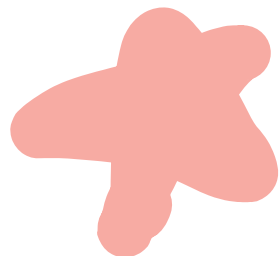




香港中文大學(深圳)

The Chinese University of Hong Kong, Shenzhen



# Introduction to Computer Science: Programming Methodology

## Lecture 3 Flow Control

Prof. Pinjia He  
School of Data Science

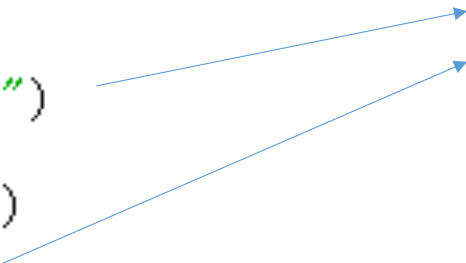
# Conditional flow 条件流.

## Program

```
x=5
if x<10:
    print("smaller")
if x>20:
    print("bigger")
print("finished")
```

## Outputs

```
smaller
finished
>>> |
```



# Comparison operators

布尔表达式.

- **Boolean expressions** ask a question and produce a **Yes/No** result, which we use to **control program flow**
- **Boolean expressions** use **comparison operators** to evaluate Yes/No or True/False
- **Comparison operators** check variables but do not change the values of variables
- **Careful!!** "=" is used for assignment

赋值.

if

$x < y$	Is x less than y?
$x \leq y$	Is x less than or equal to y?
$x == y$	Is x equal to y?
$x \geq y$	Is x greater than or equal to y?
$x > y$	Is x greater than y?
$x != y$	Is x not equal to y?

# Comparison operators

`x = 5`

`x == 5`

# Comparison operators

```
x=5
if x==5:
    print("Equals 5")

if x>4:
    print("Greater than 4")

if x>=5:
    print("Greater than or equal to 5")

if x<=5:
    print("Less than or equal 5")

if x!=6:
    print("Not equal 6")
```



```
Equals 5
Greater than 4
Greater than or equal to 5
Less than or equal 5
Not equal 6
```

# Examples of comparison

```
>>> 5 > 7                                # Is 5 greater than 7?
False
>>> x, y = 45, -3.0
>>> x > y                                # Is 45 greater than -3.0?
True
>>> result = x > y + 50                  # Is 45 greater than -3.0 + 50?
>>> result
False
>>> if 1 + 1 > 1:
...     print("I think this should print.")
...
I think this should print.
>>> "hello" > "Bye"                      # Comparison of strings.
True
>>> "AAB" > "AAC"
False
```

★ - 12-12 bit.  
(ASCII code)

# Examples of comparison

```
>>> "hello" > "Bye"      # Comparison of strings.  
True  
>>> "AAB" > "AAC"  
False
```

# ASCII Code

一种计算机编码系统，用于表示文本中的字符，包括字母、数字和符号。

ASCII printable characters		
32	space	64 @
33	!	65 A
34	"	66 B
35	#	67 C
36	\$	68 D
37	%	69 E
38	&	70 F
39	'	71 G
40	(	72 H
41	)	73 I
42	*	74 J
43	+	75 K
44	,	76 L
45	-	77 M
46	.	78 N
47	/	79 O
48	0	80 P
49	1	81 Q
50	2	82 R
51	3	83 S
52	4	84 T
53	5	85 U
54	6	86 V
55	7	87 W
56	8	88 X
57	9	89 Y
58	:	90 Z
59	;	91 [
60	<	92 \
61	=	93 ]
62	>	94 ^
63	?	95 _
		96 `
		97 a
		98 b
		99 c
		100 d
		101 e
		102 f
		103 g
		104 h
		105 i
		106 j
		107 k
		108 l
		109 m
		110 n
		111 o
		112 p
		113 q
		114 r
		115 s
		116 t
		117 u
		118 v
		119 w
		120 x
		121 y
		122 z
		123 {
		124
		125 }
		126 ~



# Examples of comparison

```
>>> "hello" > "Bye"      # Comparison of strings.  
True  
>>> "AAB" > "AAC"  
False
```

↓

$h > B : 104 > 66.$   
 $e > y : 101 > 121.$   
⋮

# Examples of comparison

对比.

```
>>> 7 == 7.0
```

```
True
```

```
>>> x = 0.1
```

```
>>> 1 == 10 * x
```

```
True
```

```
>>> 1 == x + x + x + x + x + x + x + x + x + x + x
```

```
False
```

```
>>> x + x + x + x + x + x + x + x + x + x + x
```

```
0.9999999999999999
```

```
>>> 7 != "7"
```

```
True
```

```
>>> 'A' == 65
```

```
False
```

原因: python用二进制, 不是所有

小数可被准确

表达.

字符串和数字不能比

# Floating point math

浮点数是近似值.

