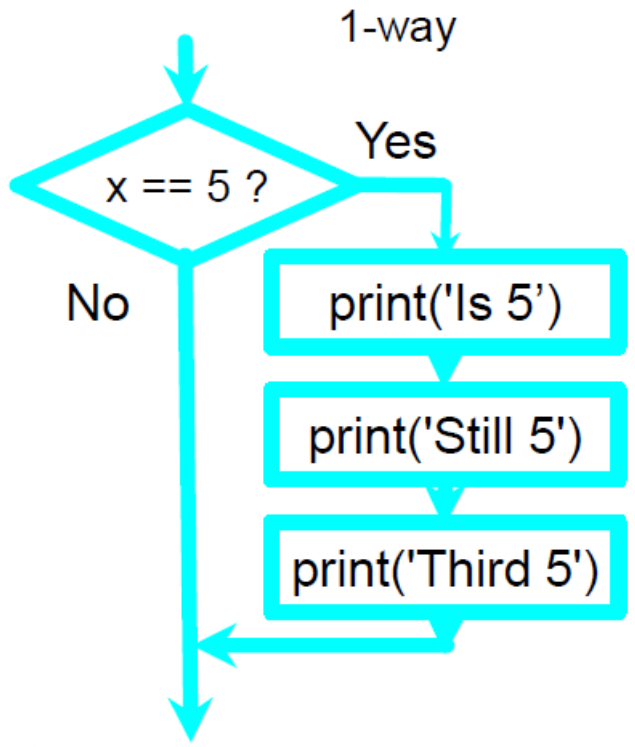
Condition is true ➔ indented statements are executed.
Condition is false ➔ indented statements are skipped.

if *condition* :
    statement
    statement
    .....

*indentation* to indicate a block

A *block* (body) for the *if statement*:
Including a *header* (:)
and indented statements

*if statement is a compound statement*




```
if  x == 5 :
    print('Is 5')
    print('Is Still 5')
    print('Third 5')
```

# Coding-Indentation

- Increase indent (TAB key / space Key)

➢Indent after an if statement (after: )

- Maintain indent

➢Indicate lines are affected by if/for statements)

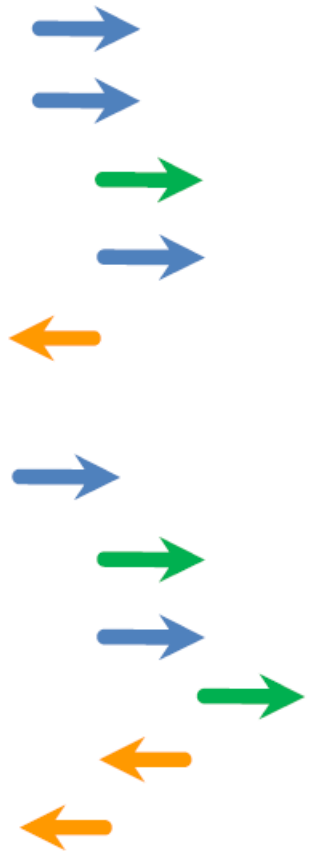
- Reduce indent (Backspace Key)

➢Indicate the end of the block

```
x = 5
print('Before 5')
if  x == 5 :
    print('Is 5')
    print('Is Still 5')
    print('Third 5')
print('Afterwards 5')
```

# Coding-Indentation

- Increase to “enter” the block.
- Maintain to stay in the block.
- Reduce to indicate end of block.



```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')

for i in range(5) :
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

# Coding-Indentation

```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')
```

Nested Blocks

```
for i in range(5) :
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

# Coding-Indentation

- Blank lines are ignored. They do not affect indentation.
- Comments on a line are ignored with regards to indentation.

```
x = 5
print('Before 5')
if  x == 5 :
    print('Is 5')

    #print('Is Still 5')
    print('Third 5')
print('Afterwards 5')
```

# Coding-Foxconn Case

- A carrier quotes a shipping rate (USD/container) and volume discounts:
- shipping rate = 1000 USD/container.
- If 200> containers >= 100, then 10% discount is applied.
- If containers >= 200, then 20% discount is applied.

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity < 200 and quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

