



Chap6 Function

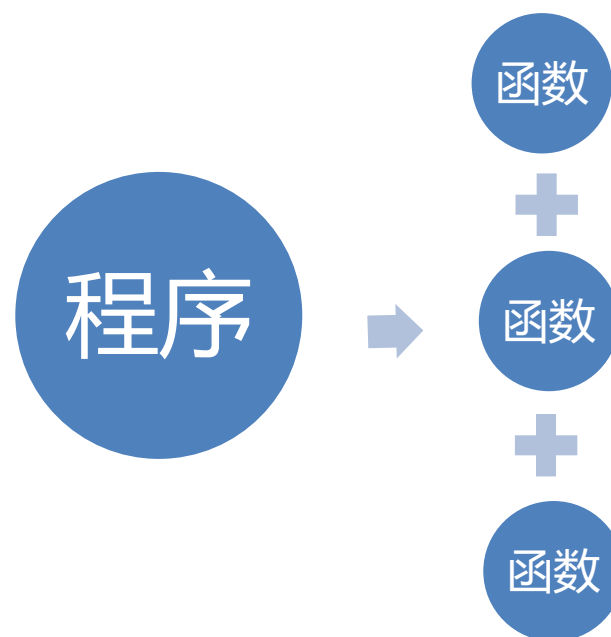
函数

Nanjing University

Department of Computer Science and Technology

Department of University Basic Computer Teaching

- 较大规模的程序通常会被划分成一个个功能模块，这些功能模块就是函数 (function)



6.1

函数的概念

- 函数是一个独立的代码块
- 在解决大规模问题时采用“模块化”策略，将一个大而复杂的原始任务分解为多个较简单的子任务，再为每个简单的子任务设计算法
- 将描述其算法的一组语句封装为一个独立代码块，为每个独立代码块定义一个名字以及能与其他独立代码块通信的接口，这种独立的代码块定义就是函数。

找前5个默尼森数

- P 是素数且 M 也是素数，并且满足等式 $M=2^P-1$ ，则称 M 为默尼森数
- 例如 $P=5$ ， $M=2^P-1=31$ ，5和31都是素数，因此31是默尼森数。

PRIME NUMBERS

2

3

5

7

Any number under 100 which can not be divided by one of the above numbers is prime.

11

13

17

19

Any number under 400 which can not be divided by one of the above numbers is prime.

23

29

31

37

41

43

47

53

59

61

67

71

73

79

83

89

97

Any number under 10,000 which can not be divided by one of the above numbers is prime.

101

103

107

109

113

127

131

137

139

149

151

157

163

167

173

179

181

191

193

197

211

223

227

229

233

239

241

251

257

263

269

271

277

281

283

293

307

311

313

317

331

337

347

349

353

359

367

373

379

383

389

397

401

409

419

421

431

433

439

443

449

457

461

463

467

473

479

481

487

491

499

503

509

521

523

527

531

533

541

547

557

563

569

571

577

587

593

599

601

607

613

617

619

631

641

643

647

653

659

661

673

677

683

689

691

697

701

709

711

719

727

733

739

743

751

757

761

769

773

787

797

809

811

821

823

827

829

833

839

847

853

857

863

869

877

881

883

887

893

897

901

907

911

919

929

937

941

943

947

953

967

971

973

979

983

991

997

Any number under 1,000,000 which can not be divided by one of the above numbers is prime.

找前5个默尼森数

- P 是素数且 M 也是素数，并且满足等式 $M=2^P-1$ ，则称 M 为默尼森数
- 例如 $P=5$ ， $M=2^P-1=31$ ，5和31都是素数，因此31是默尼森数。