- Floating point numbers are **approximated**

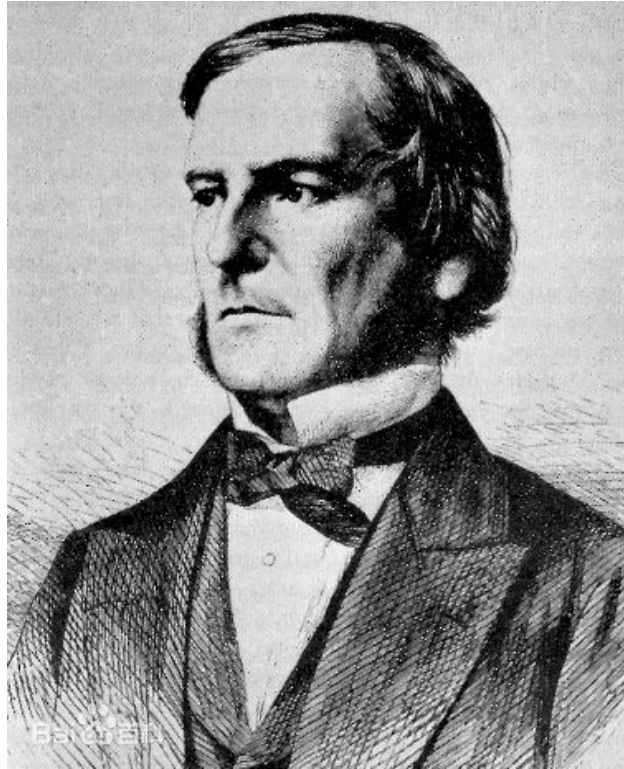
- Rational numbers (such as 0.1, which is  $1/10$ ) whose denominator is not a power of two cannot be exactly represented

分母不是2的幂不能精确表示.

- 0.100000000000000000000555111512312578270211815834045  
41015625

# Boolean type *True/False*

- Python contains a built-in **Boolean type**, which takes two values **True/False** 



**George Boole** (1815 - 1864): Mathematician, inventor of mathematical logic, significant contributions to differential and difference equations

# Boolean type

- Python contains a built-in **Boolean type**, which takes two values **True/False**

/ 空集.

- Number 0 can also be used to represent **False**. All other numbers represent **True**

and: 所有为 true 才是 true

or: - 一个为 true 就是 true.

# Bool()

```
>>> x = 0; y = 0.0; z = 0 + 0j
>>> bool(x), bool(y), bool(z)
(False, False, False)
>>> x = -1; y = 1.e-10; z = 0 + 1j
>>> bool(x), bool(y), bool(z)
(True, True, True)
>>> x = []; y = [0]; z = "0"
>>> bool(x), bool(y), bool(z)
(False, True, True)
```

$x = []$  指的是空集  
/ 0 (0.0, 0.0...)

非空集才是true.

# One way decisions 单向决策.

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

print(' Afterwards 5')

print(' Before 6')
if x==6:
    print(' Is 6')
    print(' Is still 6')
    print(' Third 6')

print(' Afterwards 6')
```



```
Before 5
Is 5
Is still 5
Third 5
Afterwards 5
Before 6
Afterwards 6
```



# Indentation 缩进

- **Increase indent**: indent after an if or for statement (after :)
- **Maintain indent**: to indicate the **scope** of the block (which lines are affected by the if/for) 指示块的范围.
- **Decrease indent**: to back to the level of the if statement or for

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

print('Afterwards 5')
```

# Indentation

- Blank lines are ignored – they do not affect indentation
- Comments on a line by themselves are ignored w.r.t. indentation

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

print('Afterwards 5')
```

# Increase/maintain/decrease

- Increase/maintain after if/for statements

- Decrease to indicate the end of a block

- Blank lines and comments are ignored

空隔行将会被忽略。

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

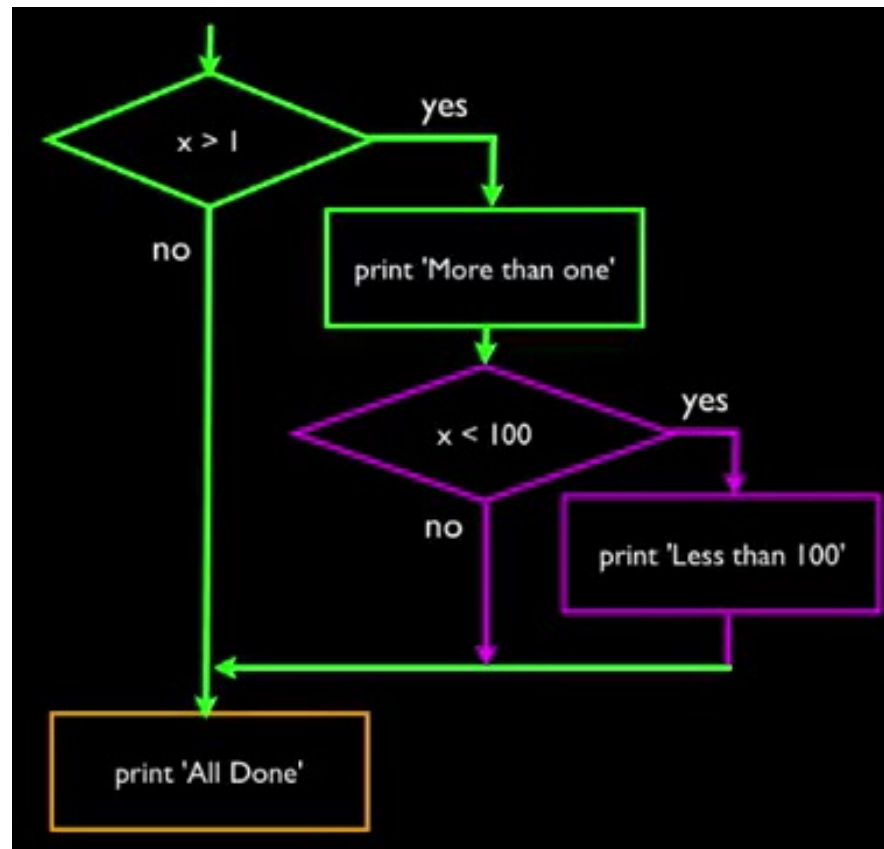
print('Before 6')
if x==6:
    print('Is 6')
    print('Is still 6')
    print('Third 6')
print('Afterwards 6')
```

