Python: Conditionals (條件判斷)

Cheng-Te Li (李政德)

chengte@mail.ncku.edu.tw

Dept. of Statistics, NCKU

Boolean Type and Operators

- Python has a boolean type called `bool`
- `bool` has only two possible values: `True` and `False`
- In normal speech, “true” and “false” are adjectives, but `True` and `False` are Python values, just as much as the `int` value 0 or the `float` value -17.3
- Only three basic boolean operators, given in order of precedence (low to high): **`or`**, **`and`**, and **`not`**
- They have meanings in line with common usage
  - **`and`** and **`or`** are binary operators
  - **`not`** is a unary operator

Examples of Boolean Operation

```
not True  # False
not False # True

False and False
# => False
False and True
# => False
True and False
# => False
True and True
# => True
```

```
False or False
# => False
False or True
# => True
True or False
# => True
True or True
# => True
```

Truth Tables

- We define boolean operators with truth tables

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

and evaluates to `True` only if **both** its operands are `True`
or evaluates to `True` if **either or both** of its operands are `True`

Relational Operators

- Most often, boolean values are created in expressions
- The most common way is to use relational operators

Symbol	Operation
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational Operators

- Relational expressions evaluate to `True` or `False`, just like we can say whether a mathematical statement is true or false
- Note that Python uses **==** for equality (**not =**)
- All relational operators are binary: compare two values
- We can compare integers to floats
  - integers are automatically converted to floats in comparisons

```
45 > 73   # False
45 > 23   # True
45 < 73   # True
45 < 23   # False
```

```
23.1 >= 23     # True
23.1 >= 23.1   # True
23.1 <= 23.1   # True
23.1 < 23      # False
```

```
67.3 == 87   # False
67.3 == 67   # False
67.0 == 67   # True
67.0 != 67   # False
67.0 != 23   # True
67 = 23
```

```
>>> 67 = 23
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

Relational Operators

- It doesn't make much sense to compare two numbers you know in advance
- Relational operators almost always involve variables

```
a = 123 + 789
b = 456 + 457
print(a > b)          # False
print(a == b)         # False
print(a != b)         # True
print(a == b - 1)     # True, suggest: a == (b - 1)
print((a - 1) >= b)   # False
```

Combining Operators

- **Precedence** rules for combining operators

  Arithmetic operators

  > Relational operators

  > Left to right

  – For example, + and / are evaluated before < or >

- Also, comparisons are evaluated before and, or, and not
  – For example, 1 + 3 > 7 is evaluated as (1 + 3) > 7

Parenthesis
Power
Multiplication
Addition
Left to Right

Parenthesis
Power
Multiplication
Addition
Relational
Left to Right

Combining Operators

- Often, we may omit the parentheses in complicated expressions
- However, for clarity, we'd rather leave them in

```
x, y, z = 2, 5, 7
x < y and y < z                # True
(x < y) and (y < z)            # True
not (x < z) or (z > y)         # True
not ((x < z) or (z > y))       # False
(not (x < z)) or (z > y)       # True
not (x < z) or not (z > y)     # False
```

Range Checking

- We often need to check whether a value lies in a given range
- Python allows chain comparisons

```
x = 3
(1 < x) and (x <= 5)     # True
1 < x <= 5               # True
y = 7
(1 < y) and (y <= 5)     # False
1 < y <= 5               # False
1 < x < y <= 10          # True
```

not Convert Numbers to Boolean

- Python converts integers to floats in mixed expressions
- Python also converts numbers to bools
- 0 and 0.0 are treated as False
- All other numbers are treated as True

```
not 0        # True
not 1        # False
not 12.3     # False
not -87      # False
```

Logically Equivalent Boolean Expressions

- In numerical algebra, there are arithmetically equivalent expressions of different forms

Logically Equivalent Boolean Expressions		
`x < y`	is equivalent to	`not(x >= y)`
`x <= y`	is equivalent to	`not(x > y)`
`x == y`	is equivalent to	`not(x != y)`
`x != y`	is equivalent to	`not(x == y)`
`not(x and y)`	is equivalent to	`(not x) or (not y)`
`not(x or y)`	is equivalent to	`(not x) and (not y)`

Logically Equivalent Boolean Expressions

```
not (10 >= 20)                          # True
10 != 20                                # True
not (10 == 20)                          # True
not (10 < 20 and 10 < 30)               # False
(not 10 < 20) or (not 10 < 30)          # False
not (10 < 20 or 10 < 30)                # False
(not 10 < 20) and (not 10 < 30)         # False
```

Comparing Strings

- Strings can be compared on their lexicographic order
- Uppercase letters come before the lower letters
- If a string s1 is a prefix of another, longer string s2, then s1 is “less than” s2

