



**Multi-dimensional View of Python**

---

# Python面面观

---

Dazhuang@NJU

Department of Computer Science and Technology  
Department of University Basic Computer Teaching



用Python玩转数据

条件

# if 语句

## 语法

```
if expression :  
    expr_true_suite
```

## expression

条件表达式:

- 比较运算符
- 成员运算符
- 逻辑运算符

## expr\_true\_suite

- expression 条件为 True 时执行的代码块
- 代码块必须缩进（通常为4个空格）

**F**<sub>ile</sub>

# Filename: ifpro.py

sd1 = 3

sd2 = 3

if sd1 == sd2:

print("the square's area is", sd1\*sd2)

# else 语句

## 语法

if expression :

    expr\_true\_suite

else:

    expr\_false\_suite

expr\_false\_suite

- expression 条件为 False 时执行的代码块
- 代码块必须缩进
- else 语句不缩进

File

# Filename: elsepro.py

sd1 = int(input("the first side: "))

sd2 = int(input("the second side: "))

if sd1 == sd2:

    print("the square's area is", sd1\*sd2)

else:

    print("the rectangle's area is", sd1\*sd2)

Input and Output

the first side: 4

the second side: 4

the square's area is 16

# elif 语句

## 语 法

```
if expression :  
    expr_true_suite  
elif expression2:  
    expr2_true_suite  
    :  
    :  
elif expressionN :  
    exprN_true_suite  
else:  
    none_of_the_above_suite
```

## expr2\_true\_suite

- expression2为True时执行的代码块

## exprN\_true\_suite

- expressionN 为 True 时执行的代码块

## else

- none\_of\_the\_above\_suite是以上所有条件都不满足时执行的代码块

# elif 语句

6

File

```
# Filename: elifpro.py
k = input('input the index of shape: ')
if k == '1':
    print('circle')
elif k == '2':
    print('oval')
elif k == '3':
    print('rectangle')
elif k == '4':
    print('triangle')
else:
    print('you input the invalid number')
```

Input and

Output

input the index of shape: 3  
rectangle

Input and

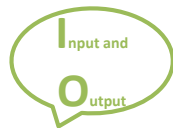
Output

input the index of shape: 8  
you input the invalid number

# 条件嵌套

7

- 同等缩进为同一条件结构



input the index of shape: 3  
the first side: 3  
the second side: 4  
the rectangle's area is 12



input the index of shape: 2  
oval



```
# Filename: ifnestpro.py
k = input('input the index of shape: ')
if k == '1':
    print('circle')
elif k == '2':
    print('oval')
elif k == '3':
    sd1 = int(input('the first side: '))
    sd2 = int(input('the second side: '))
    if sd1 == sd2:
        print('the square's area is', sd1*sd2)
    else:
        print('the rectangle's area is', sd1*sd2)
elif k == '4':
    print('triangle')
else:
    print('you input the invalid number')
```

# 猜数字游戏

- 程序随机产生一个0~300间的整数，玩家竞猜，系统给出“猜中”、“太大了”或“太小了”的提示。

File

```
# Filename: guessnum1.py
from random import randint

x = randint(0, 300)
digit = int(input('Please input a number between 0~300: '))
if digit == x:
    print('Bingo!')
elif digit > x:
    print('Too large, please try again.')
else:
    print('Too small, please try again.')
```



用Python玩转数据

# RANGE函数

# range()

## 语法

`range (start, end, step=1)`

`range (start, end)`

`range (end)`

- 产生一系列整数，返回一个range对象

 Source

```
>>> list(range(3,11,2))
```

```
[3, 5, 7, 9]
```

```
>>> list(range(3,11))
```

```
[3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> list(range(11))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

## start

- 起始值 (包含)

## end

- 终值 (不包含)

## step

- 步长 (不能为0)

`range (start, end, step=1)`

- 不包含end的值

`range (start, end)`

- 缺省step值为1

`range (end)`

- 缺省了start值为0, step为1

# range()

11

异同	range()	xrange()
语法	基本一致	
返回	列表	生成器（类似）
生成	真实列表	用多少生成多少

Python 2.x

异同	range()
语法	与Python 2.x中类似
返回	生成器（类似）
生成	用多少生成多少

Python 3.x



用Python玩转数据

# 循环

# while 循环

## 语法

**while** expression:

    suite\_to\_repeat

expression

- 条件表达式
- 当expression值为True时执行suite\_to\_repeat代码块



```
>>> sumA = 0
>>> j = 1
>>> while j < 10:
        sumA += j
        j += 1
>>> sumA
45
>>> j
10
```

# for 循环 (一)

## 语 法