嵌套决策.

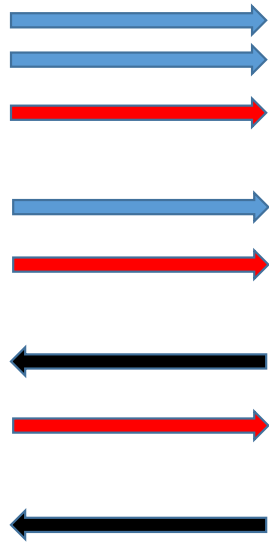
# Nested decisions

## Example

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



# Mental begin/end



```
x=10
if x>5:
    print('Greater than 5')

    if x>8:
        print('Greater than 8')

        if x>10:
            print('Greater than 10')

print('Finished')
```

# Too many nested decisions will be a disaster...

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

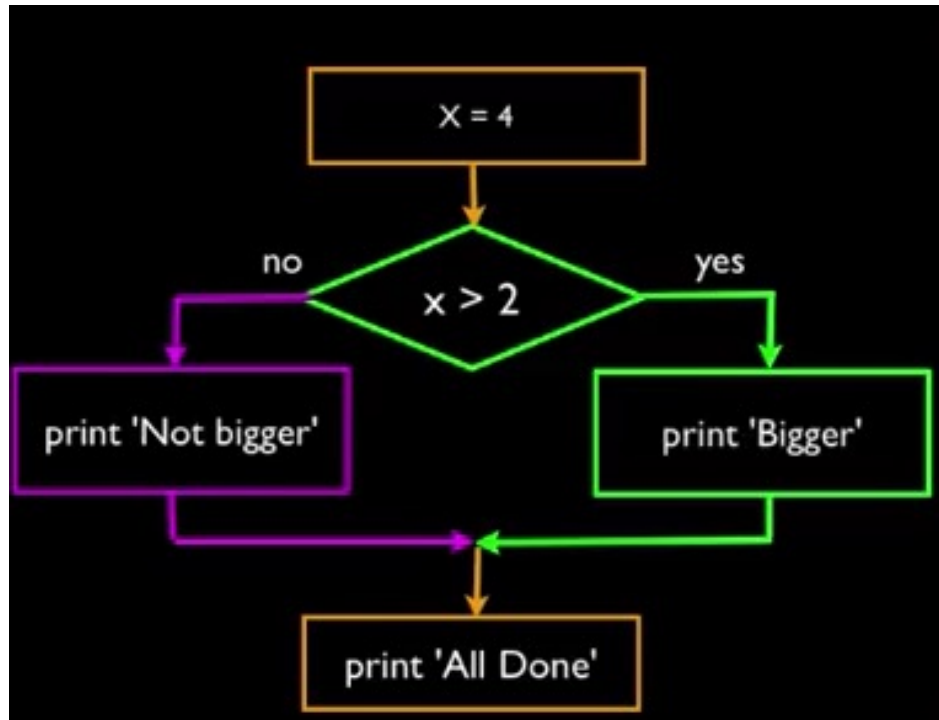


**C#论坛**  
www.ibcibc.com

# Two way decisions

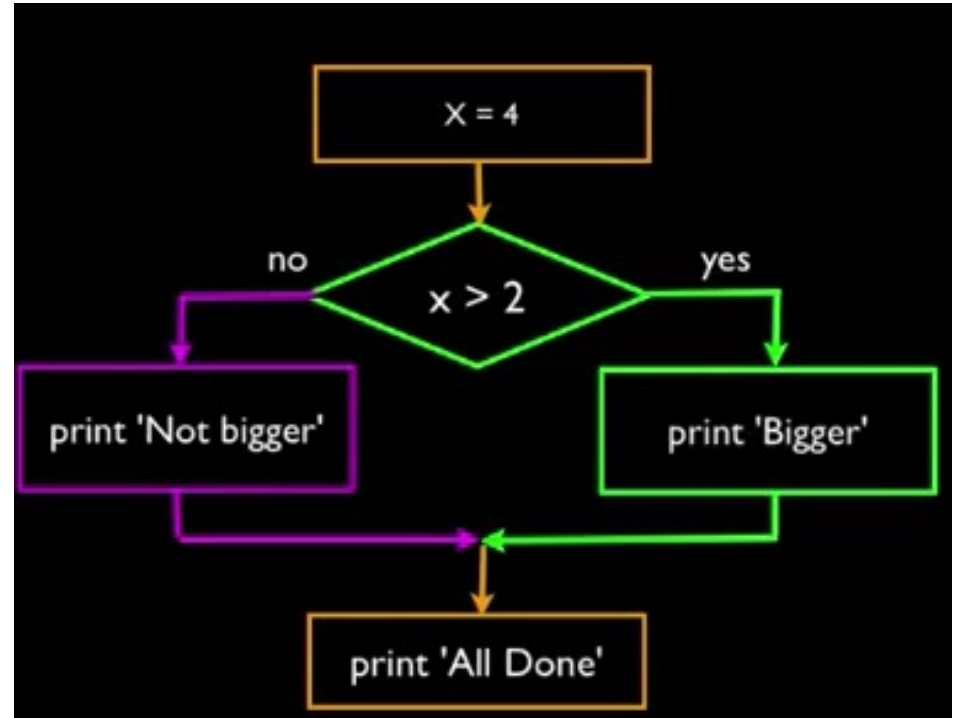
- Sometimes we want to do one thing when the logical expression is true, and another thing when it is false
- It is like a fork in the road, we need to choose one or the other path, but not both

岔路口。



# Two way decision using else

```
x=1  
  
if x>2:  
    print(' Bigger' )  
else:  
    print(' Smaller' )  
  
print(' Finished' )
```





## Tips on if - else

☆☆ if, else 必须对齐.

```
x=1
```