# Coding-Summary

- if condition: statements
- Boolean Expression

➢Comparison operators: == <= >= > < !=

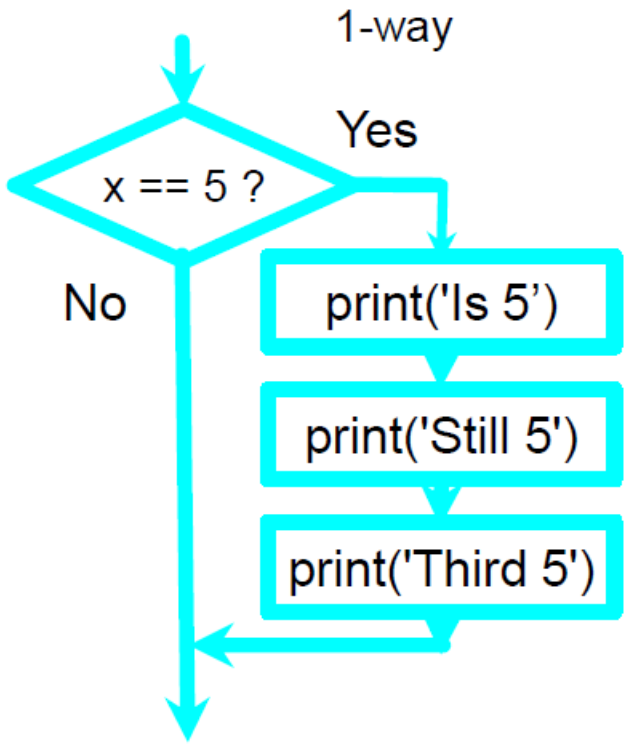
➢Logic operators: and, or, not

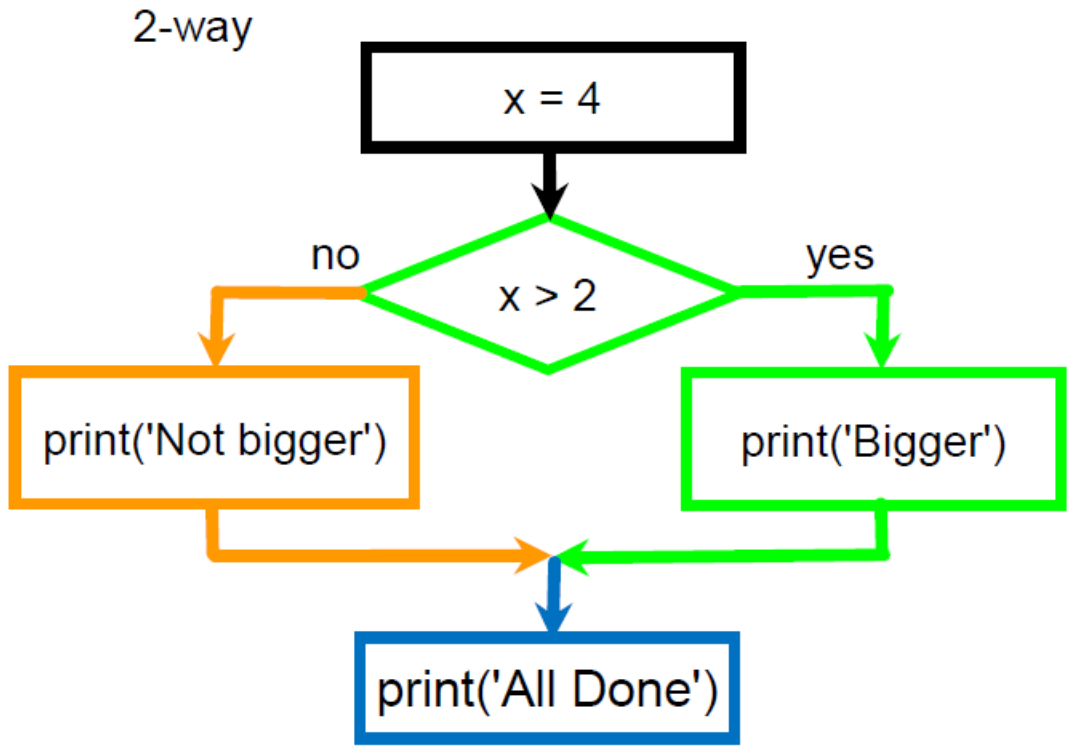
- Indentation: indicate start/end block
- One-way Decisions

# Conditional Execution and Flow Chart

- One-way decisions
  - ➢Flow Chart
  - ➢Coding-Boolean Expression, Operator, Indentation
- Two-way decisions
- Multiway decisions
- Nested decisions