```
for iter_var in iterable_object:  
    suite_to_repeat
```

## 可以明确循环的次数

- 遍历一个数据集内的成员
- 在列表解析中使用
- 生成器表达式中使用

## iterable\_object

- String
- List
- Tuple
- Dictionary
- File

## for 循环 (二)

- range()返回的是一个 *iterable\_object*
- 字符串也是一个 *iterable\_object*



```
>>> for i in range(3, 11, 2):  
        print(i, end = ' ')
```

```
3 5 7 9
```

```
>>> s = 'Python'
```

```
>>> for c in s:  
        print(c, end = ' ')
```

```
P y t h o n
```

```
>>> for i in range(len(s)):  
        print(s[i], end = ' ')
```

```
P y t h o n
```

# 猜数字游戏

- 程序随机产生一个0~300间的整数，玩家竞猜，允许猜多次，系统给出“猜中”、“太大了”或“太小了”的提示。

File

```
# Filename: guessnum2.py
from random import randint

x = randint(0, 300)
for count in range(5):
    digit = int(input('Please input a number between 0~300: '))
    if digit == x :
        print('Bingo!')
    elif digit > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.')
```



用Python玩转数据

# 循环中的

**BREAK, CONTINUE 和 ELSE**



# break 语句

- break语句终止当前循环，转而执行循环之后的语句

F<sub>ile</sub>

```
# Filename: breakpro.py
```

```
sumA = 0
```

```
i = 1
```

```
while True:
```

```
    sumA += i
```

```
    i += 1
```

```
    if sumA > 10:
```

```
        break
```

```
print('i={}, sum={}'.format(i, sumA))
```

Output:

i=6, sum=15

# while 循环和break

- 输出2-100之间的素数

Output:

2 3 5 7 11 13 17 19  
23 29 31 37 41 43  
47 53 59 61 67 71  
73 79 83 89 97

File

```
# Filename: prime.py
from math import sqrt
j = 2
while j <= 100:
    i = 2
    k = sqrt(j)
    while i <= k:
        if j%i == 0: break
        i += 1
    if i > k:
        print(j, end = ' ')
    j += 1
```

# for循环和break

- 输出2-100之间的素数

Output:

```
2 3 5 7 11 13 17 19
23 29 31 37 41 43
47 53 59 61 67 71
73 79 83 89 97
```

F<sub>ile</sub>

```
# Filename: prime.py
from math import sqrt
for i in range(2, 101):
    k = int(sqrt(i))
    for j in range(2, k+1):
        if i%j == 0:
            flag = 0
            break
    if ( flag ):
        print(i, end = ' ')
```

## continue语句

- 在while和for循环中，continue语句的作用：
  - 停止当前循环，重新进入循环
  - while循环则判断循环条件是否满足
  - for循环则判断迭代是否已经结束

# continue语句

- 循环中的break:



File

# Filename: breakpro.py

sumA = 0

i = 1

while i <= 5:

    sumA += i

    if i == 3:

        break

    print('i={},sum={}'.format(i, sumA))

    i += 1

- 循环中的continue:



File

# Filename: continuepro.py

sumA = 0

i = 1

while i <= 5:

    sumA += i

    i += 1

    if i == 3:

        continue

    print('i={},sum={}'.format(i, sumA))

# 猜数字游戏（想停就停，非固定次数）

- 程序随机产生一个0~300间的整数，玩家竞猜，允许玩家自己控制游戏次数，如果猜中系统给出提示并退出程序，如果猜错给出“太大了”或“太小了”的提示，如果不想继续玩可以退出并说再见。

File

```
# Filename: guessnum3.py
from random import randint
x = randint(0, 300)
go = 'y'
while go == 'y':
    digit = int(input('Please input a number between 0~300: '))
    if digit == x:
        print('Bingo!')
        break
    elif digit > x:
        print('Too large, please try again.')
    else:
        print('Too small, please try again.')
    print('Input y if you want to continue.')
    go = input()
else:
    print('Goodbye!')
```

# 循环中的else语句

- 循环中的else:
  - 如果循环代码从break处终止, 跳出循环
  - 正常结束循环, 则执行else中代码



```
# Filename: prime.py
from math import sqrt
num = int(input('Please enter a number: '))
j = 2
while j <= int(sqrt(num)):
    if num % j == 0:
        print('{:d} is not a prime.'.format(num))
        break
    j += 1
else:
    print('{:d} is a prime.'.format(num))
```





用Python玩转数据

# 自定义函数

内置  
函数

函数调用之前必须先定义

自定义  
函数

# 自定义函数的创建

## 语法