```
'A' < 'a'          # True
'A' < 'z'          # True
'abc' < 'abd'      # True
'abc' < 'abcd'     # True
'abc' == 'abc'     # True
'abc' != 'xyz'     # True
```

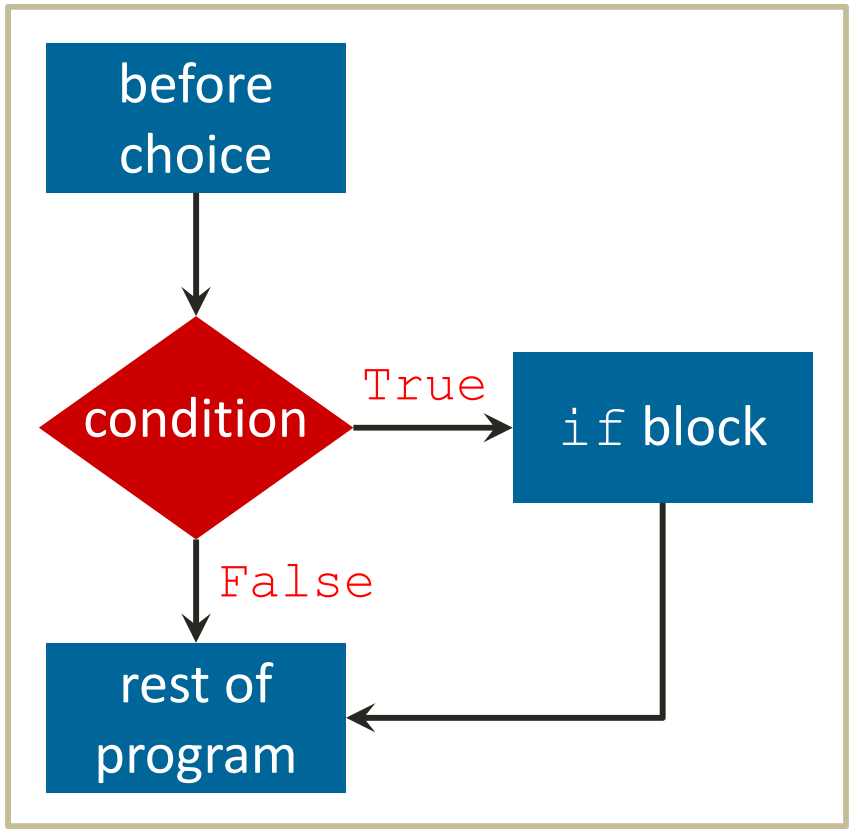
`if` statements

- Use an `if` statement to make a choice
- General form

  `if` ⟨**condition**⟩**:**

  ⟨**block**⟩

  indentation



- If ⟨**condition**⟩ is `True` the ⟨**block**⟩ is executed
- If it is `False`, ⟨**block**⟩ is skipped
- Note that the ⟨**block**⟩ must be **indented**

Condition

- Boolean expressions as a question and produce a `True` or `False` result which we can use to control program flow
  - Boolean operators (`and`, `or`, `not`)
  - Relational operators (see below)
- Comparison operators look at variables but do not change the variables

Symbol	Operation
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Code Example

Table: Solution Categories Based on pH Level

pH Level	Solution Category
0–4	Strong acid
5–6	Weak acid
7	Neutral
8–9	Weak base
10–14	Strong base

We can make Python execute certain statements when the pH level represented by some variable falls into a certain category.

Example Code

```
ph = float(input("Enter a PH value: "))
if ph < 5.0:
    print("Strong acid")
if ph < 7.0:
    print("Weak acid")
```

```
c:\Python35-32\test>python test.py
Enter a PH value: 5.7
Weak acid
```

- The body of the first if statement is not executed
- But the body of the second one is
- What about a PH of 3.7 ?

```
c:\Python35-32\test>python test.py
Enter a PH value: 3.7
Strong acid
Weak acid
```

What can we do?

Example Code: Using `if`

```python
ph = float(input("Enter a PH value: "))
if 0.0 <= ph < 5.0:
    print("Strong acid")
if 5.0 <= ph < 7.0:
    print("Weak acid")
```

```
c:\Python35-32\test>python test.py
Enter a PH value: 3.7
Strong acid
```

- This works!
- Or we could use
  - Nested `if`
  - `elif` clause

In-Class Exercise: Grading

- Write a program (grading.py)
  - Allow the user to input a score
  - Print the comment based on the table