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

```
print('Finished')
```



```
x=1
```

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

```
print('Finished')
```



- Else must come after if
- Use indentation to match if and else

缩进.

# Example

```
x=1  
  
if x>2:  
    if x>5:  
        print('Bigger than 5')  
    else:  
        print('Smaller than 5')  
print('Finished')
```

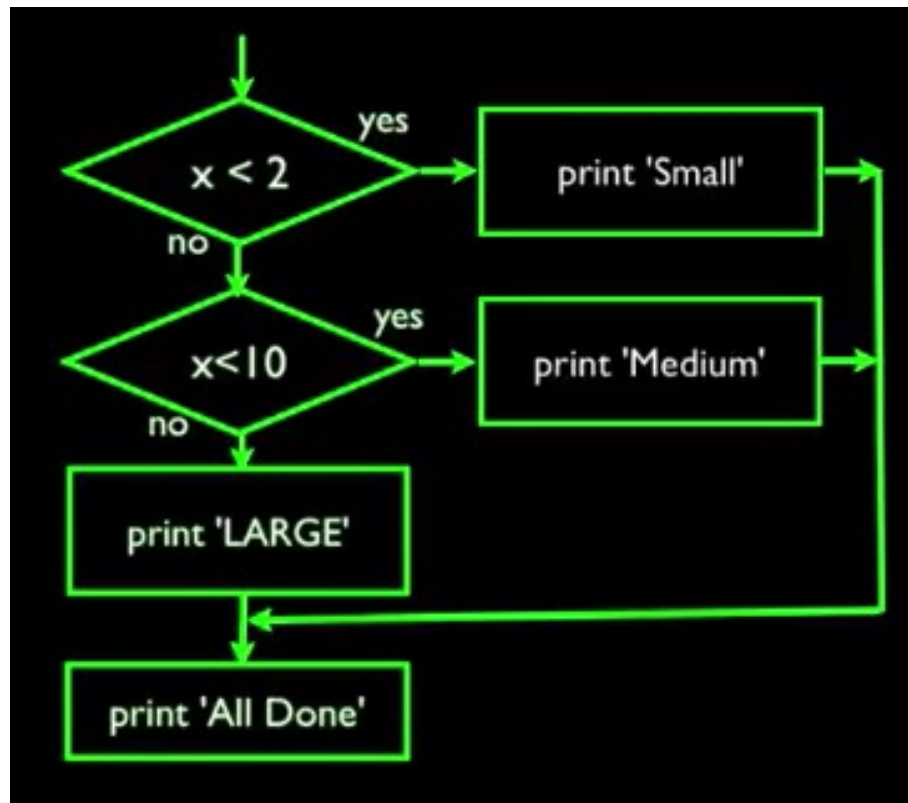
多路分支.

# Multi-way decisions

elif: else if-

```
x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')
else:
    print('Large')

print('Finished')
```



即若条件1判断不成功，继续判断

条件2，成功则不会继续

# Multi-way decision

条件1和2; 条件1和2成功时不执行  
判断2.

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

Medium  
Finished.

# Multi-way decision

```
x=56
if x<2:
    print('Small')
elif x<10:
    print('Medium')
elif x<20:
    print('Large')
elif x<40:
    print('Huge')
else:
    print('Ginormous')

print('Finished')
```

Ginormous  
Finished.

