



Data Processing Using Python

Multi-dimensional View of Python

ZHANG Dazhuang

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

Data Processing Using Python

1

CONDITION

if Statement

Grammar

if **expression** :
 expr_true_suite

expression

Conditional expression :

- Comparison operator
- Member access operator
- Logical operator

expr_true_suite

- If **expression** is true, then execute this block
- There should be indent in front of the block (4 spaces in general)



Filename: ifpro.py

```
sd1 = 3
```

```
sd2 = 3
```

```
if sd1 == sd2:
```

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

else Statement

Grammar

if expression :

 expr_true_suite

else:

 expr_false_suite

expr_false_suite

- If **expression** is False, then execute this
- Code block needs indent
- 'else' has no indent

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 Statement

Grammar

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

expr2_true_suite

- If **expression2** is True, then execute

exprN_true_suite

- If **expressionN** is True, then execute

none_of_the_above_suite

- If all expression above is False, then execute

elif Statement

F_{ile}

```
# 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')
```

I
nput and
O
utput

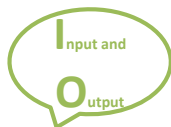
input the index of shape: 3
rectangle

I
nput and
O
utput

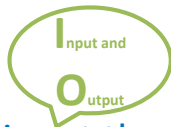
input the index of shape: 8
you input the invalid number

Nesting

- The same indents mean they belong to the same statement block



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')
```

Example: Guess Number

- The program randomly generate an integer between 0 and 300, player inputs a number and system returns a hint from three choices, 'Too large', 'Too small' or 'Bingo'.

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.')
```


Data Processing Using Python

2

RANGE FUNCTION

range()

Grammar

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

`range (start, end)`

`range (end)`

- Generate a series of numbers, and returns a range object



```
>>> 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

- Start value (contained in the range)

end

- End value (not in the range)

step

- Gap between continuous elements

range (start, end, step=1)

- Do not contain the 'end' value

range (start, end)

- Default step is 1

range (end)

- Default start = 0, step = 1

range()

11

| | range() | xrange() |
|------------------|-----------------|---------------------|
| Grammar | Almost the same | |
| Return Value | list | Generator (similar) |
| Generated Result | Real list | Generated when used |

Python 2.x

| | range() |
|------------------|---------------------|
| Grammar | Similar to Python 2 |
| Return Value | Generator (similar) |
| Generated Result | Generated when used |

Python 3.x

Data Processing Using Python

3

LOOP

while loop

Grammar

while expression:

 suite_to_repeat

expression

- Condition expression
- When expression is True, suite_to_repeat is executed.



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

for loop (I)

Grammar

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

Explicitly define loop count


- Scan all member in a dataset
- Used in list comprehension
- Used in generator expressions

iterable_object

- String
- List
- Tuple
- Dictionary
- File

for loop (II)

- String is an *iterable_object*
- The return result of range() is also an *iterable_object*



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

p
y
t
h
o
n

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

3 5 7 9
```

Example: Guess Number with Loop

16

- The program randomly generate an integer between 0 and 300, player inputs a number and system returns a hint from three choices, 'Too large', 'Too small' or 'Bingo'.



```
# 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.')
```



Data Processing Using Python

4

BREAK, CONTINUE & ELSE IN LOOP

break Statement

- **Break** terminates current loop to execute the statement following the loop



```
# 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 loop and break

- Output the prime number in range from 2 to 100

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 = i+1
    if i > k:
        print(j, end = ' ')
    j += 1
```

for loop and break

- Output the prime number in range from 2 to 100

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_{ile}


```
# 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 = ' ')
```

flag = 1

- Continue in **for** loop and **while** loop
 - Stop current loop and reenter the loop
 - If it is **while** loop, then check whether the **expression** is True
 - If it is **for** loop, then check whether the iteration is finished


Continue Statement

- **Break** in loop :



```
# Filename: breakpro.py
sumA = 0
i = 1
while i <= 5:
    sumA += i
    if i == 3:
        break
    print('i={},sum={}'.format(i,sumA))
    i += 1
```

- **Continue** in loop :



```
# Filename: continuepro.py
sumA = 0
i = 1
while i <= 5:
    sumA += i
    i += 1
    if i == 3:
        continue
    print('i={},sum={}'.format(i,sumA))
```

Example: Guess Number with Exit

- The program randomly generate an integer between 0 and 300, player inputs a number to guess.
 - If the guess is True, then system returns 'Bingo' and exit.
 - If the guess is False, then system returns hint 'Too large' or 'Too small'
 - If player decides to exit, then system returns farewell information.



```
# 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()
    print(go)
else:
    print('Goodbye!')
```

Else Statement in loop

- **else** in loop :
 - If loop is stopped by **break**, then jump out of the block
 - If loop stops normally, then execute the code in **else** block



```
# 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))
```


Data Processing Using Python

5

USER-DEFINED FUNCTION

Function

26

**Built-in
Function**

Function should be defined
before calling.