```
def function_name([arguments]):
```

```
    "optional documentation string"
```

```
    function_suite
```

S<sub>ource</sub>

```
>>> def addMe2Me(x):
```

```
        'apply operation + to argument'
```

```
        return x+x
```

# 自定义函数的调用

- 函数名加上函数运算符，一对小括号
  - 括号之间是所有可选的参数
  - 即使没有参数，小括号也不能省略

Source

```
>>> addMe2Me()
```

Traceback (most recent call last):

```
File "<pyshell#6>", line 1, in <module>
    addMe2Me()
```

TypeError: addMe2Me() takes exactly 1 argument (0 given)

Source

```
>>> addMe2Me(3.7)
```

```
7.4
```

```
>>> addMe2Me(5)
```

```
10
```

```
>>> addMe2Me('Python')
'PythonPython'
```

- 输出1-100之间的素数

Output:


2 3 5 7 11 13 17 19  
23 29 31 37 41 43  
47 53 59 61 67 71  
73 79 83 89 97

File

```
# Filename: prime.py
from math import sqrt
def isprime(x):
    if x == 1:
        return False
    k = int(sqrt(x))
    for j in range(2,k+1):
        if x%j == 0:
            return False
    return True
for i in range(2,101):
    if isprime(i):
        print( i, end = ' ')
```

# 默认参数（一）

- 函数的参数可以有一个默认值，如果提供有默认值，在函数定义中，默认参数以赋值语句的形式提供



```
>>> def f(x = True):  
    """whether x is a correct word or not"""  
    if x:  
        print('x is a correct word')  
        print('OK')  
  
>>> f()  
x is a correct word  
OK  
  
>>> f(False)  
OK
```


## 默认参数（二）

- 默认参数的值可以改变



```
>>> def f(x, y = True):  
    """x and y both correct words or not """  
    if y:  
        print(x, 'and y both correct')  
        print(x, 'is OK')  
  
>>> f(68)  
68 and y both correct  
68 is OK  
  
>>> f(68, False)  
68 is OK
```

- 默认参数一般需要放置在参数列表的最后




```
def f(y = True, x):  
    '''x and y both correct words or not'''  
    if y:  
        print(x, 'and y both correct')  
        print(x, 'is OK')
```

SyntaxError: non-default argument follows default argument



# 关键字参数

- 关键字参数是让调用者通过使用参数名区分参数。允许改变参数列表中的参数顺序

 Source

```
>>> def f(x, y):  
    '''x and y both correct words or not'''  
    if y:  
        print(x, 'and y both correct')  
        print(x, 'is OK')  
>>> f(68, False)  
68 is OK  
>>> f(y = False, x = 68)  
68 is OK  
>>> f(y = False, 68)  
SyntaxError: non-keyword arg after keyword arg  
>>> f(x = 68, False)  
SyntaxError: non-keyword arg after keyword arg
```


# 传递函数

- 函数可以像参数一样传递给另外一个函数




```
>>> def addMe2Me(x):  
    return x+x  
  
>>> def self(f, y):  
    print(f(y))  
  
>>> self(addMe2Me, 2.2)  
4.4
```

- 匿名函数



```
>>> def addMe2Me(x):  
    'apply operation + to argument'  
    return x + x  
>>> addMe2Me(5)  
10
```