使用函数可以在整体上简化程序结构，降低程序开发和修改的复杂度，提高程序的可读性和可复用性。

Python中的函数

7



Python中的函数

内建函数

指包含在 `__builtins__` 模块中的函数，安装完 Python 后可以直接使用

标准库

需要先导入模块再使用函数，每个库有相关的一些函数

第三方库

非常多，是 Python 重要的特征和优势

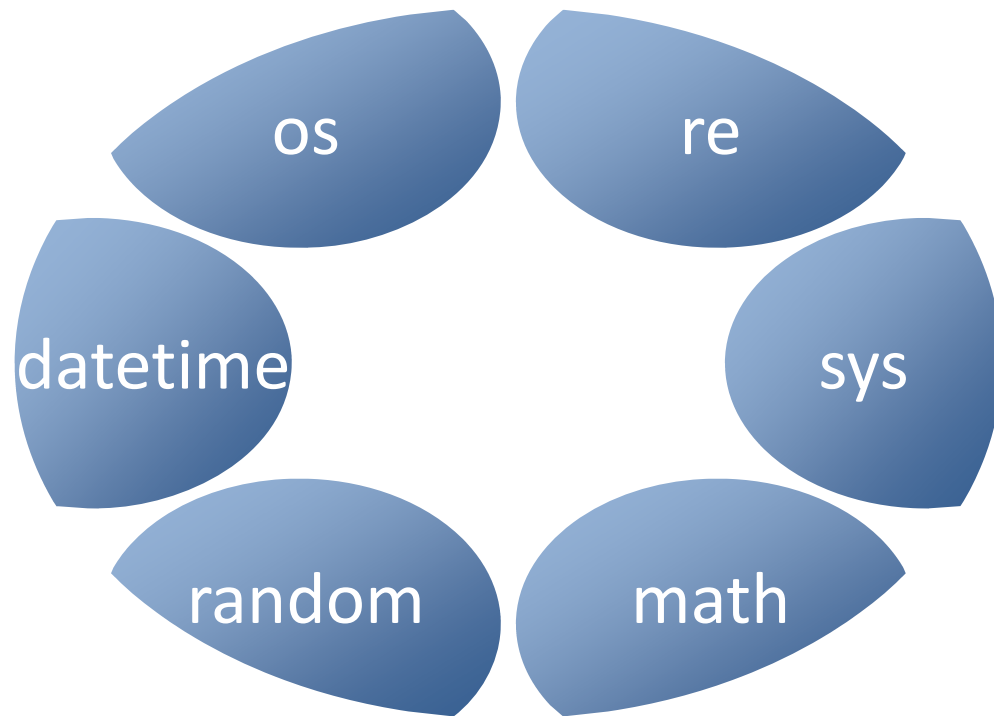
用户自定义函数

有固定的定义、调用和参数传递方式等

6.2

常用Python 标准库函数

常用Python标准库函数



已由系统事先定义
使用时直接导入后调用：

模块名.函数名(参数表)
函数名(参数表)

6.2.1 os模块中的函数

os模块中常用的处理文件及目录的函数

12

dir(os)



```
>>> import os
>>> os.getcwd()
'C:\\WINDOWS\\system32'
>>> path = 'd:\\temp'
>>> os.chdir(path)
>>> os.listdir(path)
['act.txt', 'awc', ..., 'web', 'write.exe']
```



```
>>> os.getcwd()
'd:\\temp'
>>> os.rename('current.txt', 'new.txt')
>>> os.remove('new.txt')
>>> os.mkdir('d:\\temp\\tempdir')
>>> os.rmdir('d:\\temp\\tempdir')
```

```
os.path.split(path)
os.path.join(path, name)
```

6.2.2 random模块中的函数

random模块——伪随机数生成器

14

S
ource

dir(random)

```
>>> import random
>>> random.seed(100)
>>> random.random() # 生成一个[0, 1.0)之间的一个随机浮点数
0.1456692551041303
>>> random.random()
0.45492700451402135
>>> random.random()
0.7707838056590222
>>> random.seed(100)
>>> random.random()
0.1456692551041303
```

random模块中常用函数的功能和使用方法

15

Source

dir(random)

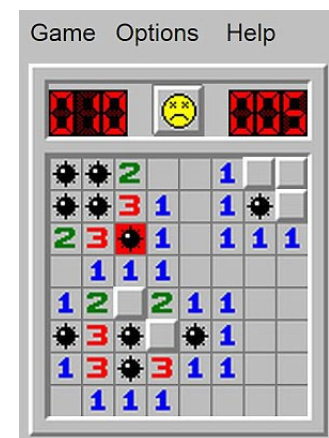
```
>>> import random
>>> random.choice(['C++', 'Java', 'Python'])
'Java'
>>> random.randint(1, 100)
37
>>> random.randrange(0, 10, 2)
4
>>> random.uniform(5, 10)
5.776718084305783
```

random模块中常用函数的功能和使用方法

16

Source

```
>>> import random
>>> random.sample(range(100), 10)
[16, 49, 26, 6, 61, 64, 29, 28, 34, 72]
>>> nums = [1002, 1004, 1001, 1005, 1008]
>>> random.shuffle(nums)
>>> nums
[1002, 1008, 1001, 1005, 1004]
```



6.2.3 datetime模块中的函数

datetime模块—date类中的常用函数例

18



```
>>> import datetime
>>> datetime.date.today()
datetime.date(2022, 7, 18)
>>> d = datetime.date(2020, 1, 1)
>>> d
datetime.date(2020, 1, 1)
>>> print(d)
2020-01-01
>>> d.isoformat()
'2020-01-01'
```

year

month

day

datetime模块—time类中的常用函数例

19

hour

minute

second

```
>>> import datetime
>>> t = datetime.time(22, 10, 15)
>>> t
datetime.time(22, 10, 15)
>>> print(t)
22:10:15
>>> t.isoformat()
'22:10:15'
```

datetime模块—datetime类中的常用函数例

20

Source

```
>>> from datetime import datetime
>>> dt = datetime.now()
>>> print(dt)
2022-07-18 15:25:07.092205
>>> print(dt.date())
2022-07-18
>>> print(dt.time())
15:25:07.092205
>>> print(dt.strftime('%a, %b %d %Y %H:%M'))
Mon, Jul 18 2022 15:25
```

形式1	形式2	含义
%a	%A	星期
%b	%B	本地月份
%d		月份
%y	%Y	年份
%H	%I	小时数
%M		分钟数

year

month

day

hour

minute

timestamp()和fromtimestamp()

21



```
>>> dt = datetime.datetime(2020, 1, 1, 0, 0)
>>> print(dt)
2020-01-01 00:00:00
>>> ts = dt.timestamp()
>>> ts
1577808000.0
>>> print(datetime.datetime.fromtimestamp(ts))
2020-01-01 00:00:00
```

6.2.4 sys模块中的属性

sys模块中的属性

```
import sys  
dir(sys)
```

```
>>> import sys  
>>> lst = []  
>>> for line in sys.stdin:  
.....     lst.append(line)
```

```
import sys
```

sys.argv命令行参数

```
$ test.py 1 2 3.45
```

```
s = 0  
for x in sys.argv[1:]:  
    s += eval(x)  
print(s)
```