**User-
Defined
function**

Creation of User-Defined Function


27

Grammar

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

```
"optional documentation string"
```

```
function_suite
```



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

Calling of User-Defined Function

- Function Name with Function Operator
(a pair of Parentheses)
 - Content between parentheses are the parameters
 - Function operator () can not be omitted

S
ource

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

Traceback (most recent call last):

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

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

S
ource

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

```
7.4
```

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

```
10
```

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

User-Defined Function

- Output all prime numbers between 0 and 100

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**_{ile}

```
# 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 = ' ')
```

Default Parameter (I)


- Parameter of function can have a default value. If it has one, the default value is presented with the form of assignment statement.



```
>>> 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
```

Default Parameter (II)

- The value of default parameter can be changed


 `>>> 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`

Default Parameter (III)

- Usually, the default parameter should be placed at the end of parameter list.




```
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


Keyword Argument

- Keyword argument helps caller distinguish different parameters by name and allow to change the order of parameter in parameter list.

 `>>> 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`

Use Function as a Parameter

- Function can be passed to another function as a parameter



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

Lambda expressions

- Anonymous function



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



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

Lambda expressions

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



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



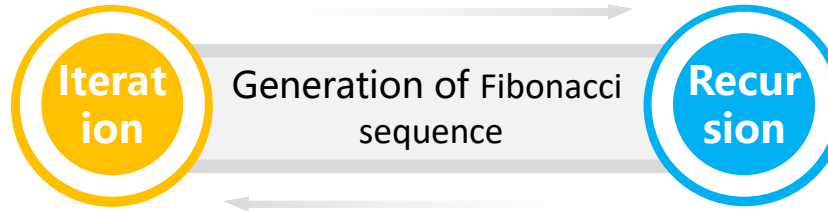
```
my_add = lambda x, y : x + y
```

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

Data Processing Using Python

6 RECURSION


Recursion




Recursion is a good example of computational thinking.

Iteration and Recursion

- Recursion must have a boundary condition to stop execution.
 - $n == 0$ or $n == 1$
- Code of recursion is simpler and more user-friendly.

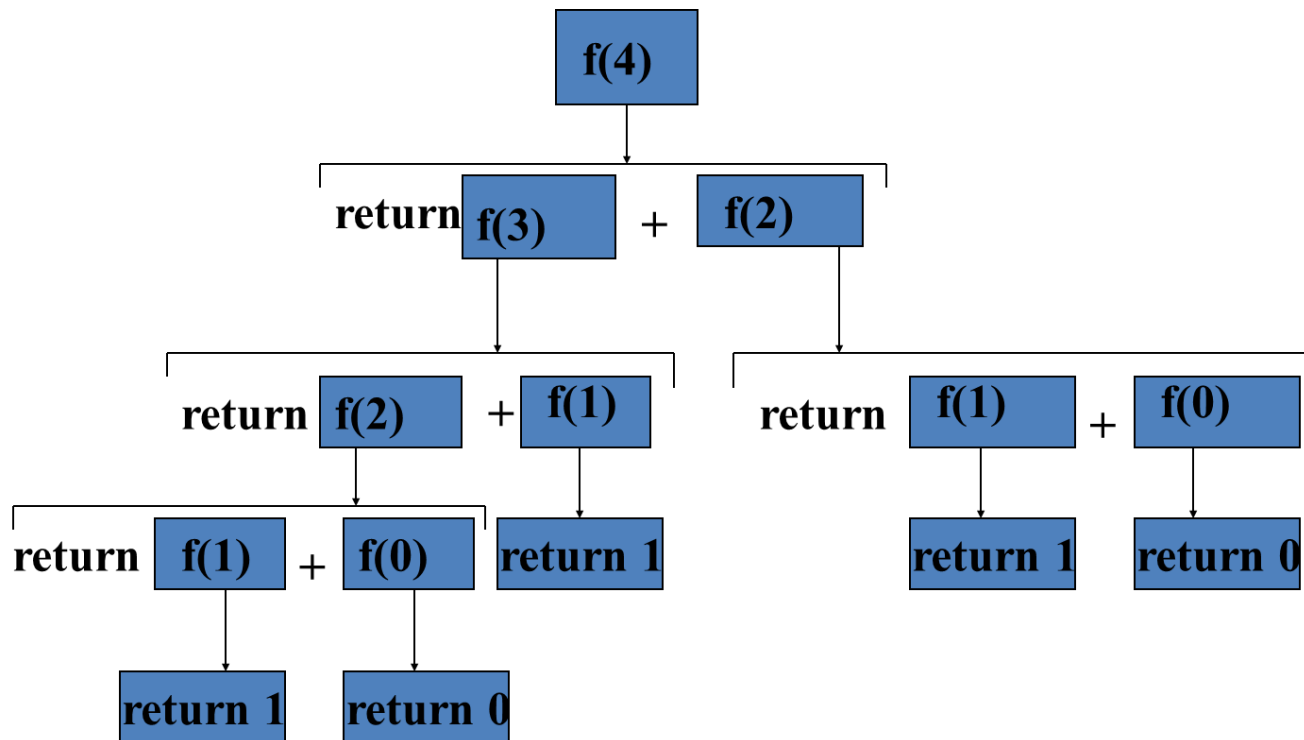
 **S**_{ource}