# Which will never be printed?

x=4

```
if x<=2:
    print('Below 2')
elif x>2:
    print('Above 2')
else:
    print('Something else')
print('Finished')
```

x=8

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

# Logical operators

- Logical operators can be used to combine several logical expressions into a single expression

将逻辑表达式转换为单个表达式。

- Python has three logical operators: not, and, or

## Example

```
>>> not True
```

```
False
```

```
>>> False and True
```

```
False
```

```
>>> not False and True
```

```
True
```

```
>>> (not False) and True
```

```
True
```

```
>>> True or False
```

```
True
```

and: 2个t才t

or: 1个t就t.

=

# Same as previous statement.



# Example

```
>>> not False or True
```

```
True
```

# Same as: (not False) or True.

```
>>> not (False or True) ≠ not True.
```

```
False
```

```
>>> (False and False) or True # Same as: (False and False) or True.
```

```
True
```

```
>>> False and (False or True)
```

```
False
```

# Try/except structure

- You surround a dangerous part of code with **try/except**

try代码有效, 则跳过 except 块.

- If the code in try block **works**, the except block is **skipped**

- If the code in try block **fails**, the except block will be **executed**

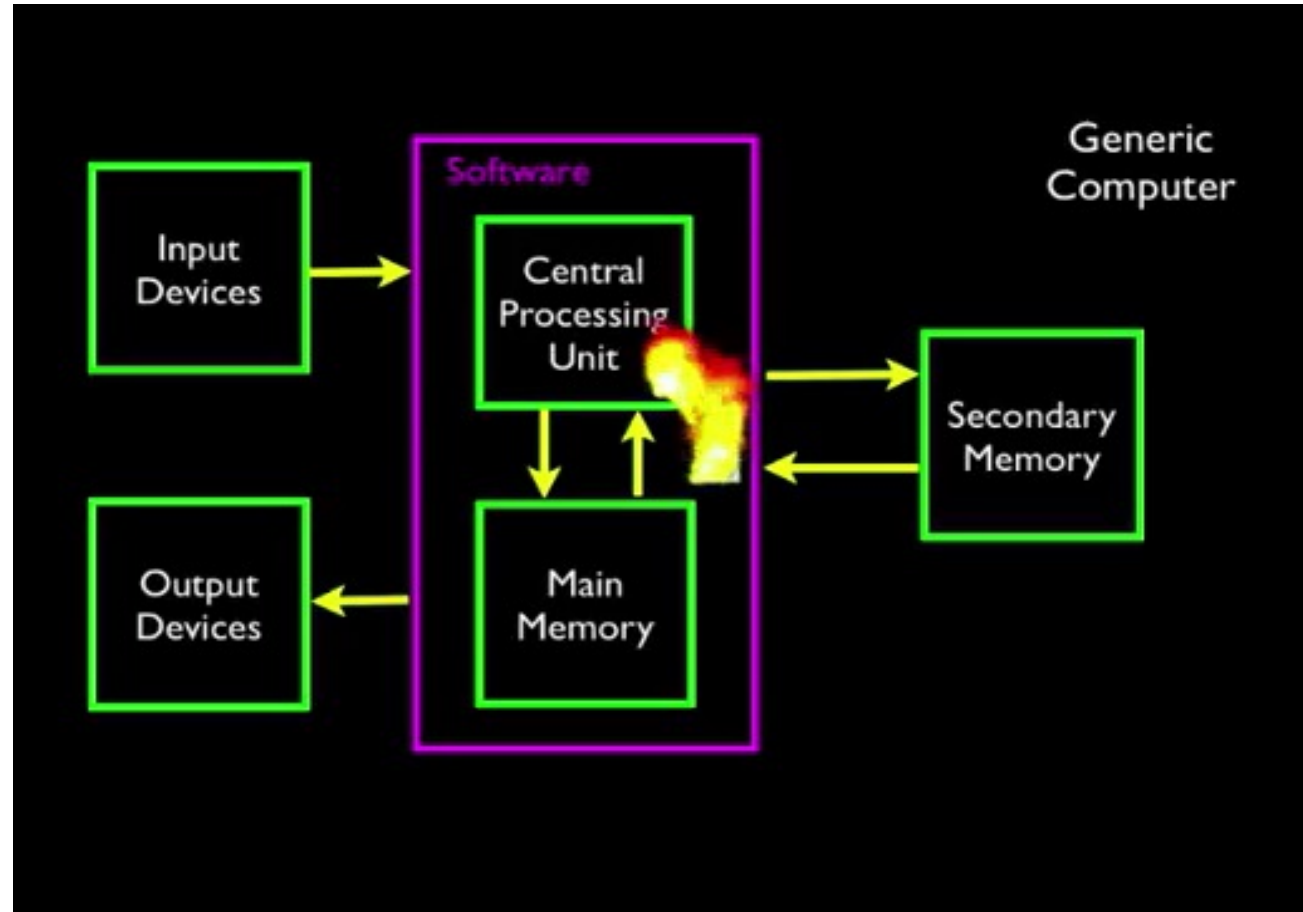


执行.