# Two-Way Decisions

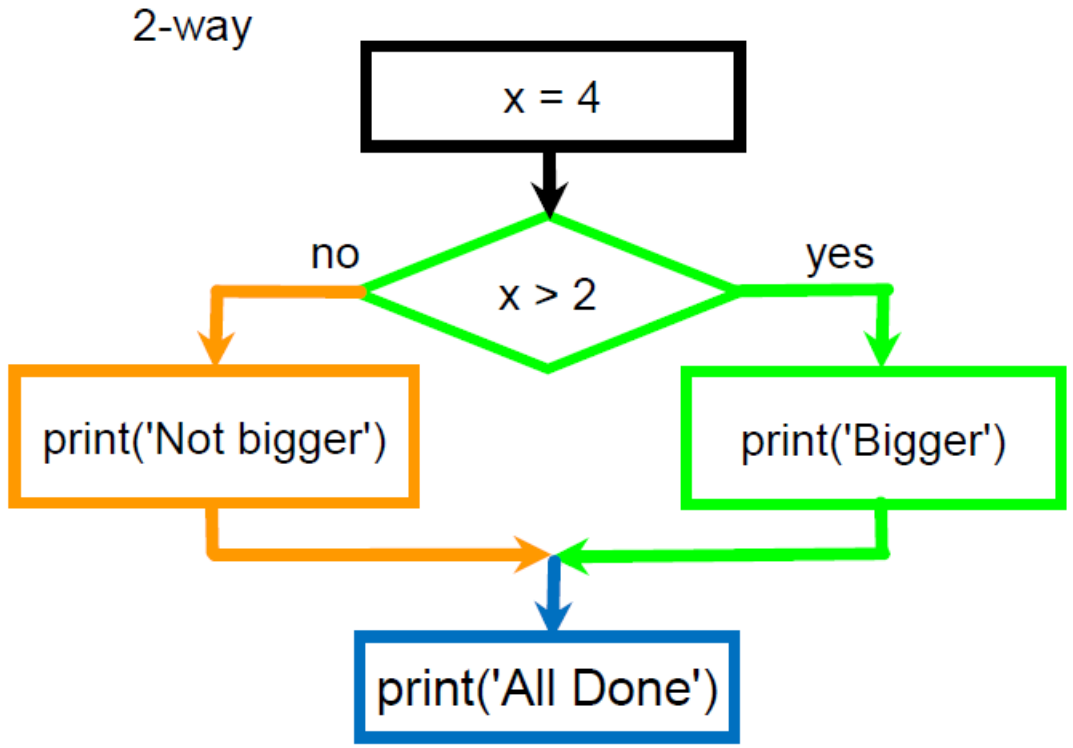


# Two-Way Decisions

- Sometimes we want to do one thing if a Boolean expression is True and something else if the expression is False.

- It is like an intersection in the road. We must choose one or the other path, but not both.



# Two-Way Decisions-Code

Condition is true ➔ indented statements under if are executed.
Condition is false ➔ indented statements under else are executed.

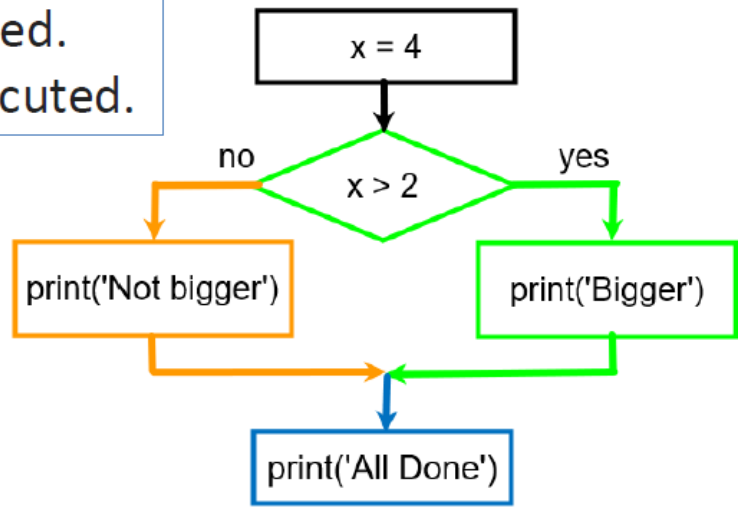
if *condition* :
    statements

A *block* for the *if statement*:

else :
    statements

A *block* for the *else statement*:

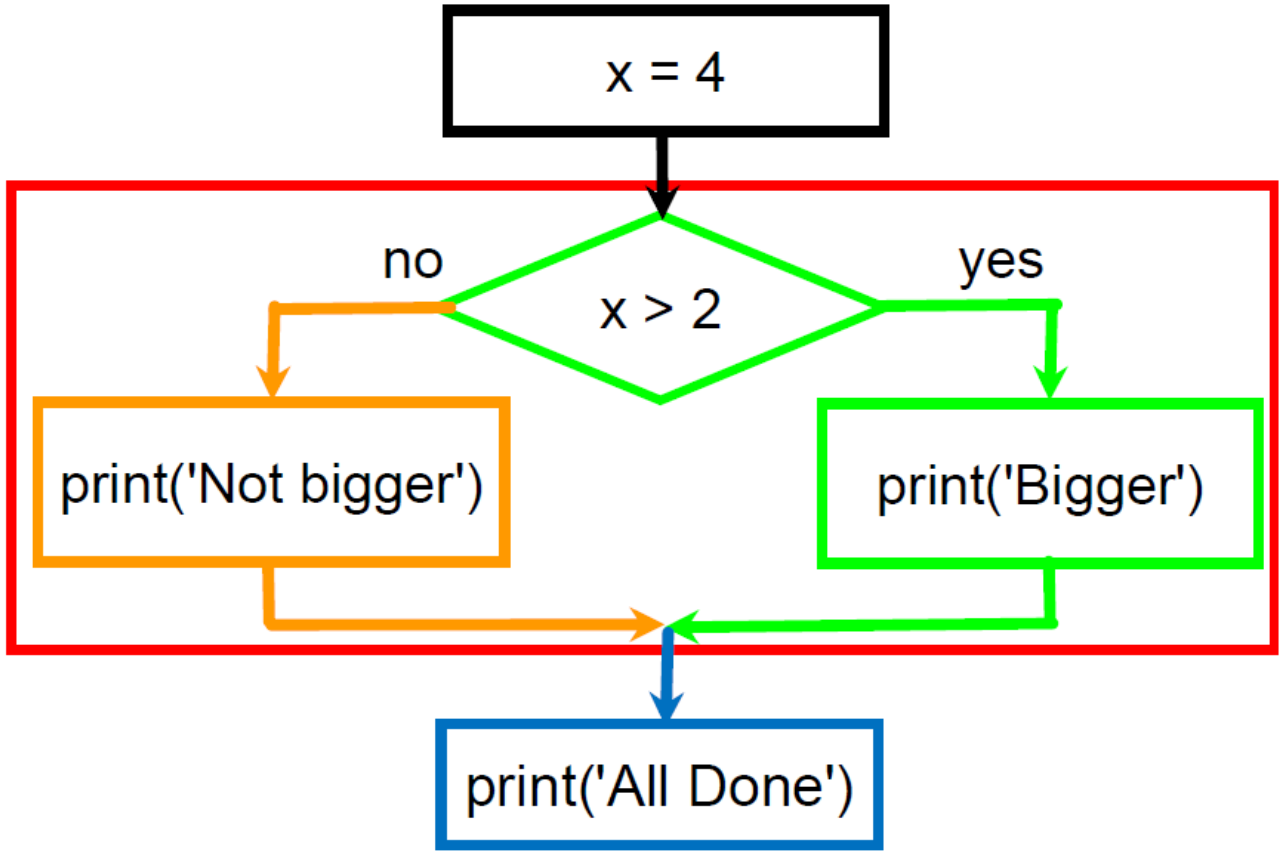
*indentation* to indicate a block




```
x = 4
if x > 2 :
    print('Bigger')
else :
    print('Not bigger')
print('All Done')
```

# Two-Way Decisions-Visualize Blocks

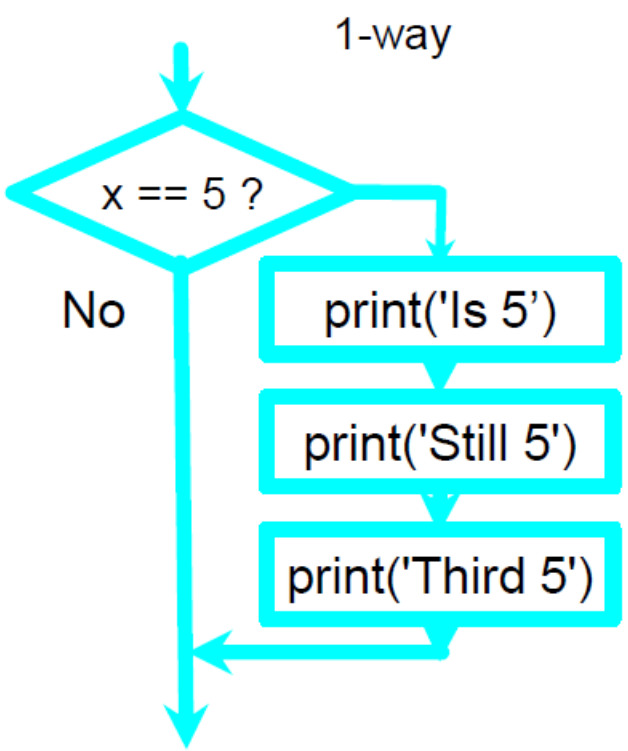
```
x = 4

if x > 2 :
    print('Bigger')
else :
    print('Not bigger')

print('All Done')
```