6.3

函数的定义和调用

内建函
数或
标准库
函数

函数调用之前必须先定义

自定义
函数

6.3.1 函数的定义

函数的定义

语 法

```
def 函数名([参数表]):  
    "文档字符串"  
    函数体
```

def

- 表示函数开始，在第一行书写，该行被称为函数首部，用一个冒号结束；

函数名

- 函数名是函数的名称，是一个标识符，取名时尽量要做到见名识义；

函数的定义

语 法

```
def 函数名([参数表]):  
    "文档字符串"  
    函数体
```

文档字符串是可选的

参数表

- 函数名后紧跟一对圆括号(), 括号内可以有0个、1个或多个参数, 参数间用逗号分隔, 这里的参数称为形式参数 (简称形参), 形参只有被调用后才分配内存空间, 调用结束后释放所分配的内存空间;

函数体

- 函数体需要缩进, 它包含赋值语句和一些功能语句, 如果想定义一个什么也不做的函数, 函数体可以用pass语句表示。

自定义函数的创建



#example.py

```
>>> def printStr(x):  
    "print the string"  
    print(x)
```

一个简单的打印一个字符串的函数



```
>>> from example import printStr  
>>> print (printStr.__doc__)  
print the string
```

6.3.2 函数的返回

函数的返回

语 法

return 表达式1, 表达式2, ..., 表达式n

返回值

- 通常会通过return语句将值带回给主调函数
- 位置在函数体内
- 如果是返回多个值，则构成一个元组
- 如果不需要返回任何值，则不用return语句或用return None语句

函数的返回

32



```
>>> def foo(x, y):  
    """  
    计算参数的和  
    """  
    return x + y
```

返回两个参数的和的
函数定义

6.3.3 函数的调用

函数的返回

语 法

函数名([参数表])

调用

- 函数调用时括号中的参数称为实际参数（简称为实参），在函数调用时分配实际的内存空间。
- 如果有多个实参，实参间用逗号分隔。
- 可以没有实参，调用形式为：
 函数名() 圆括号不能省略。
- 调用时实参将值一一传递给形参，程序执行流程转移到被调用函数，函数调用结束后返回到之前的位置继续执行。

函数的导入和调用

35



```
>>> from example import printStr  
>>> printStr('Hi, Python!')  
Hi, Python!
```



example.py

例6.1 编写函数gcd(x, y)求x和y最大公约数



```
# prog6-1.py
```

```
def gcd(x, y):
```

```
    ''' calculate the GCD of x and y '''
```

```
    while x % y != 0:
```

```
        x, y = y, x % y
```

```
    return y
```

函数返回

```
x = eval(input("Enter the first number: "))
```

```
y = eval(input("Enter the second number: "))
```

```
gcdxy = gcd(x, y) # 调用函数
```

```
print('GCD({0:d}, {1:d}) = {2:d}'.format(x, y, gcdxy))
```

Output:

Enter the first number: 16

Enter the second number: 24

GCD(16, 24) = 8

例6.2 求1~100之间的所有素数

- 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

prog6-2 .py

```
from math import sqrt
def isprime(x):
    if x % 2 == 0:
        return x == 2
    k = int(math.sqrt(x))
    for j in range(3, k+1, 2):
        if x % j == 0:
            return False
    return True

for i in range(1, 101):
    if isprime(i):
        print(i, end = ' ')
```

例6.2 求1~100之间的所有素数

File

```
# prog6-2 .py
```

```
import math
```

```
def isprime(x):
```

```
    if x % 2 == 0:
```

```
        return x == 2
```

```
    k = int(math.sqrt(x))
```

```
    for j in range(3, k+1, 2):
```

```
        if x%j == 0:
```

```
            return False
```

```
    return True
```

```
for i in range(1,101):
```

```
    if isprime(i):
```

```
        print( i, end = ' ')
```

File

```
if __name__ == "__main__":
```

```
    for i in range(1,101):
```

```
        if isprime(i):
```

```
            print( i, end = ' ')
```

Python 中的main函数（主模块）

39

File

#test.py

```
def foo(x):  
    return x * x
```

```
x = 3  
result = foo(x)
```

Source

```
>>> import test  
>>> print(test.result)  
9
```

Python 中的main函数（主模块）

40



#test.py

```
def foo(x):  
    return x * x
```

```
if __name__ == "__main__":  
    x = 3  
    result = foo(x)
```



```
>>> import test
```

```
>>> print(test.result)
```

Traceback (most recent call last):

File "<pyshell#86>", line 1, in <module>
 print(test.result)

AttributeError: module 'test' has no attribute
'result'

```
>>> print(test.foo(3))
```

9

例6.3 左移或右移字符串中的字符

File

```
# prog6-3.py
```

```
def moveSubstr(s, flag, n):  
    if n > len(s):  
        return -1  
    else:  
        if flag == 1:  
            return s[n:] + s[:n]  
        else:  
            return s[-n:] + s[:-n]
```

File

```
if __name__ == "__main__":  
    s, flag, n = input("enter the 'string,flag,n':").split(',')  
    result = moveSubstr(s, int(flag), int(n))  
    if result != -1:  
        print(result)  
    else:  
        print('the n is too large')
```

Output:

输入测试数据

beautiful,1,3

输出结果为:

beautiful

若输入:

I love Python,2,3

输出结果为:

fulbeauti

主函数调用多个功能函数

```
def f():  
    ...  
def g():  
    ...  
if __name__ == '__main__':  
    f()  
    g()
```

```
def f():  
    ...  
def g():  
    ...  
  
f()  
g()
```

嵌套调用

```
def f():  
    ...  
def g():  
    f()  
  
if __name__ == '__main__':  
    g()
```