```
c:\Python35-32\workspace>python grading.py
Please enter a score: 77
Good

c:\Python35-32\workspace>python grading.py
Please enter a score: 22
Inadequate

c:\Python35-32\workspace>python grading.py
Please enter a score: 99
Outstanding
```

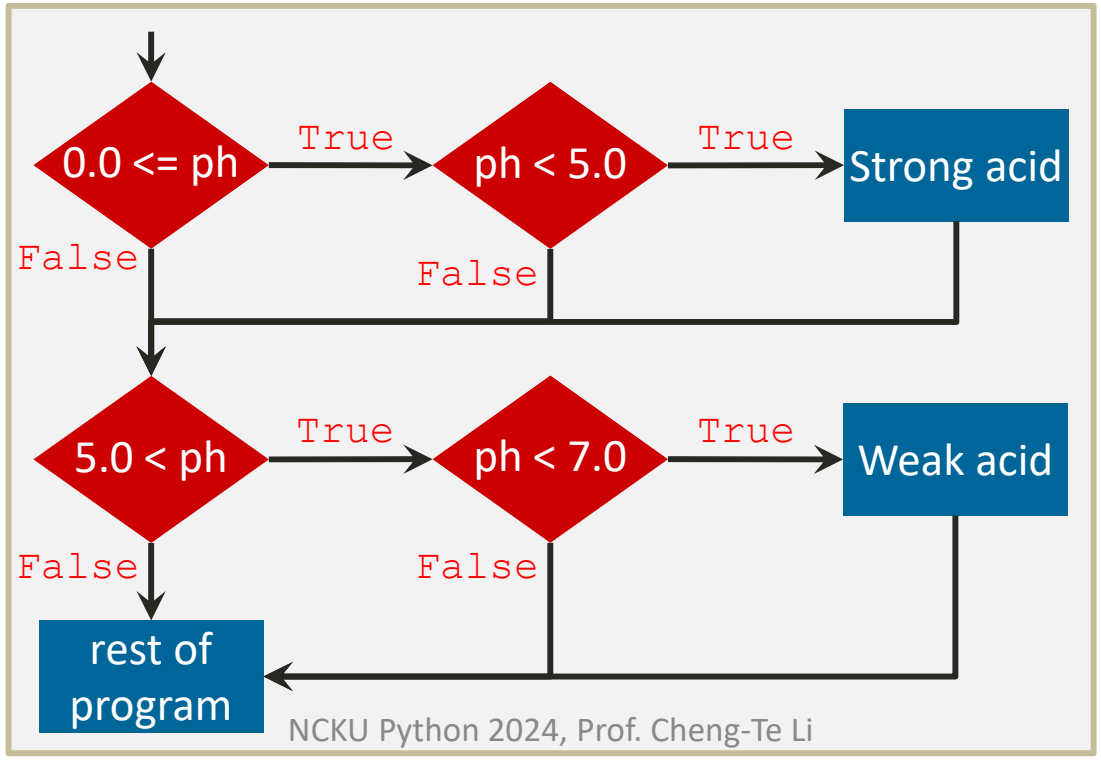
Grade	Comment
90 – 100	Outstanding
80 – 89	Exceptional
70 – 79	Good
60 – 69	Adequate
50 – 59	Marginal
0 – 49	Inadequate

In-Class Exercise: Grading

```python
score = float(input("Please enter a score: "))
if 90 <= score <= 100:
    print("Outstanding")
if 80 <= score < 90:
    print("Exceptional")
if 70 <= score < 80:
    print("Good")
if 60 <= score < 70:
    print("Adequate")
if 50 <= score < 60:
    print("Marginal")
if 0 <= score < 50:
    print("Inadequate")
```

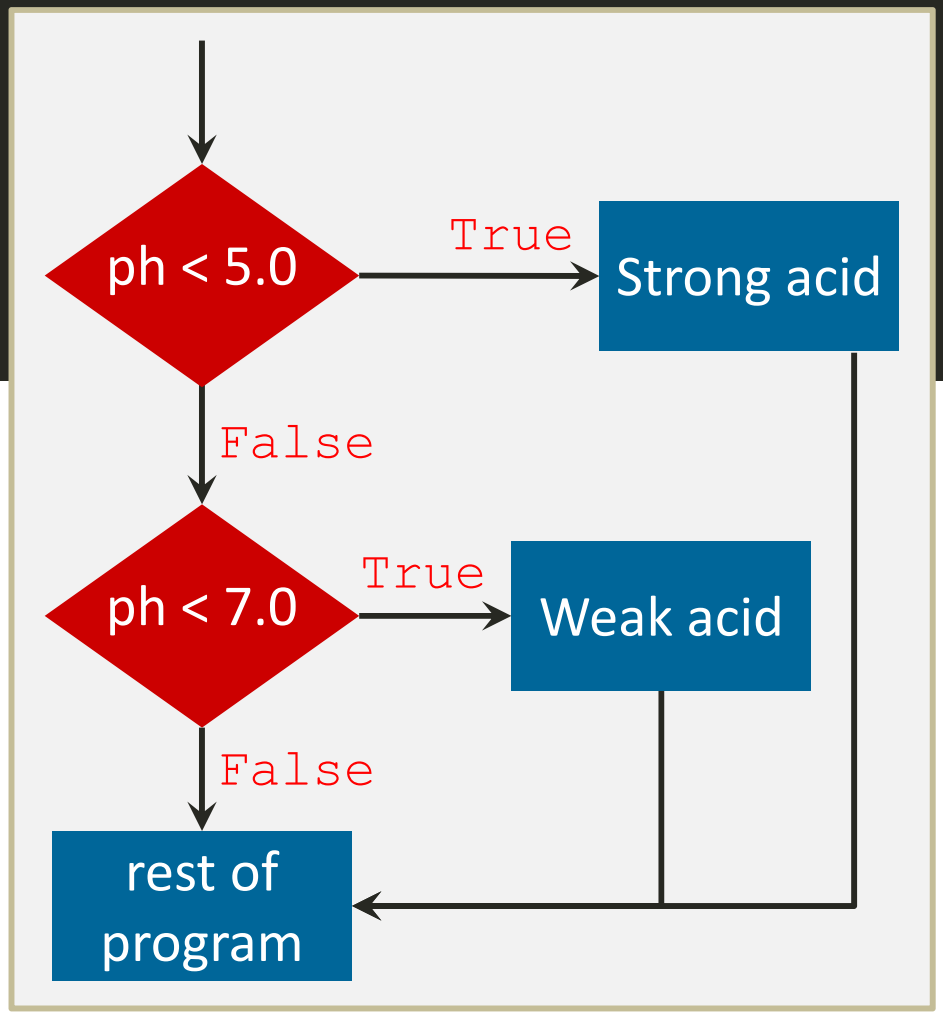
Example Code: Using Nested `if`