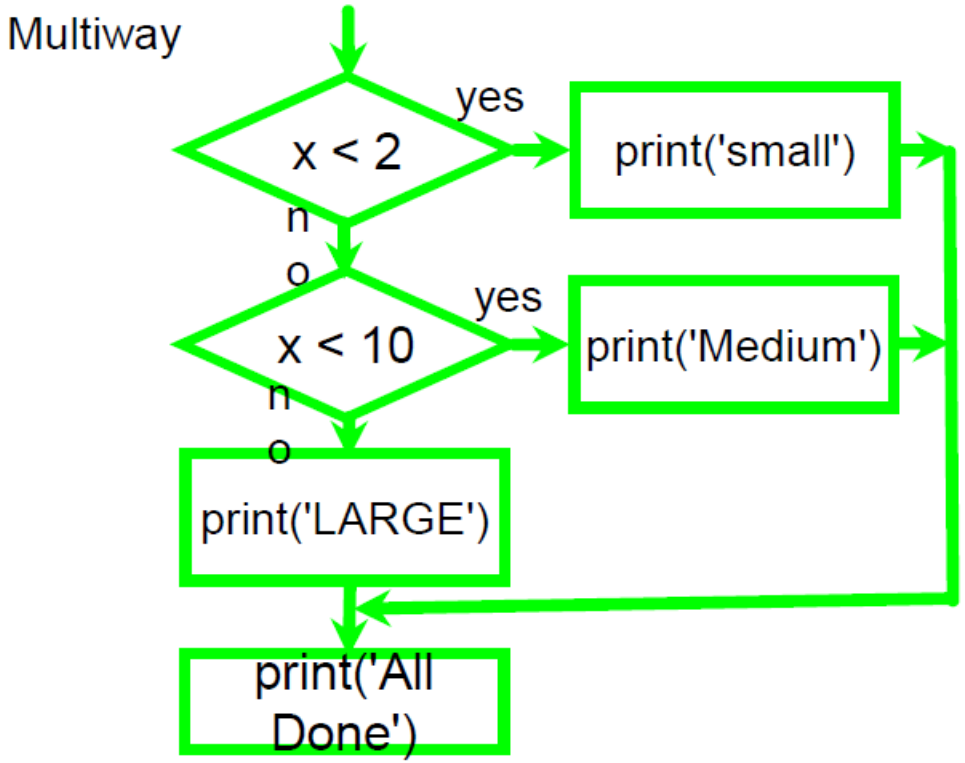


# Conditional Execution and Flow Chart

- One-way decisions
  - ➢Flow Chart
  - ➢Coding-Boolean Expression, Operator, Indentation
- Two-way decisions
- Multiway decisions
- Nested decisions

# Multiway Decisions

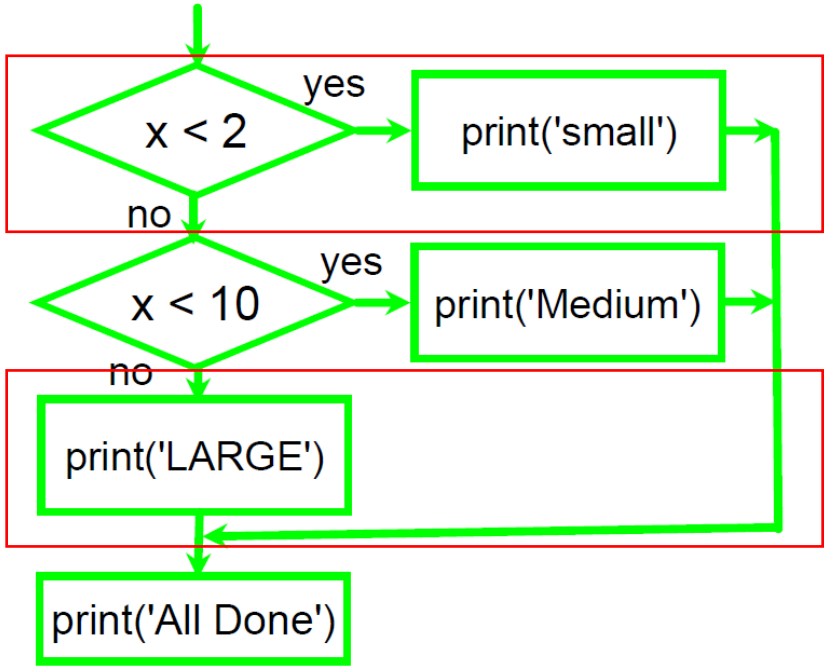


# Multiway Decisions

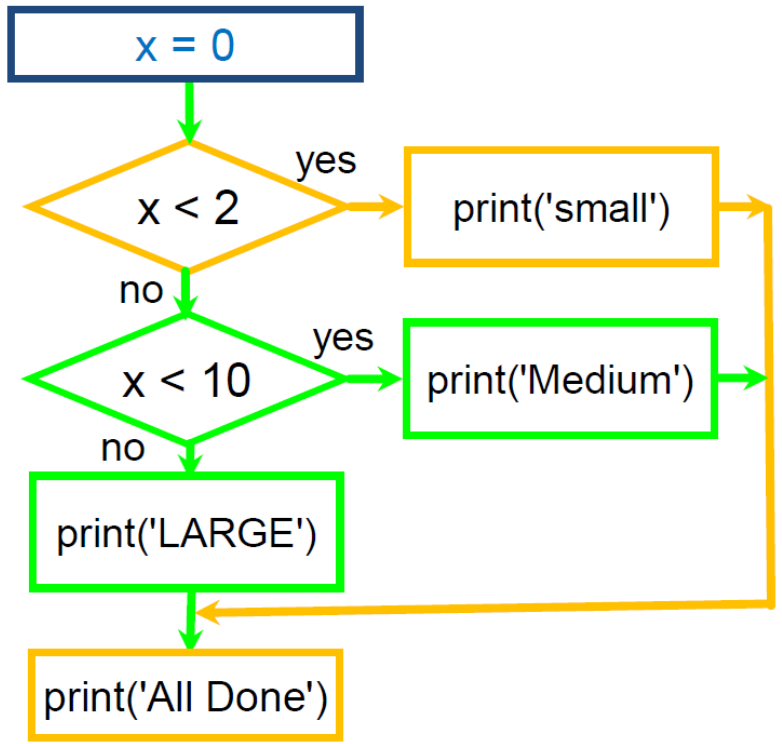
Sometimes there are more than two possibilities, and we need more than two ways (branches).

```
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All Done')
```