例6.4 模拟一个简易的用户注册和登录系统

44

File

```
# prog6-4.py
```

```
account = {'Zhangsan': '123456'}
```

account为全局变量

```
def sign_up():
```

```
    user_name = input("Please input your user name: ")
```

```
    while user_name in account.keys():
```

```
        user_name = input("User name exists, please choose another one:")
```

```
    password = input("Please input your password: ")
```

```
    account[user_name] = password
```

```
    print("Successfully sign up!")
```

例6.4 模拟一个简易的用户注册和登录系统

45



```
def sign_in():
    user_name = input("Please input your user name: ")
    if user_name not in account.keys():
        print("User name not found.")
    else:
        count = 0
        password = input("Please input your password: ")
        while account[user_name] != password:
            count += 1
            if count >= 3 :
                print("Bye - bye")
                break
            password = input("Wrong password, please input again: ")
        if account[user_name] == password:
            print("Successfully sign in!")
```

例6.4 模拟一个简易的用户注册和登录系统

46

File

```
if __name__ == '__main__':  
    while True:  
        # 注册0登陆1  
        cmd = input("Sign Up or Sign In? Please input 0 or 1:")  
        while cmd != '0' and cmd != '1':  
            print('Wrong command, please input again: ')  
            cmd = input("Sign Up: 0, Sign in: 1")  
        if cmd == '0':  
            sign_up()  
            continue  
        if cmd == '1':  
            sign_in()  
            break
```

例6.4 模拟一个简易的用户注册和登录系统

47

`__main__` 模块调用了 `sign_up()` 和 `sign_in()` 函数。算法设计为注册部分不退出，登录成功或密码输入超过错误次数后退出

Output:

Sign Up or Sign In? Please input 0 or 1:

0

Please input your user name: Lisi

Please input your password: 123456

Successfully sign up!

Sign Up or Sign In? Please input 0 or 1:

1

Please input your user name: Lisi

Please input your password: 123456

Successfully sign in!

例6.5 编写函数计算平均成绩

- 根据给出的一组学生的3门课成绩信息，编写函数计算每个学生的平均成绩，返回平均成绩最高和最低两位学生的姓名。



prog6-5.py

```
def search(scores):
    maxScore = 0
    minScore = 100
    for k, v in scores.items():
        avg = (scores[k][0] + scores[k][1] + scores[k][2]) // 3
        if avg >= maxScore:
            maxScore = avg
            maxName = k
        if avg <= minScore:
            minScore = avg
            minName = k
    return maxName, minName

if __name__ == "__main__":
    dictScores = {'Jerry': [87, 85, 91], 'Mary': [76, 83, 88], 'Tim': [97, 95, 89], 'John': [77, 83, 81]}
    maxName, minName = search(dictScores)
    print('{0} got the first place, {1} got the last.'.format(maxName, minName))
```

Output:

Tim got the first place, John got the last.

lambda函数

lambda函数又称为匿名函数，即没有具体的函数名
lambda函数的目的是让用户快速地定义单行函数，简化用户使用函数的过程。

```
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
```

例6.5 编写函数计算平均成绩——lambda函数

51



```
def search(scores):  
    t = sorted(scores.items(), key = lambda d : (d[1][0] +  
        d[1][1] + d[1][2]) // 3)  
    return t[len(t)-1][0], t[0][0]
```

使用lambda函数确定了排序函数sorted()的参数key, 确定了scores.items()的排序关键字

例6.5 编写函数计算平均成绩——lambda函数 52

File

```
>>> dScores = {'Jerry': [87, 85, 91], 'Mary': [76, 83, 88], 'Tim':  
[97, 95, 89], 'John': [77, 83, 81]}  
>>> a = sorted(dScores.items(), key = lambda d:d[0]) # 默认  
[('Jerry', [87, 85, 91]), ('John', [77, 83, 81]), ('Mary', [76, 83,  
88]), ('Tim', [97, 95, 89])]  
>>> a = sorted(dScores.items(), key = lambda d:d[1][0])  
[('Mary', [76, 83, 88]), ('John', [77, 83, 81]), ('Jerry', [87, 85,  
91]), ('Tim', [97, 95, 89])]
```


lambda函数与函数式编程

53



```
>>> lst = [3, 2, 5, 8, 1]
>>> list(map(lambda x: x*2, lst))
[6, 4, 10, 16, 2]
>>> lst = [1, 2, 3, 4, 5, 6]
>>> list(filter(lambda x: x%2 == 0, lst))
[2, 4, 6]
>>> from functools import reduce
>>> lst = [1, 2, 3, 4, 5]
>>> reduce(lambda x, y: x + y, lst)
15
```

例6.6 寻找数字朋友组

 *# prog6-6.py*

```
def findNumFriends(s):  
    s = s.split(',')  
    d = {}  
    for num in s:  
        sumNum = sum(map(int, num))  
        d[sumNum] = d.get(sumNum, []) + [num]  
    lst = sorted(d.items())  
    result = map(sorted, [x[1] for x in lst])  
    return result
```

例6.5 寻找数字朋友组

F_{ile}

prog6-6.py

```
if __name__ == "__main__":  
    s = input("Enter the numbers: ")  
    result = findNumFriends(s)  
    for item in result:  
        print(item)
```

Output:

Enter the numbers:

143,267,342,562,224,134,276,252

['134', '143', '224']

['252', '342']

['562']

['267', '276']

6.4

函数的参数

- 函数调用时将实参一一传递给形参，通常实参和形参的个数要相同，类型也要相容，否则容易发生错误。
 - 参数在调用过程中的变化
 - 参数可以设定默认值

参数的可变性

```
def add(b):  
    b.append(4)  
    print(b)
```

```
a = [2, 6, 5]  
add(a)  
print(a)
```

```
def caps(b):  
    b = b.upper()  
    print(b)
```

```
a = 'hello'  
caps(a)  
print(a)
```

可变对象 VS 不可变对象

位置参数

59

Source

```
>>> def printGrade(name, stuID, grade):  
        print("{0}({1})'s grade is {2}.".format(name, stuID, grade))  
>>> printGrade('Mary', '1002', 'A')  
Mary(1002)'s grade is A.
```

实参写反会怎样?