## Example

```
astr = 'Hello bob'  
istr = int(astr)  
print('First', istr)
```

```
astr = '123'  
istr = int(astr)  
print('Second', istr)
```



# Use try/except to **capture errors**

```
astr = 'Hello bob'
```

```
try:
```

```
    istr = int(astr)
```

```
except:
```

```
    istr = -1
```

```
print('First', istr)
```

```
astr = '123'
```

```
try:
```

```
    istr = int(astr)
```

```
except:
```

```
    istr = -1
```

```
print('Second', istr)
```

fail

- When the first conversion **fails**, it just **stops into the except block**, and the program continues

→ First, -1

- When the second conversion **succeeds**, it just **skips the except block**

Second, 123.

# Try/except

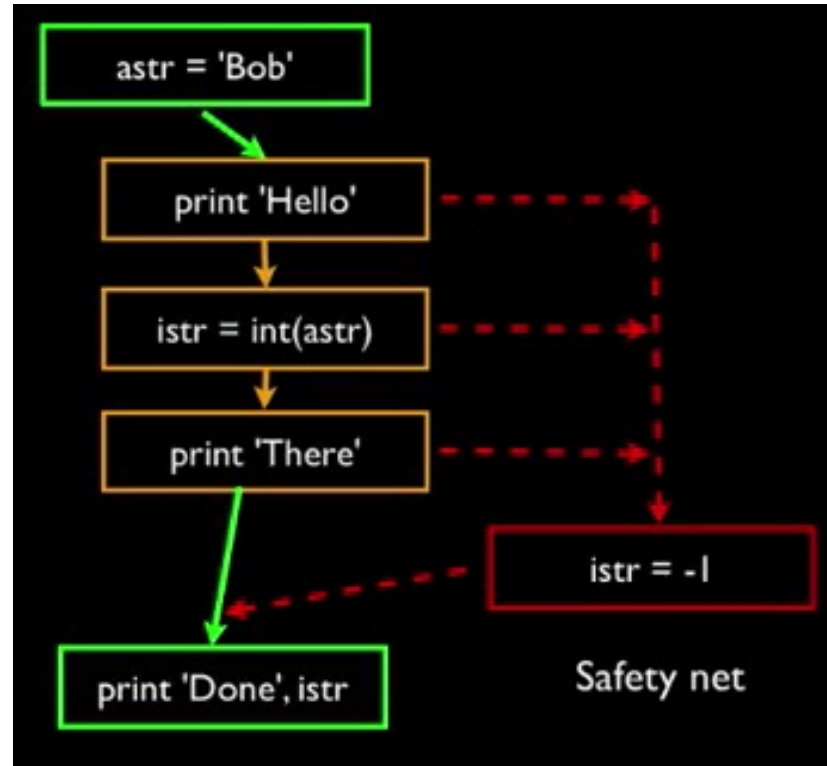
```
astr = 'Bob'
try:
    print('Hello')
    istr = int(astr)
    print('There')
except:
    istr = -1
    print('Done', istr)
```

X

✓

✓

Hello  
Done, -1.



# Example

string.

```
rawstr = input('Enter a number:')  
try:  
    ival = int(rawstr) X  
except:  
    ival = -1 ✓  
  
if ival > 0:  
    print('Nice work') X  
else:  
    print('Invalid number') ✓
```

Invalid number.

# Practice

- Write a program to instruct the user to input the working hours and hourly rate, and then output the salary. If the working hours exceed 40 hours, then the extra hours received 1.5 times pay.



# Updates

- The course will be mixed-mode, starting next week
- Office hour: 10-11 am Thu., Rm. 420b, Daoyuan Building

## ★ Practice

- ✓ Write a program to instruct a user to input a date (both month and day), and then output the new month and day when the inputted date is advanced by one day (leap years are ignored)

△ 规则不清楚.

~~Answer~~

%: 求余数.

#Add a day to a given date

month = int(input('Enter a month (1-12):'))

day = int(input('Enter a day (1-31):'))

daysInMonth = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

if day < daysInMonth[month-1]:

print(month, day+1)

else:

month = month%12 + 1

print(month, 1)

→ 回归 tuple.

☆ 这样12月不用另起一类分析.

//: 何取整; %: 取余数.

#每个月的天数

# Repeated flow

## Program

```
n=5  
while n>0:  
    print(n)  
    n = n - 1  
print("Finish")
```

缩进 →

循环

## Outputs

```
5  
4  
3  
2  
1  
Finish  
>>>
```

- Loops (repeated steps) have **iterative variables** that change each time through a loop
- Often these iterative variables go through **a sequence of numbers**

# ● An infinite loop

知道while-个死循环。

n=5

```
while n>0:  
    print('Lather')  
    print('Rinse')
```

永远运行下去。

n=n-1

```
print('Dry off!')
```

缩进。

- What is wrong with this program?

# Another loop

```
n=0  
while n>0:  
    print('Lather')  
    print('Rinse')  
    n=n-1  
    print('Dry off!')
```

不実行。  
Dry off!