```
ph = float(input("Enter a PH value: "))
if 0.0 <= ph:
    if ph < 5.0:
        print("Strong acid")
if 5.0 <= ph:
    if ph < 7.0:
        print("Weak acid")
```



Example Code: Using elif

```
1  ph = float(input("Enter a PH value: "))
2  if ph < 5.0:
3      print("Strong acid")
4  elif ph < 7.0:
5      print("Weak acid")
```



If the condition of the `if` is `true`, then neither the `elif` nor its block is executed!

`elif` Clauses

- A condition-and-block pair is called a **clause**
- `elif` is for "else if"
- General form:

```
if ⟨condition1⟩:
    ⟨block1⟩
elif ⟨condition2⟩:
    ⟨block2⟩
```

indentation

```
if ⟨condition3⟩:
    ⟨block3⟩
if ⟨condition4⟩:
    ⟨block4⟩
```

indentation

  - If the ⟨**condition2**⟩ is True, then the ⟨**block2**⟩ is executed;
  - if the ⟨**condition2**⟩ is False, then it is skipped
- The ⟨**block**⟩ must be **indented**
- An `elif` may be preceded by other `elif`,
  and the top one of these must be preceded by an `if`
- An `if` may be followed by any number (including 0) of `elif`

What will these programs print?

- Or there are some errors?

```
x, y = 7, 16
if x < 5:
    print("hello1")
    if y > 8:
      print("hello2")
elif 5 <= x < 10:
      print("hello3")
      if y > 16:
          print("hello4")
elif x >= 10:
    print("hello5")
print("hello6")
```

```
x, y = 2, 5
if x < 5:
    print("hello1")
    elif y > 2:
        print("hello2")
    if y > 8:
        print("hello3")
if x >= 5:
    print("hello4")
elif y > 16:
    print("hello5")
print("hello6")
```

```
x, y, z = 3, 7, 11
if x < 5:
    print("hello1")
    if y > 8:
        print("hello2")
    elif y <= 8:
        print("hello3")
elif z >= 10:
    print("hello4")
    elif y > 16:
        print("hello5")
print("hello6")
```

hello3
hello6

```
  File "xxx.py", line 4
    elif y > 2:
       ^
SyntaxError: invalid syntax
```

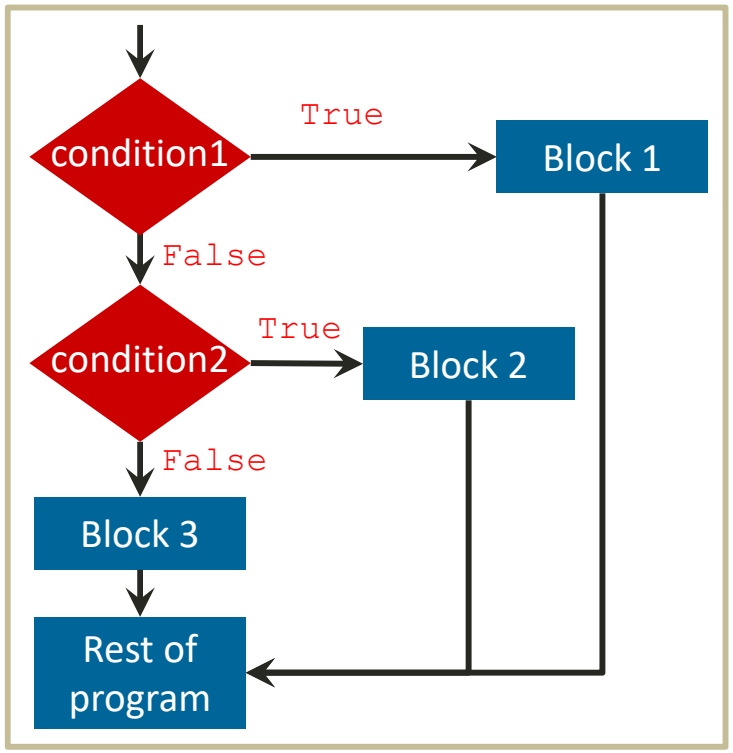
```
  File "xxx.py", line 10
    elif y > 16:
       ^
SyntaxError: invalid syntax
```

else Clause

- General form:

```
if ⟨condition1⟩:
    ⟨block1⟩
elif ⟨condition2⟩:
    ⟨block2⟩
else:
    ⟨block3⟩
```

indentation



- An `if` statement can have at most one else clause
  - An `if` statement can have neither `elif` nor `else`
- The `else` clause must be the final clause
- Note that:

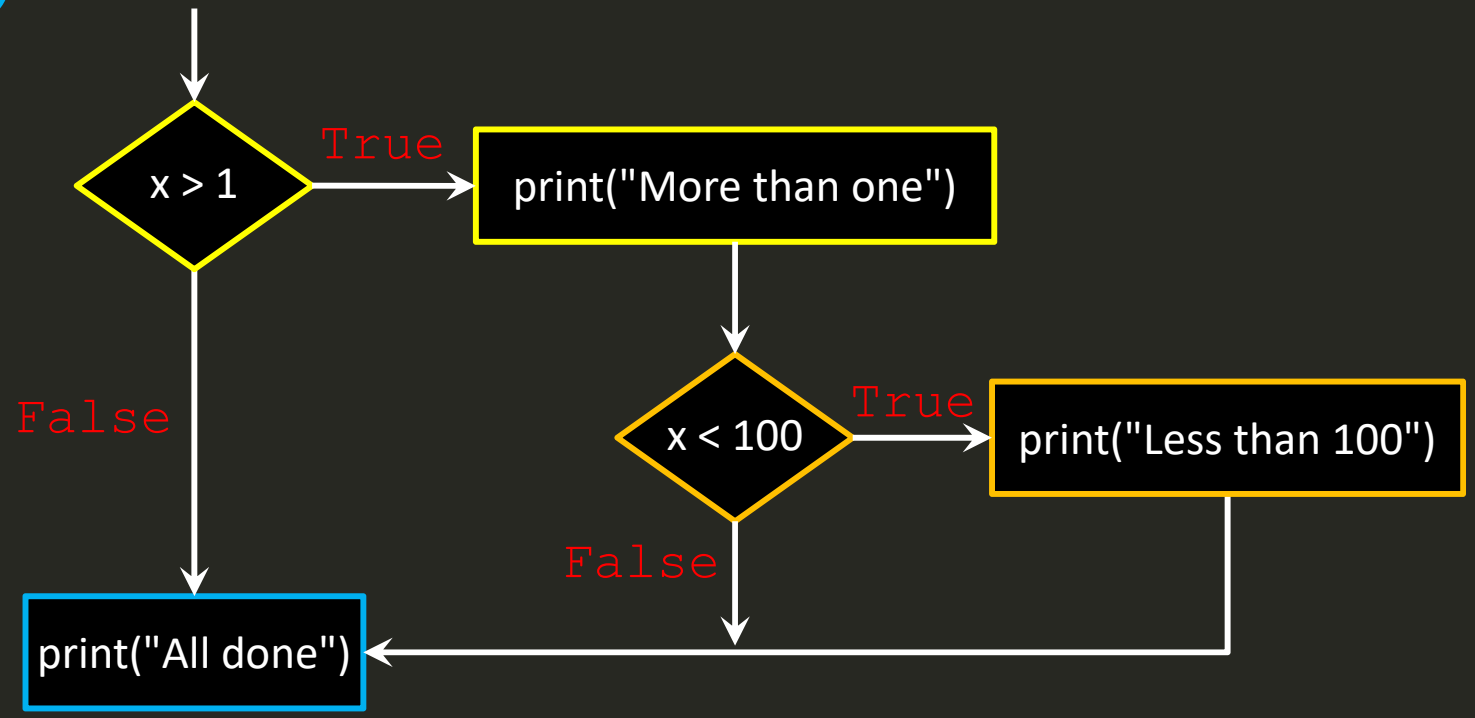
```
if ⟨condition1⟩:
    ⟨if block⟩
else:
    ⟨else block⟩
```

is equivalent to