Source

```
>>> printGrade('A', '1002', 'Mary')  
A(1002)'s grade is Mary.
```

错误的结果

1. 关键字参数




```
>>> def printGrade(name, stuID, grade):  
    print("{0}({1})'s grade is {2}.".format(name, stuID, grade))  
>>> printGrade(name = 'Jerry', grade = 'B' , stuID = '1005')  
Jerry(1005)'s grade is B.
```

关键字参数是使用参数名提供的参数。有了关键字参数顺序就不会有影响，并且调用时每个参数的含义更清晰。


2. 默认参数

- 在定义函数时给某些参数设定默认值，默认参数以赋值语句的形式给出



```
>>> def area(r, pi = 3.14159):  
    return pi * r * r
```


- 设定了 π 的默认值，确定了统一的精度，调用时如果使用默认值则该位置的实参可以省略不写



```
>>> area(3)  
28.274309999999996
```


2. 默认参数

- 默认参数值在调用时可以修改



```
>>> area(4, 3.14)
50.24
```

- 调用时也一样可以利用关键字参数改变参数顺序



```
>>> area(pi = 3.14, r = 4)
50.24
```

2. 默认参数



```
>>> def printGrade(name, className = 'Courage', grade):  
    print("{0}({1})'s grade is {2}.".format(name, className, grade))  
>>> printGrade('Mary', 'A')
```

SyntaxError: non-default argument follows default argument

语法错误：非默认参数跟
在了默认参数之后

默认参数是可以修改的，如果在定义时将默认参数放在非默认参数前面的话，Python解释器无法判断给出的第二个实参'A'是修改了默认参数的值还是对应了后面的参数

2. 默认参数



```
>>> def printGrade(name, grade, className = 'Courage'):
    print("{0}({1})'s grade is {2}.".format(name, className, grade))
>>> printGrade('Mary', 'A')
Mary(Courage)'s grade is A.
```

定义函数时必须将默认参数放在非默认参数的后面。

3. 可变长参数

- Python中允许传递一组数据给一个形参，形参tupleArgs前有一个“*”号，是可变长位置参数的标记，用来收集其余的位置参数，将它们放到一个元组中



```
>>> def greeting(args1, *tupleArgs):  
    print(args1)  
    print(tupleArgs)
```

3. 可变长参数

S_{ource}

```
>>> def greeting(args1, *tupleArgs):  
    print(args1)  
    print(tupleArgs)  
>>> greeting('Hello,', 'Wangdachuan', 'Liuyun', 'Linling')  
Hello,  
( 'Wangdachuan', 'Liuyun', 'Linling')
```

实参中'Hello,'传递给位置参数args1，其余的三个字符串传递给可变长的位置参数tupleArgs，调用后将这组实参放到一个元组中输出。

3. 可变长参数



```
>>> def greeting(args1, *tupleArgs):  
    print(args1)  
    print(tupleArgs)  
>>> names = ('Wangdachuan', 'Liuyun', 'Linling')  
>>> greeting('Hello,', *names)  
Hello,  
('Wangdachuan', 'Liuyun', 'Linling')
```


例6.7 寻找全数字

如果一个 n ($n \leq 9$) 位数刚好包含了1至 n 中所有数字各一次则这个数被称为全数字数，例如四位数1324就是一个全数字数。从键盘输入一组（不止一个）整数，输出其中的全数字，若找不到则输出 “not found”。具体要求如下：

(1)编写函数pandigital(nums)，查找列表nums中的全数字数，存入结果列表中并返回；

(2)编写主模块，从键盘输入多个正整数数字串（中间用一个逗号分隔），调用函数pandigital()查找其中的全数字数，将它们用逗号间隔输出在一行上，若找不到则输出 “not found”。

例6.7 寻找全数字

```
 # prog6-7.py
def pandigital(nums):
    lst = []
    for x in nums:
        length = len(str(x))
        if {int(k) for k in str(x)} == set(range(1, length+1)):
            lst.append(x)
    return lst

if __name__ == '__main__':
    nums = eval(input())
    lst = pandigital(nums)
    if lst:
        print(*lst, sep = ',') # 序列解包
    else:
        print('not found')
```

输入:
1243,322,321,1212,2354
输出:
1243,321
输入:
124,23
输出:
not found

3. 可变长参数——可变长关键字参数

- 用两个星号标记可变长的关键字参数。可变长关键字参数允许传入多个(可以是0个)含参数名的参数,这些参数在函数内自动组装成一个字典。也可先将参数名和参数构建成一个字典,然后作为可变长关键字参数传递给函数调用。



```
>>> def assignment(**dictArgs):  
        print(dictArgs)  
>>> assignment(x = 1, y = 2, z = 3)  
{'x': 1, 'z': 3, 'y': 2}  
>>> data = {'x': 1, 'z': 3, 'y': 2}  
>>> assignment(**data)  
{'x': 1, 'z': 3, 'y': 2}
```