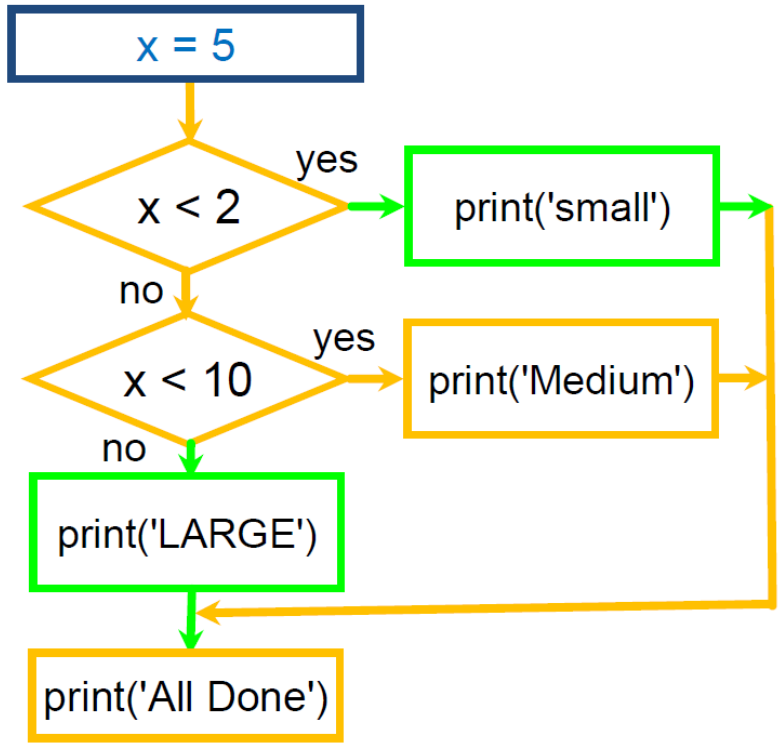
# Multiway Decisions

```
x = 0
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All Done')
```



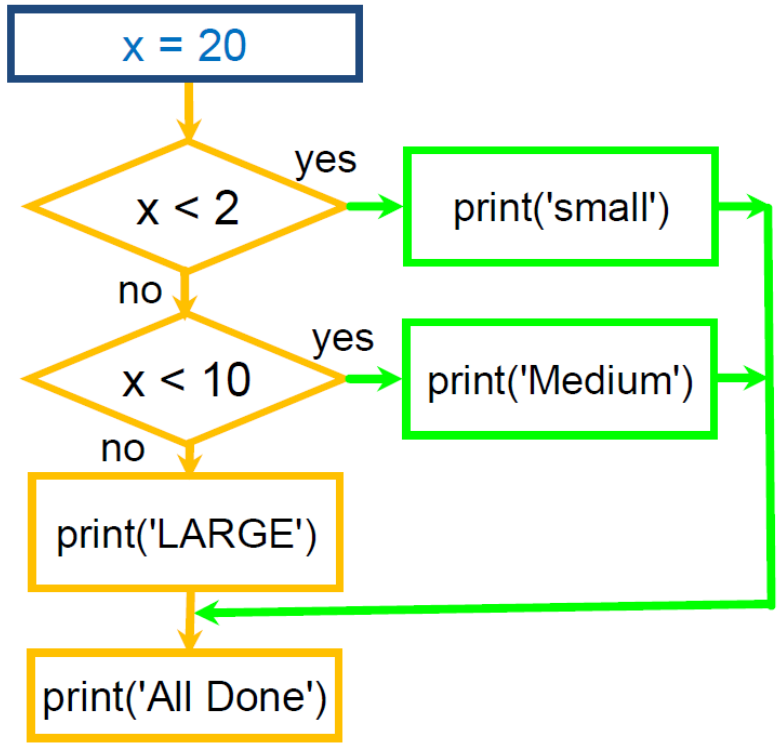
# Multiway Decisions

```
x = 5
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All Done')
```