```
# 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(a)
```

 **S**_{ource}

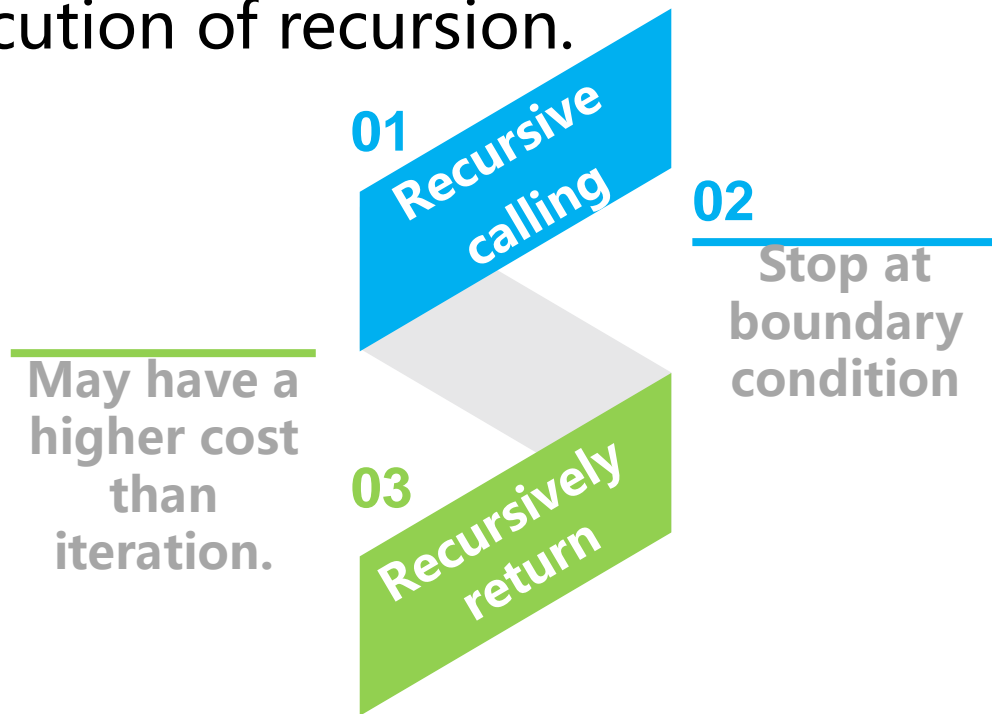
```
# the nth Fibonacci number
def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n - 1) + fib(n - 2)
```

Recursion



Recursion

- The execution of recursion.



Tower of Hanoi

- Tower of Hanoi

It consists of three rods and a number of disks of different sizes, which can slide onto any rod.

The move should obey following rules.

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
- 3, No disk may be placed on top of a smaller disk.



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
```

Data Processing Using Python


7

VARIABLE SCOPE

Scope of Variable

44

- Global variable
- Local variable



A blue oval with a tail pointing to the code block, containing the letter 'F' and the word 'ile'.

```
# Filename: global.py
global_str = 'hello'

def foo():
    local_str = 'world'
    return global_str + local_str
```




An orange oval with a tail pointing to the code block, containing the letter 'S' and the word 'ource'.

```
>>> foo()
'helloworld'
```

Variable with the Same Name

- Global and Local variables have the same name.



```
# Filename: samename.py
a = 3
def f( ):
    a = 5
    print(a ** 2)
```

Change Value of Global Variable

- Is the method practicable?



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** statement marks the identity of global variable



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