4. 可变长参数——可变长位置参数 和可变长关键字参数

71

```
>>> def greeting(x, *args, **kwargs):
    print(x, end = ' ')
    print(*args, sep = ',')
    for x in kwargs:
        print(x+' : '+kwargs[x])

>>> names = ['Wangdachuan', 'Liuyun', 'Linling']
>>> info = {'schoolName' : 'NJU', 'City' : 'Nanjing'}
>>> greeting('Hello,', *names, **info)
Hello, Wangdachuan,Liuyun,Linling
schoolName: NJU
City: Nanjing
```

例6.8 实现用户信息注册登记

72

- 要求必须登记姓名，性别和手机号码，其他如年龄、职业等信息不强制登记。



例6.8 实现用户信息注册登记

73

F
ile

prog6-8.py

```
def register(name, gender, phonenumber, **otherinfo):  
    ''' register users information '''  
    print('name: ', name, 'gender: ', gender, 'phone num: ', phonenumber)  
    print('other information: ', otherinfo)
```

例6.8 实现用户信息注册登记

74



```
>>> register('Chenqian', 'M', '111111111111')  
name: Chenqian gender: M phone num: 11111111111  
other information: {}
```

例6.8 实现用户信息注册登记

Source

```
>>> otherinfo = {'age': 24, 'city': 'Nanjing', 'job': 'teacher'}  
>>> register('Limei', 'F', '22222222222', **otherinfo)  
name: Limei gender: F phone num: 2222222222  
other information: {'age': 24, 'city': 'Nanjing', 'job': 'teacher'}
```

除了name、gender、phonenum外如果没有登记额外的信息，则可变长关键字参数otherinfo收集不到任何参数，调用后输出一个空字典。如果有额外的信息登记，参数传递时这些参数被组装成一个字典。

6.5

变量的作用域

- 每个变量都有自己的作用域，也就是在某个代码段内使用该变量是合法的，在此代码段外使用该变量则是非法的。除了作为全局作用域的变量外，每次函数调用都会创建一个新的作用域。



变量作用域

78



```
>>> def f(): x = 5
```

```
>>> f()
```

```
>>> print(x)
```

Traceback (most recent call last):


File "<pyshell#17>", line 1, in <module>

print(x)


NameError: name 'x' is not defined

对不同作用域同名变量的处理

79


 `>>> def f(): x = 5`
`>>> x = 3`
`>>> f()`
`>>> print(x)`
`3`



 `>>> def f(): y = 5`
`>>> x = 3`
`>>> f()`
`>>> print(x)`
`3`

函数内部使用全局变量

函数内部虽然可以使用全局变量，但使用要慎重，如果在多个函数内部使用全局变量，则无法确定全局变量某一时刻的值，容易发生错误。




```
>>> def f():  
        y = 5  
        print(x + y)  
  
>>> x = 3  
>>> f()  
8
```


局部变量和全局变量同名时

81

在局部变量（包括形参）和全局变量同名时，局部变量屏蔽（Shadowing）全局变量

- 程序执行结果=？

 `>>> x = 3`
`>>> def f():`
 `x = 5`
 `print(x ** 2)`
`>>> f()`

函数内部全局变量的使用

82



```
>>> x = 3
```

```
>>> def f():
```

```
    print(x ** 2)
```

```
    x = 5
```

```
    print(x ** 2)
```

```
>>> f()
```

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

f()

File "<pyshell#2>", line 2, in f

print(x ** 2)

UnboundLocalError: local variable 'x' referenced before assignment

函数内部同时出现局部变量和全局变量

83



```
>>> x = 3
>>> def f():
    global x
    print(x ** 2)
    x = 5
    print(x ** 2)
>>> x = 3
>>> f()
9
25
```

使用关键字global声明将使用全局变量

嵌套定义

84

```
def f1():  
    def f2():  
        print('inner')  
    print('outer')  
    f2()  
  
f1()
```

```
def f1():  
    print('outer')  
def f2():  
    print('inner')  
  
f1()  
f2()
```

```
>>> f2()
```

NameError: name 'f2' is not defined

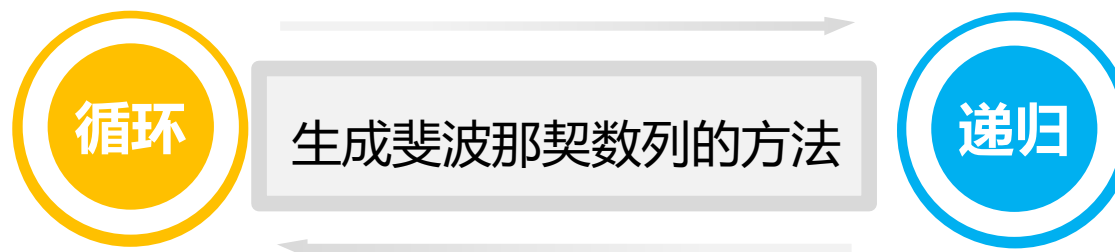
嵌套定义与闭包

```
def counter():  
    count = 0  
    def inc():  
        # nonlocal count # 用nonlocal对普通变量闭包，否则无法修改  
        # count += 1  
        return count  
    return inc  
  
foo = counter()  
print(foo())
```

