



Chap4 Function

函数

Nanjing University

Department of Computer Science and Technology

Department of University Basic Computer Teaching

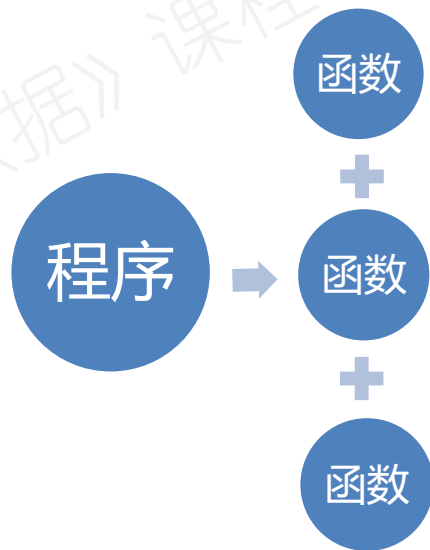
找前5个默尼森数

- 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	467	471	473	479	487	491	493	503	509	517																																																																											
521	523	541	547	551	557	563	569	571	577	581	587	593	599	601	607	613	617	619	623	627	631	637	641	643	647	653	659	661	667	671	673	677	683	687	691	697	701	707	709																																																												
713	719	727	731	733	737	743	749	751	757	761	767	771	773	779	781	787	791	793	797	803	809	811	817	821	823	827	829	833	837	839	841	847	851	853	857	859	863	867	869	871	877	881	883	887	891	893	897	901	907	911	913	917	919	923	927	929	931	937	941	943	947	953	959	961	967	971	973	977	983	989	991	993	997																										
Any number under 1,000,000 which can not be divided by one of the above numbers is prime.																																																																																																			

- 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)



4.1

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。

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。

Python中的函数



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

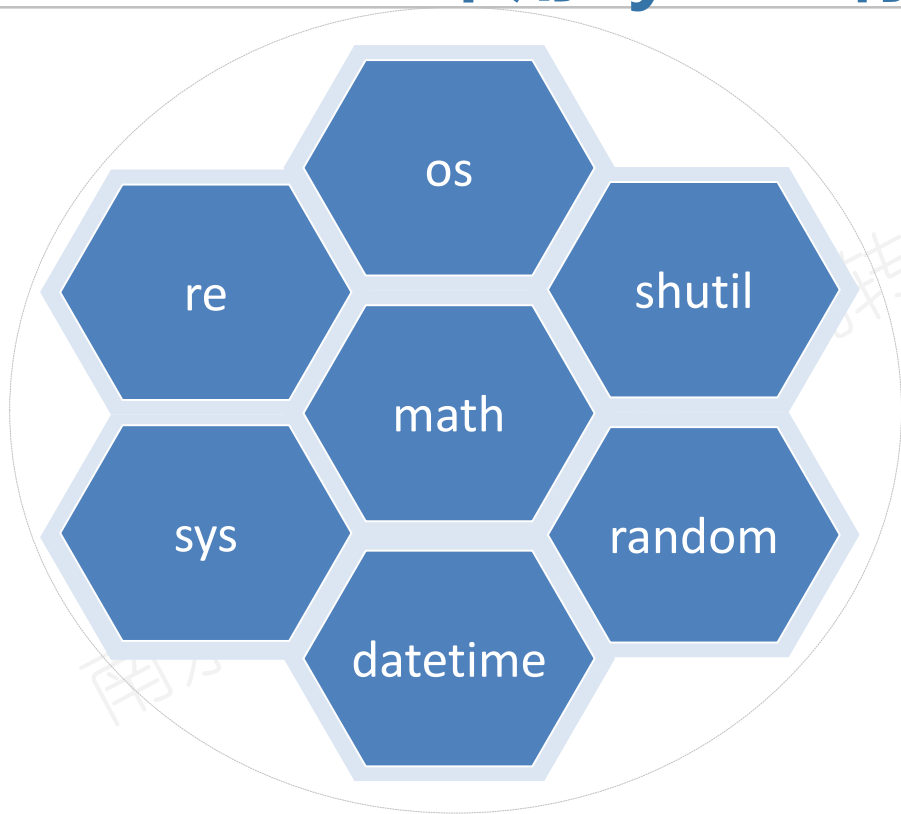
用户自定义函数

有固定的定义、调用和参数传递方式等

4.2

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)

os模块中常用的处理文件及目录的函数

11

```
import os
dir(os)
```

```
>>> import shutil
>>> shutil.rmtree(path)
>>> os.path.join(root_dir, 'a.txt')
```

```
Source
>>> import os
>>> os.getcwd()
'C:\\WINDOWS\\system32'
>>> path = 'd:\\temp'
>>> os.chdir(path)
>>> os.getcwd()
'd:\\temp'
>>> os.listdir(path)
['act.txt', 'awc', ..., 'web', 'write.exe']
>>> os.rename('current.txt', 'new.txt')
>>> os.remove('new.txt')
>>> os.mkdir('d:\\temp\\tempdir')
>>> os.rmdir('d:\\temp\\tempdir')
```