# Multiway Decisions

```
x = 20
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All Done')
```



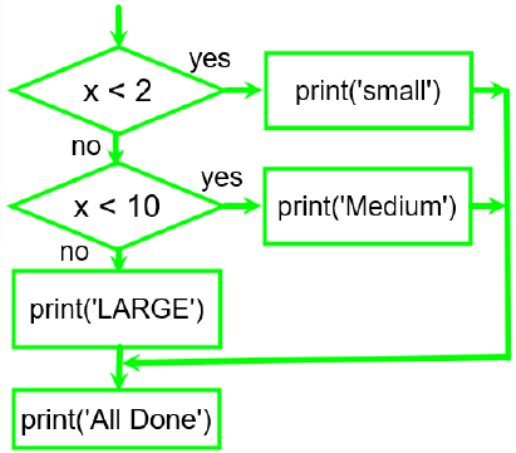
# Multiway Decisions-Code

- elif = else if
- No limit on the number of “elif”
- “else” can be at the end, but there doesn't have to be one
- Each condition is checked in order.
- Only statements for the first satisfied condition is executed

if *condition 1*:
    statements
} A *block* for the *if statement*:

elif *condition 2*:
    statements
} A *block* for the *elif statement*:

else :
    statements
} A *block* for the *else statement*:




```
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```

# Multiway Decisions-Variation

- Not necessary to have else.
- Like one-way decisions with elif.

```
# No Else
x = 5
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')

print('All Done')
```

```
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')
elif x < 20 :
    print('Big')
elif x < 40 :
    print('Large')
elif x < 100:
    print('Huge')
else :
    print('Ginormous')
```

# Multiway Decisions-Test

- Which print statements will never be executed regardless of the value for x?

```
if x < 2 :
    print('Below 2')
elif x >= 2 :
    print('Two or more')
else :
    print('Something else')
```

```
if x < 2 :
    print('Below 2')
elif x < 20 :
    print('Below 20')
elif x < 10 :
    print('Below 10')
else :
    print('Something else')
```

# Multiway Decisions-Foxconn Case

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity < 200 and quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
elif quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

# Conditional Execution and Flow Chart

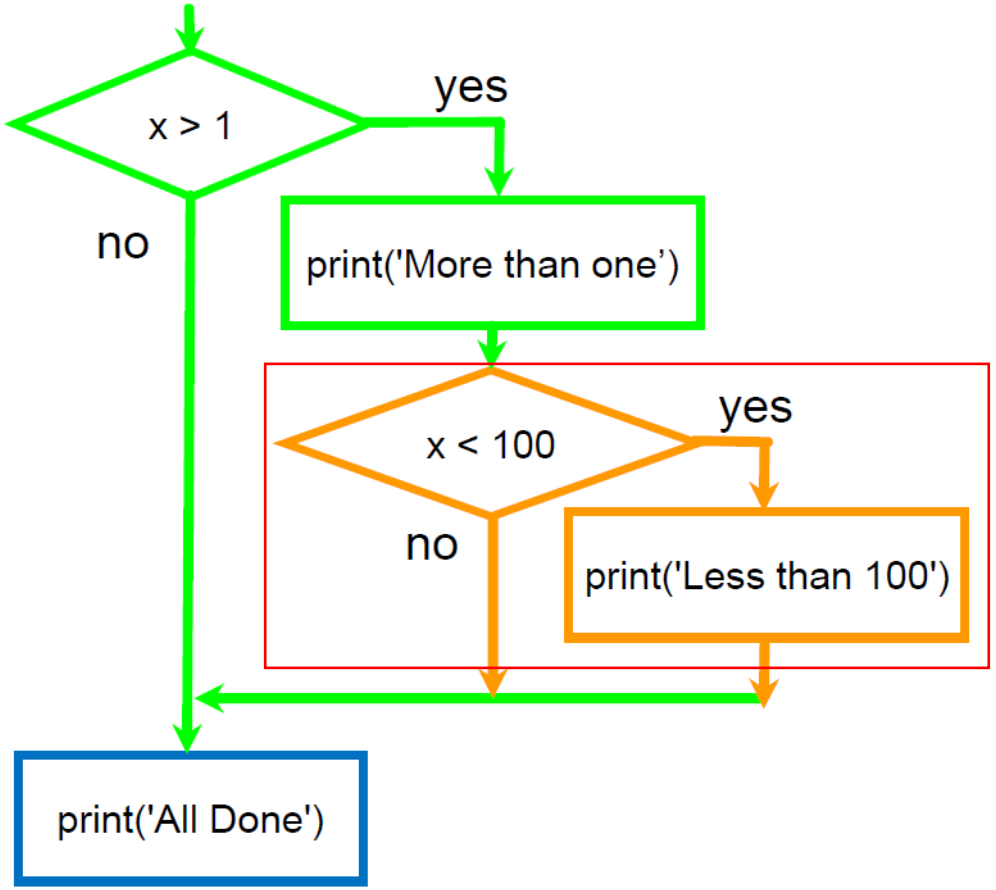
- One-way decisions
  - ➢Flow Chart
  - ➢Coding-Boolean Expression, Operator, Indentation
- Two-way decisions
- Multiway decisions
- Nested decisions

# Nested Decisions

- One block inside one block.

➢One-way decision in one-way decision.