闭包（closure）：嵌套函数中内层函数引用到了外层函数的自由变量，就构成了闭包

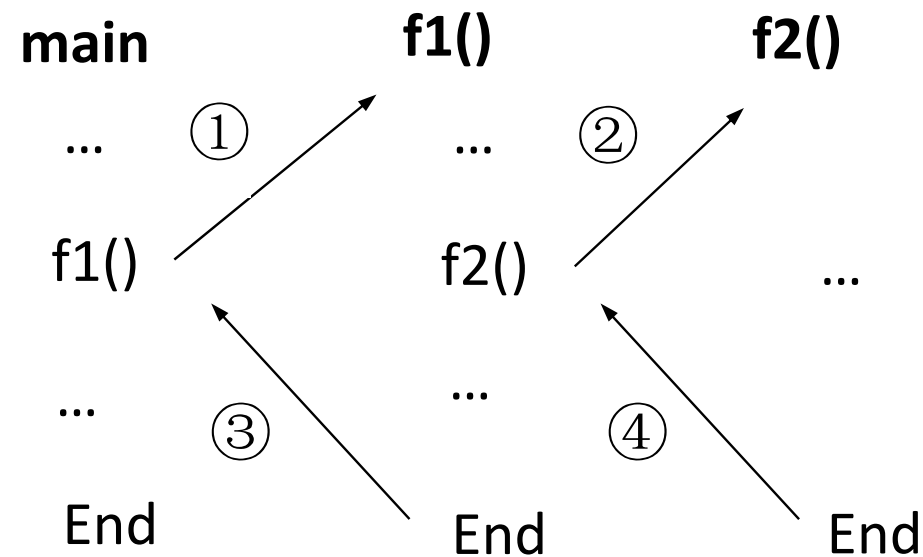
6.6

递归



递归是最能表现计算思维的算法之一

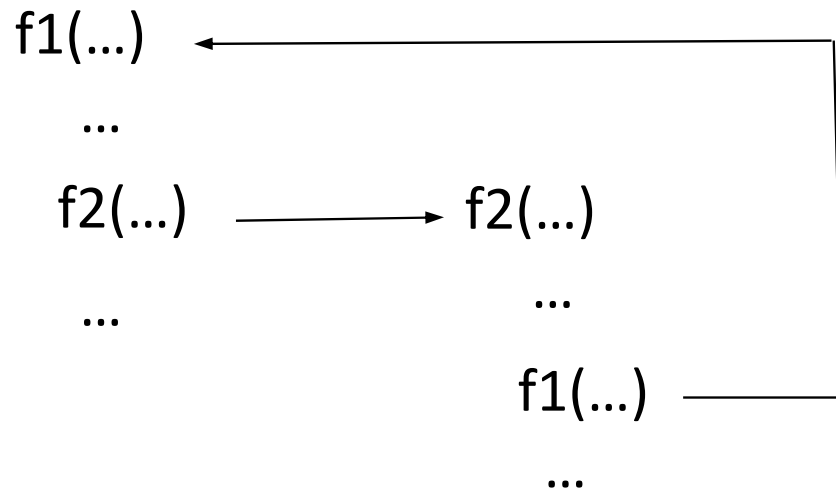
函数的嵌套调用



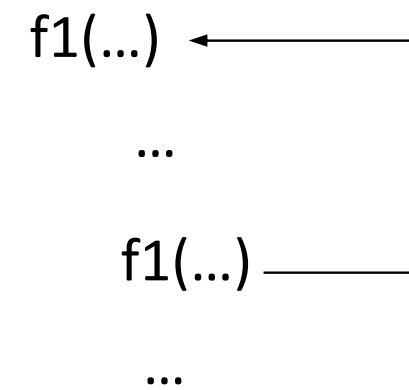
- 函数间可以相互调用，如果在__main__模块中调用了f1函数，在函数f1中又调用了函数f2，就形成了函数的嵌套调用。

直接递归和间接递归

89



(a) 间接递归调用

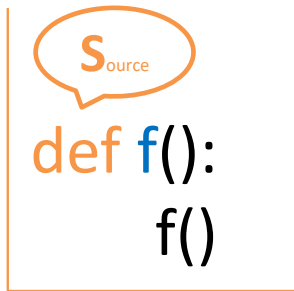


(b) 直接递归调用

递归是特殊的嵌套调用，是对函数自身的调用

正确的递归调用的要求

- 有一个比原始调用规模小的函数副本；
- 有基本情况即递归终止条件。



The diagram shows a code snippet for a recursive function. Above the code is a speech bubble containing the word "Source". The code is enclosed in a box with an orange border.

```
def f():  
    f()
```

无穷递归 (infinite recursion)

正确的递归调用的要求

要求1

有一个比原始调用
规模小的函数副本

要求2

有基本情况，即递
归终止条件

递归调用的过程

- 每一次递归调用要解决的问题都要比上一次的调用简单，规模较大的问题可以往下分解为若干个规模较小的问题，规模越来越小最终达到最小规模的递归终止条件（基本情况）
- 解决完基本情况后函数沿着调用顺序逐级返回上次调用，直到函数的原始调用处结束
- 一般会包含一个选择结构，条件为真时计算基本情况结束递归调用后返回，条件为假时简化问题执行副本继续递归调用。