```
if ⟨condition1⟩:
    ⟨if block⟩
if not ⟨condition1⟩:
    ⟨else block⟩
```

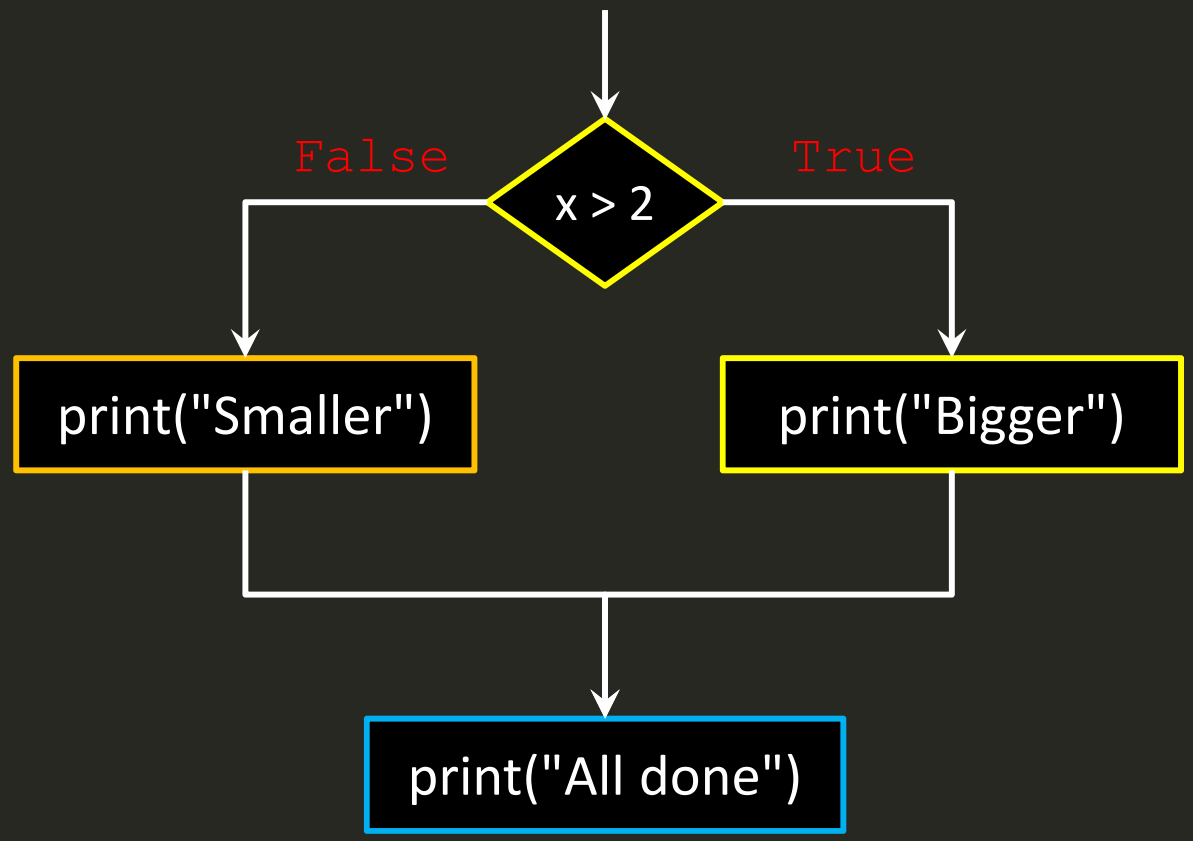
Nested Decisions

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



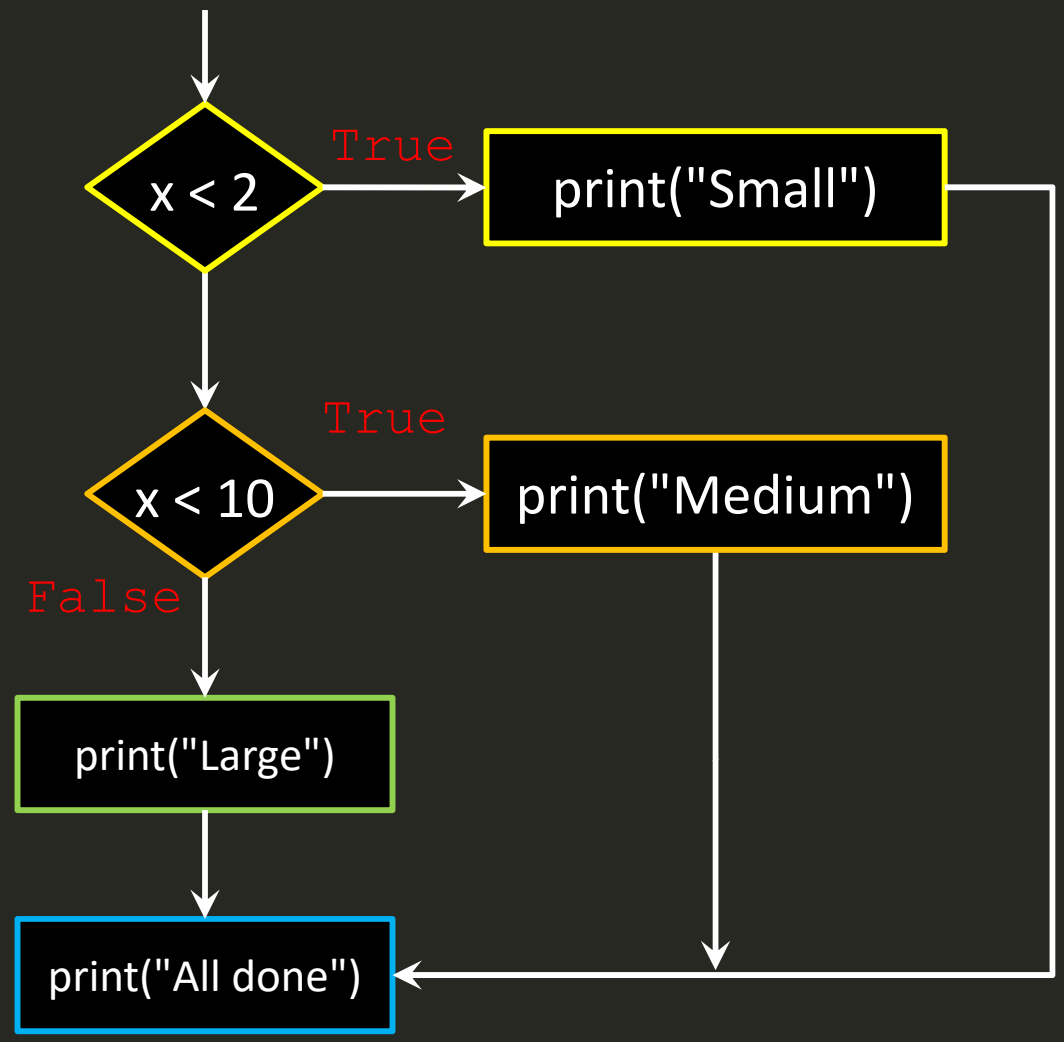
Two-way Decisions