- What is wrong with this program?

# Breaking out of a loop

结束循环.

- The break statement ends the current loop, and jumps to the statement which directly follows the loop

```
while (True):  
    line = input('Enter a word:')  
    if line == 'done':  
        break  
    print(line)  
print('Finished')
```

即: 若输入 done  
input 将变为  
'Finished'.  
不会 print 出 line.

完成迭代

# Finishing an iteration with continue

重新开始

```
while True:
    line = input('Input a word:')
    if line[0] == '#': continue
    if line == 'done':
        break
    print(line)
    print('Done')
```

[0]: 第一位  
[1]: 第二位  
[2]: 第三位  
:

直接进行 print

结束当前循环

- The **continue** statement ends the current iteration, and **start the next iteration immediately**



# Indefinite loop 无限循环

- **While** loops are called “indefinite loops”, since they keep going until a logical condition becomes false
- Till now, the loops we have seen are relatively easy to check whether they will terminate 终止.
- Sometimes it can be hard to determine whether a loop will terminate

# Definite loop

- Quite often we have a finite set of items 有限.
- We can use a loop, each iteration of which will be executed for each item in the set, using the for statement
- These loops are called “definite loops” because they execute an exact number of times
- It is said that “definite loops iterate through the members of a set”

# A simple for loop

## Example

```
for i in [5, 4, 3, 2, 1]:  
    print(i)  
print('Finished')
```

## Output

```
5  
4  
3  
2  
1  
Finished
```

# For loop

Example

```
for i in [5, 4, 3, 2, 1]:  
    print(i)  
print('Finished')
```

Output

```
5  
4  
3  
2  
1  
Finished
```

迭代变量.

- For loops (definite loops) have explicit iteration variables that change each time through a loop.
- These iteration variables move through a sequence or a set

# In

- The iteration variable “**iterates**” through a **sequence** (ordered set)
- The block (body) of the code is executed once for each value in the sequence
- The **iteration variable** moves through all of the values in the sequence

迭代变量

Iteration variable

Sequence with  
five elements

```
for i in [5, 4, 3, 2, 1]:  
    print(i)
```

# Another example

## Example

```
friends = ['Tom', 'Jerry', 'Bat']  
for friend in friends:  
    print('Happy new year', friend)  
print('Done')
```

## Output

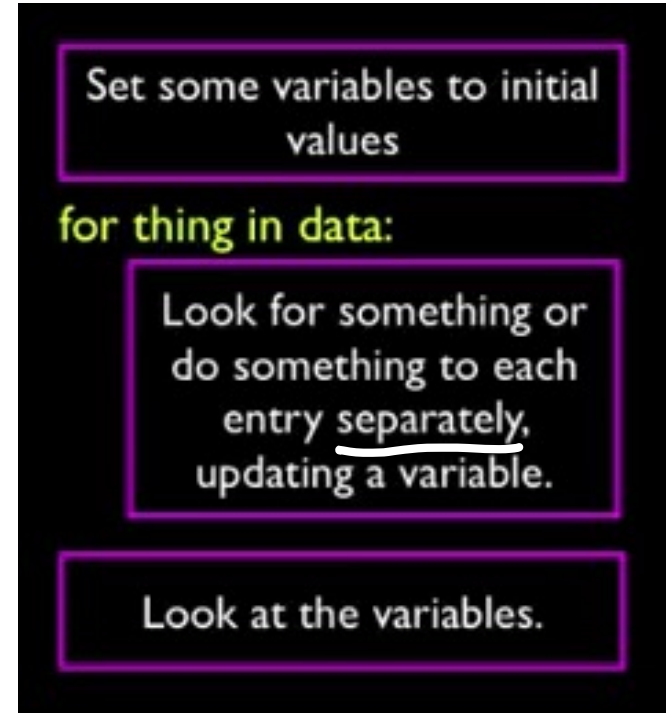
```
Happy new year Tom  
Happy new year Jerry  
Happy new year Bat  
Done
```

# Loop samples

- Note: though these examples are simple, the patterns apply to all kinds of loops

# Making “smart” loops

- The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time





# Looping through a set

循环遍历集合。

## Example

```
print('Before')
for thing in [3, 5, 100, 34, 6, 87]:
    print(thing)
print('After')
```

## Output

```
Before
3
5
100
34
6
87
After
```

# Finding the largest number

```
25  1  114 117 150 152 120  46  19 126
191 121 104 116 160 105  89 125  40  14
 31 139 113  94  97 193 154 140 195 122
112 163 177  48  78 101 130  83  35 197
 44  54 106 143  59  38  3  41  93  81
 20 164  4  11 131  0 107  71 159  69
181 178 173 148  62 142 170  72  37 145
 60 187 198  99  15  82  26  8 192  17
129  73  45  9  24 188  42 151  51 183
179  79  50  76  34  33 185 102 193 184
```

- Use a **variable** to store the largest number we have seen so far
- If the current number is **larger**, we assign it to the store variable