```
>>> r = lambda x : x + x  
>>> r(5)  
10
```

# lambda函数

```
def my_add(x, y) : return x + y
```

```
lambda x, y : x + y
```

```
my_add = lambda x, y : x + y
```

```
>>> my_add(3, 5)  
8
```



用Python玩转数据

# 递归



循环

生成斐波那契数列的方法

递归

递归是最能表现计算思维的算法之一

# 循环和递归

- 递归必须要有边界条件，即停止递归的条件
  - $n == 0$  or  $n == 1$
- 递归的代码更简洁，更符合自然逻辑，更容易理解

Source

# the nth Fibonacci number

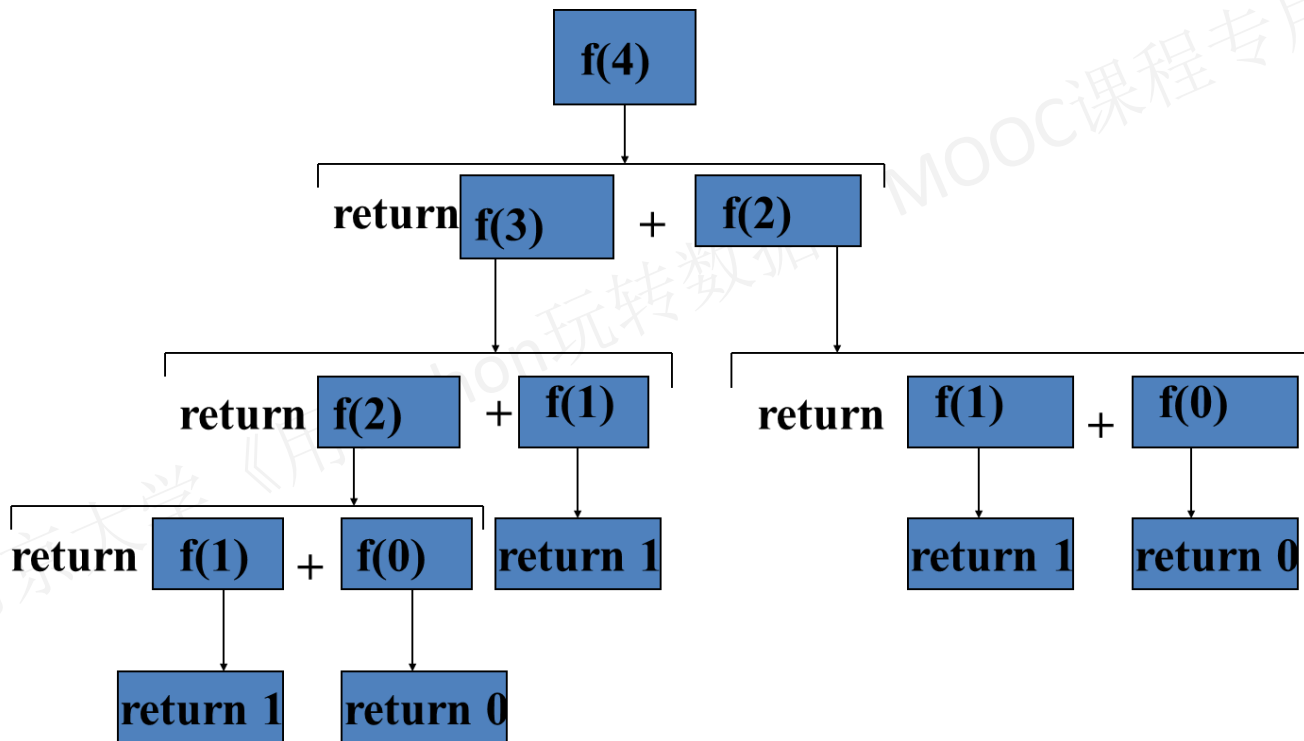
```
def fib(n):  
    a, b = 0, 1  
    count = 1  
    while count < n:  
        a, b = b, a+b  
        count = count + 1  
    print(b)
```

Source

# the nth Fibonacci number

```
def fib(n):  
    if n == 0 or n == 1:  
        return n  
    else:  
        return fib(n - 1) + fib(n - 2)
```

# 递归





- 递归的执行

01

逐层递归调用

02

遇到边界条件  
停止递归

03

逐层返回调用  
至最初层

系统资源消耗

比循环 **大**

# 汉诺塔

- 汉诺塔游戏

三个塔座A、B、C上各有一根针，通过B把64个盘子从A针移动到C针上，移动时必须遵循下列规则：

- (1) 圆盘可以插入在A、B或C塔座的针上
- (2) 每次只能移动一个圆盘
- (3) 任何时刻都不能将一个较大的圆盘压在较小的圆盘之上



# Filename: Hanoi.py

```
def hanoi(a,b,c,n):
```

```
    if n==1:
```

```
        print(a,'->',c)
```

```
    else:
```

```
        hanoi(a,c,b,n-1)
```

```
        print(a,'->', c)
```

```
        hanoi(b,a,c,n-1)
```

```
hanoi('a','b','c',4)
```

## Output:

a -> b

a -> c

b -> c

a -> b

c -> a

c -> b

a -> b

a -> c

b -> c

b -> a

c -> a

b -> c

a -> b

a -> c

b -> c

## 7

用Python玩转数据

# 变量作用域

# 变量作用域

- 全局变量
- 局部变量



F<sub>ile</sub>

# Filename: global.py

global\_str = 'hello'

def foo():

    local\_str = 'world'

    return global\_str + local\_str



S<sub>ource</sub>

>>> foo()

'helloworld'

- 全局变量和局部变量用同一个名字



```
# Filename: samename.py
```

```
a = 3
```

```
def f( ):
```

```
    a = 5
```

```
    print(a ** 2)
```

# 改变全局变量的值

- 方法是否可行？



# Filename: scopeofvar.py

def f(x):

print(a)

a = 5

print(a + x)

a = 3

f(8)

UnboundLocalError: local variable 'a'  
referenced before assignment

- global语句强调全局变量



# Filename: scopeofvar.py

```
def f(x):  
    global a  
    print(a)  
    a = 5  
    print(a + x)
```

```
a = 3  
f(8)  
print(a)
```

Output:

3  
13  
5