```
x = 4
if x > 2:
    print("Bigger")
else:
    print("Smaller")
print("All done")
```



Multi-way Decisions

```
x = 0
if x < 2:
    print("Small")
elif x < 10:
    print("Medium")
else:
    print("LARGE")
print("All done")
```



Multi-way Puzzles

- What will never print?

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

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

Indentation

- **Increase** indent after an `if` statement or `while` and `for` loop statements (after **:**)
- **Maintain** indent to indicate the scope of the **block** (which lines are affected by the `if`/`for`)
- **Reduce** indent *back to* the level of the `if` statement or `while`/`for` statement to indicate the end of the block
- Blank lines are ignored - they do not affect indentation
- Comments on a line by themselves are ignored with regard to indentation

Generally we use TAB for indentation!

increase / maintain indent after if or for
decrease indent 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):
    # here is in a for loop
    print(i)
    if i > 2:
        print("Bigger than 2")

    print("Done with i", i)
print("All Done")
```

Think about Begin/End of blocks

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

for i in range(5):
    # here is in a for loop
    print(i)
    if i > 2:
        print("Bigger than 2")

    print("Done with i", i)
print("All Done")
```

block

block

block

In-Class Exercise

```python
myinput = input("pH value: ")
if len(myinput) > 0:
    ph = float(myinput)
    if ph < 0.0 or ph > 14.0:
        print("Invalid pH value")
    elif 0 <= ph < 7.0:
        print("Acidic")
    elif 7.0 < ph <= 14.0:
        print("Basic")
    else:
        print("Neutral")
else:
    print("No pH value given")
```

Your task: modify the code to fit this table

pH Level	Solution Category
0 <= pH < 5	Strong acid
5 <= pH < 7	Weak acid
7 <= pH < 8	Neutral
8 <= pH < 10	Weak base
10 <= pH < 15	Strong base

In-Class Exercises

- Write a program that convert temperatures (Celsius/Fahrenheit) selected and entered by users

$$°C = (°F - 32) \times 5 / 9$$

$$°F = (°C \times 9 / 5) + 32$$

```python
# Temperature Conversion Program (Celsius-Fahrenheit / Fahrenheit-Celsius)

# Display program welcome
print("This program will convert temperatures (Celsius/Fahrenheit)")
print("Enter (F) to convert Fahrenheit to Celsius")
print("Enter (C) to convert Celsius to Fahrenheit")