# Counting in a loop

## Example

```
count = 0
print('Before', count)
for thing in [3, 4, 98, 38, 9, 10, 199, 78]:
    count = count + 1
    print(count, thing)
print('After', count)
```

tuple  
循环最后一个变量值。

## Output

```
Before 0
1 3
2 4
3 98
4 38
5 9
6 10
7 199
8 78
After 8
```

- To count how many times we have executed a loop, we can introduce a counting variable, which increases itself in each iteration

## Practice

- Given a set of numbers, write a program to calculate their sum using for loop

```
sum = 0
print("Before", sum)
for thing in [...]:
    sum = sum + thing
print(sum)
print("After", sum)
```

## Answer

```
numberSet = [3, 4, 98, 38, 9, 10, 199, 78]
```

```
total = 0
```

```
print('Before', total)
```

```
for num in numberSet:
```

```
    total = total + num
```

```
    print(total, num)
```

```
print('Last', total)
```

→ update  
the value  
of total.

Before 0

3 3

7 4

105 98

143 38

152 9

162 10

361 199

439 78

Last 439

# Practice

- Given a set of numbers, write a program to calculate their average using for loop

count, sum.

# Answer

```
numberSet = [3, 4, 98, 38, 9, 10, 199, 78]  
total = 0  
count = 0  
print('Before', total)  
for num in numberSet:  
    total = total + num  
    count = count + 1  
    print(count, total, num)  
print('Last', total, total/count)
```

```
Before 0  
1 3 3  
2 7 4  
3 105 98  
4 143 38  
5 152 9  
6 162 10  
7 361 199  
8 439 78  
Last 439 54.875
```

过滤.

# Filtering in a loop

## Example

```
print('Before')  
for value in [23, 3, 43, 39, 80, 111, 99, 3, 65]:  
    if value > 50:  
        print('Large value:', value)  
print('After')
```

## Output

```
Before  
Large value: 80  
Large value: 111  
Large value: 99  
Large value: 65  
After
```

- We can use an **if** statement in a loop to **catch/filter** the values we are interested at



# ☆ Search using a Boolean variable

## Example

```
found = False
print('Before', found)

for value in [9, 41, 12, 3, 74, 15]:
    if value == 74:
        found = True
    print(found, value)
print('After', found)
```

## Output

```
Before False
False 9
False 41
False 12
False 3
True 74
True 15
After True
```

每一个都要  
循环

也就是 found  
已经被赋值  
为 True 了。

上一步已经有 found = True.

- If we want to search in a set and double check whether a specific number is in that set
- We can use a Boolean variable, set it to False at the beginning, and assign True to it as long as the target number is found

# Finding the largest number

## Example

```
largest_so_far = -1
print('Before', largest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if num > largest_so_far:
        largest_so_far = num
        print(largest_so_far, num)

print('After', largest_so_far)
```

为这个新数  
赋值。

## Output

```
Before -1
9 9
39 39
39 21
98 98
98 4
98 5
100 100
100 65
After 100
```

- Use a variable to store the largest number we have seen so far
- If the current number is larger, we assign it to the store variable





# Finding the smallest number

```
smallest_so_far = -1  
print('Before', smallest_so_far)  
  
for num in [9, 39, 21, 98, 4, 5, 100, 65]:  
    if num < smallest_so_far:  
        smallest_so_far = num  
    print(smallest_so_far, num)  
  
print('After', smallest_so_far)
```

○ 設置大小

- Use a **variable** to store the smallest number we have seen so far
- If the current number is **smaller**, we assign it to the store variable
- What is the problem with this program?

○

找不到的

★ 注意 True, False, None 均有字母大写!

## Finding the smallest number

Example

防止没的数比数集  
中的小.

```
smallest_so_far = None  
print('Before', smallest_so_far)
```

```
for num in [9, 39, 21, 98, 4, 5, 100, 65]:  
    if smallest_so_far == None: 一个循环.  
        smallest_so_far = num  
    elif num < smallest_so_far: 但 None 这-  
        smallest_so_far = num 步又进行  
    print(smallest_so_far, num) 第一次.
```

```
print('After', smallest_so_far)
```

Output

Before None

9 9  
9 39  
9 21  
9 98  
4 4  
4 5  
4 100  
4 65

After 4

- We still use a variable to store the **smallest value seen so far**
- In the first iteration, the smallest value is **none**, so we need to use an **if** statement to check this

# The **is** and **is not** operator



```
smallest_so_far = None
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if smallest_so_far is None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

- Python has a “is” operator which can be used in logical expression
- Implies “is the same as”
- Similar to, but stronger than ==
- “is not” is also an operator



# Is operator

## Example

✓ `print(10 is 10)`

✓ `a = 10`  
`b = 10`  
`print(a is b)`

✓ `a = '123'`  
`b = '123'`  
`print(a is b)` 地址相同.

✗ `a = [1, 2, 3]`  
`b = [1, 2, 3]`  
`print(a is b)`  
地址不同.

## Output

True  
True  
True  
False

(a == b) ✓.  
value 同.

tips: id( )

a = [1, 2, 3]

b = [1, 2, 3]

print( id(a) ) > 地址不同

print( id(b) )

print( a is b ) ⇒ False.