```
x = 42
if x > 1 :
    print('More than one')
    if x < 100 :
        print('Less than 100')
print('All Done')
```




- Pay attention to the indentation!

# Nested Decisions-Suggestions

- Although the indentation of the statements makes the structure apparent, nested conditionals become difficult to read.
- In general, it is a good idea to avoid them when you can.
- Use logical operators to simplify nested conditional statements.

Nested:

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

Simplified:

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

# Nested Decisions-Foxconn Case

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity < 200:
    if quantity >= 100:
        total_cost = total_cost * (1.0 - 0.1)
else:
    total_cost = total_cost * (1.0 - 0.2)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

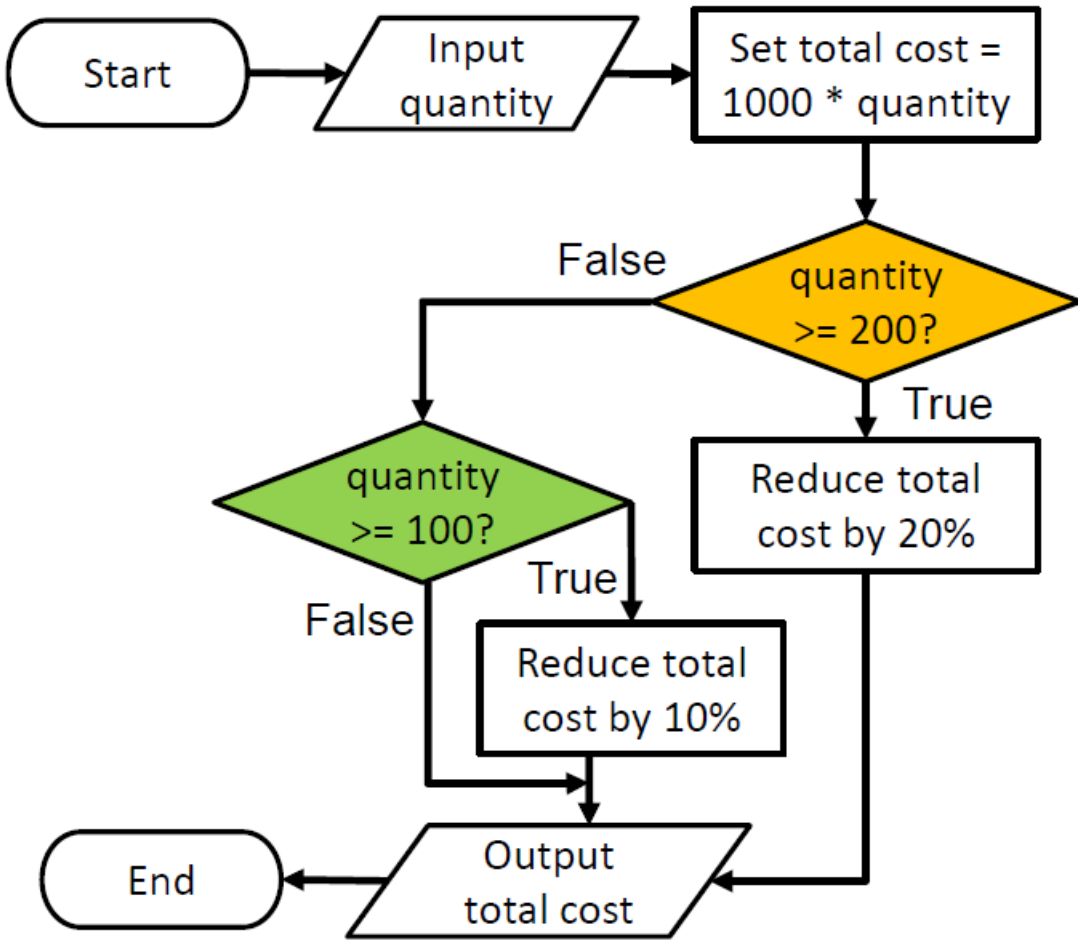
```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
elif quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity < 200 and quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
#output:
print('The total cost is ' + str(total_cost) + '.')
```

# Nested Decisions-Solve the Problem

- Have a question.
- Build a flow chart.
- Write the code.

```
#input:
quantity = int(input('What is the shipping quantity? '))
#process:
total_cost = 1000.0 * quantity
#conditional steps:
if quantity >= 200:
    total_cost = total_cost * (1.0 - 0.2)
elif quantity >= 100:
    total_cost = total_cost * (1.0 - 0.1)
#output:
print('The total cost is ' + str(total_cost) + '.')
```



# Summary

- if condition: statements
- Boolean Expression
  - Comparison operators: == <= >= > < !=
  - Logic operators: and, or, not
- Indentation: indicate start/end block
- One-way Decisions
- Two-way decisions
  - if: statements else: statements
- Multi way decisions
  - If: statements elif : statements else: statements
- Nested Decisions: better to avoid.
- Problem to Flowchart to Code.

# Acknowledgement


- Acknowledgements / Contributions



- Initial Development: Charles Severance, University of Michigan School of Information
- Further Development: Zhou Xu, Hong Kong Polytechnic University
- Continuous development: Xiaoyu Wang, Hong Kong Polytechnic University