# Get temperature to convert
which = input("Enter selection: ")
temp = int(input("Enter temperature to convert: "))

# Determine temperature conversion needed and display results
if which == "F":
    converted_temp = (temp - 32) * 5.0 / 9.0
    print(temp, "degrees Fahrenheit equals", converted_temp, "degrees Celsius")
else:
    if which == "C":
        converted_temp = (9.0 / 5.0 * temp) + 32
        print(temp, "degrees Celsius equals", converted_temp, "degrees Fahrenheit")
    else:
        print("INVALID INPUT!")
```

In-Class Exercise: Risk of Heart Disease

- Write a program (bmiage.py) that allows users to
  - Input his/her BMI value and age
  - Print out his/her risk of heart disease based on the following table

BMI	Age < 45	Age ≥ 45
< 22	Low	Medium
≥ 22	Medium	High

Figure: Risk of heart disease, based on age and body mass index

```
c:\Python35-32\workspace>python bmiage.py
Enter your age: 30
Enter your BMI: 25
Your risk of heart disease is medium
```

```
c:\Python35-32\workspace>python bmiage.py
Enter your age: 50
Enter your BMI: 50
Your risk of heart disease is high
```

In-Class Exercise: Risk of Heart Disease

```python
age = int(input("Enter your age: "))
bmi = float(input("Enter your BMI: "))
if age < 45:
    if bmi < 22.0:
        risk = 'low'
    else:
        risk = 'medium'
else:
    if bmi < 22.0:
        risk = 'medium'
    else:
        risk = 'high'
print("Your risk of heart disease is", risk)
```

BMI	Age	
	< 45	≥ 45
< 22	Low	Medium
≥ 22	Medium	High

Alternative Solution with Stored Conditionals

```python
age = float(input("Your Age: "))
bmi = float(input("Your BMI: "))
young = age < 45
slim = bmi < 22.0
if young and slim:
    risk = 'low'
elif young and not slim:
    risk = 'medium'
elif not young and slim:
    risk = 'medium'
elif not young and not slim:
    risk = 'high'
print(risk)
```

BMI	Age	
	< 45	≥ 45
< 22	Low	Medium
≥ 22	Medium	High

Stored Conditionals as List Indices

```python
age = float(input("Your Age: "))
bmi = float(input("Your BMI: "))
table = [['medium', 'high'], ['low', 'medium']]
young, heavy = age < 45, bmi >= 22.0
risk = table[young][heavy]
print(risk)
```

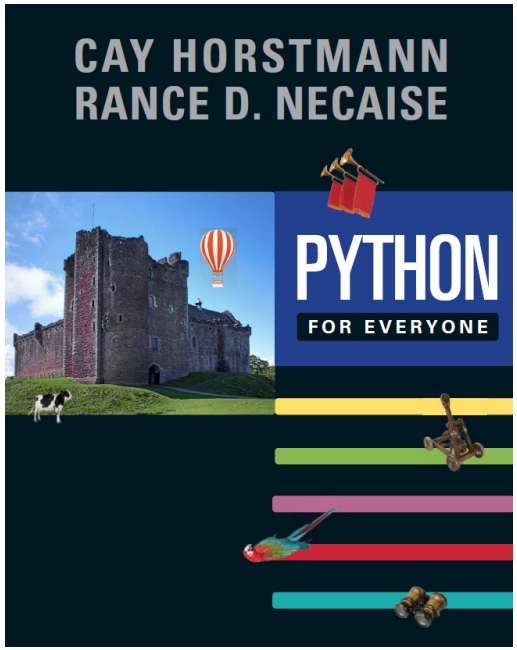
	0	1
0	'medium'	'high'
1	'low'	'medium'

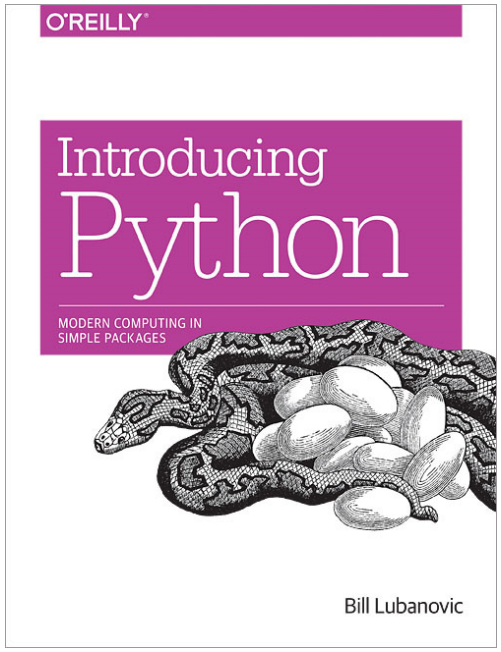
BMI	Age < 45	Age ≥ 45
< 22	Low	Medium
≥ 22	Medium	High

Summary

- Logic Operator
- Relational Operator
- `if`
- `elseif`
- `else`
- Nested, Two-way, Multi-way choices

Suggested Reading





P.91 – P.135

P.69 – P.74