递归的特点



例6.8 编写递归函数计算n的阶乘

$$n! = \begin{cases} 1 & (\text{当 } n=1) \\ n \times (n-1)! & (\text{当 } n>1) \end{cases}$$

n的阶乘的定义是一种递归定义

例6.8 编写递归函数计算n的阶乘



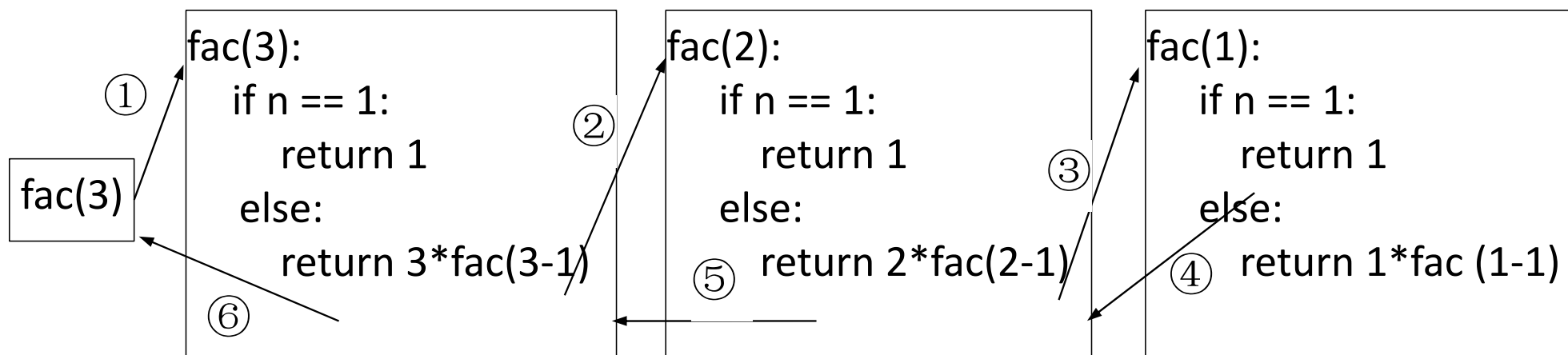
prog6-8.py

```
def fac(n):  
    if n == 1:  
        return 1  
    else:  
        return n * fac(n-1)
```

递归必须要有边界条件，
即停止递归的条件

例6.8 编写递归函数计算n的阶乘 ——递归过程

96



递归函数的每次调用时系统都为函数的局部变量（包括形参）分配本次调用使用的存储空间，直到本次调用结束返回主调程序时方才释放。

例6.9 10进制数转成2进制数输出

97

```
def trans(n):  
    if n >= 2:  
        trans(n // 2)  
    print(n % 2, end = '')
```

求n的阶乘

斐波那契数列

用二分法求方程的根



有明显的迭代方式,
一般不推荐使用递归实现

经典的Hanoi（汉诺塔）游戏

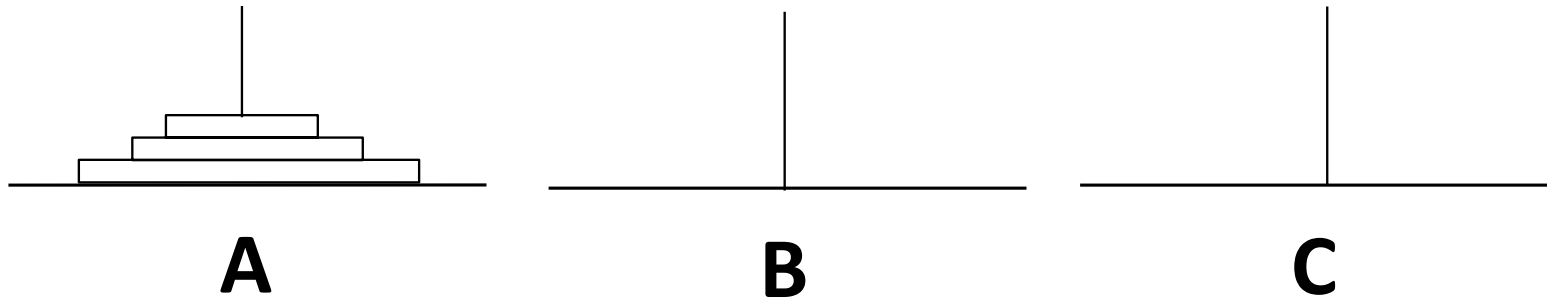
八皇后



适合用递归实现

例6.10 Hanoi塔问题

99



Hanoi塔问题的描述是：有3个底座（分别标记为A、B和C）各有一根针，A座的针上已从下往上从大到小依次叠放了64个（简单起见用3个盘子表示）大小不同的盘子，要求将A座针上的盘子全部搬到C座的针上。在搬运过程中可以借助于B座的针，每次只能搬一个盘子，任何时候每根针上的盘子都必须保持大盘子在下小盘子在上的叠放顺序。

例6.10 Hanoi塔问题

- 64个盘子需要搬运 $2^{64}-1$ 次。当A座的针上只有一个盘子时（即 $n=1$ ）数据规模最小，依据搬运要求和搬运规则可以很容易搬运这个盘子，因此直接输出搬运该盘子的操作指令后返回。当A座针上有多个盘子时（即 $n>1$ ），可以将这 n 个盘子的搬运操作分解为三部分：
 - （1）输出将A座针上的前 $n-1$ 个盘子借助C座针搬到B座针的搬运指令；
 - （2）输出将A座针上剩下的最后一个盘子（编号为 n ）直接从A座针搬到C座针的搬运指令；
 - （3）输出将B座针上的 $n-1$ 个盘子借助A座针搬到C座针的搬运指令。

例6.10 Hanoi塔问题

F_{ile}

```
# prog6-10.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)
```

```
n = int(input('input the number of plates: '))  
hanoi('a', 'b', 'c', n)
```

Output:

```
input the number of plates: 3  
a -> c  
a -> b  
c -> b  
a -> c  
b -> a  
b -> c  
a -> c
```

- 查看递归深度

```
>>> import sys
```

```
>>> sys.getrecursionlimit()
```

```
1000
```

- 手工修改默认值

```
sys.setrecursionlimit(20000)
```

- 函数的概念
- 常用Python标准库函数
- 函数的定义和调用
- 函数的参数
- 变量的作用域
- 递归函数