sys模块中标准输入和输出属性

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



```
>>> import sys
>>> lst = []
>>> for line in sys.stdin:
.....     lst.append(line)
>>> lst = []
>>> for line in sys.stdin:
...     name, score = line.split()
...     lst.append(name)
>>> s = 0
>>> for x in sys.stdin:
...     if x.strip() != '0':
...         s += int(x)
...     else:
...         break
>>> sys.stdout.write('hello')
hello5
```

random模块中常用函数的功能和使用方法

13

Source

```
>>> import random
>>> random.seed(100)
>>> random.random()
0.1456692551041303
>>> random.random()
0.38859914082194214
>>> random.choice(['C++', 'Java', 'Python'])
'Java'
>>> random.randint(1, 100)
37
>>> random.randrange(0, 10, 2)
4
>>> random.uniform(5, 10)
5.776718084305783
```

```
import random
dir(random)
```

random模块中常用函数的功能和使用方法

14

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]
```




随机字符串小任务

```
['xx:xx.xxx GET /service2/controller22/action223\n',  
'xx:xx.xxx POST /service3/controller31/action312\n',  
'xx:xx.xxx DELETE /service3/controller33/action333\n',  
'xx:xx.xxx UPDATE /service1/controller11/action111\n',  
'xx:xx.xxx DELETE /service1/controller12/action122\n',  
'xx:xx.xxx GET /service1/controller13/action132\n',  
'xx:xx.xxx POST /service3/controller32/action322\n',  
'xx:xx.xxx GET /service3/controller32/action321\n',  
'xx:xx.xxx UPDATE /service3/controller32/action321\n',  
'xx:xx.xxx GET /service3/controller32/action322\n']
```

```
method = ['GET', 'POST', 'UPDATE', 'DELETE']
```

```
URI1 = 'service', URI2 = 'controller', URI3 = 'action'
```

datetime模块—date类中的常用函数例

 Source

```
>>> import datetime
>>> datetime.date.today()
datetime.date(2020, 3, 21)
>>> 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

17

Source

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

hour

minute

second

datetime模块—datetime类中的常用函数例

18

Source

```
>>> import datetime
>>> dt = datetime.datetime.now()
>>> print(dt)
2020-03-21 22:37:36.919642
>>> print(dt.date())
2020-03-21
>>> print(dt.time())
22:37:36.919642
>>> print(dt.strftime('%a, %b %d %Y %H:%M'))
Sat, Mar 21 2020 22:37
```

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

year

month

day

hour

minute

second

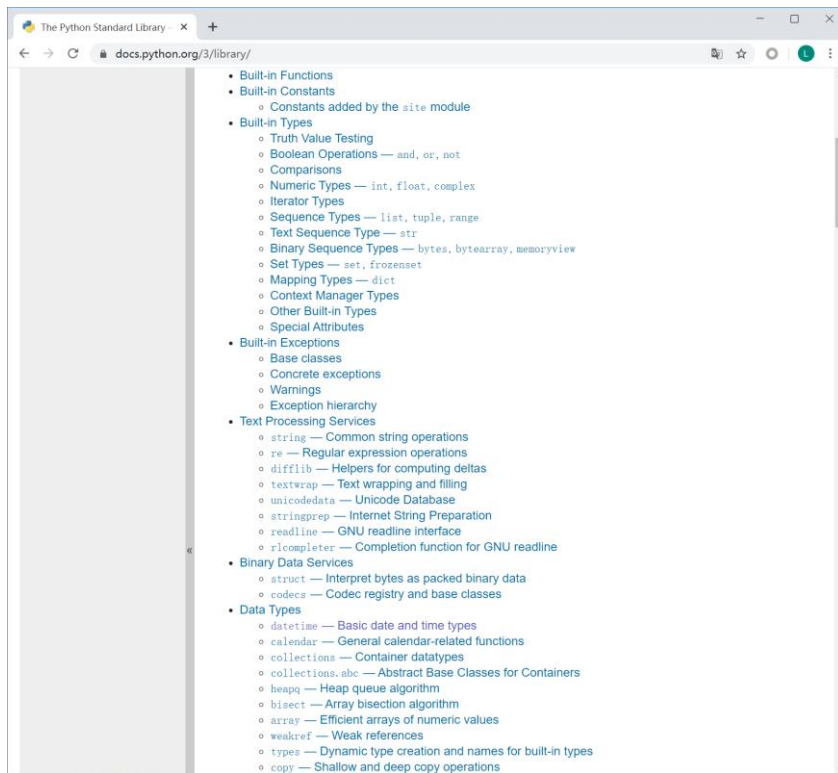
timestamp()和fromtimestamp()

Source

```
>>> 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
```

The Python Standard Library

20



参考链接:

<https://docs.python.org/3/library/>

4.3

函数的定义和调用

自定义函数的创建



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

```
>>> from pStr import printStr  
>>> print(printStr.__doc__)
```

def 函数名([参数表]):

["文档字符串"]

函数体



pStr.py

函数的返回

语 法

`return` 表达式1, 表达式2, ..., 表达式n

```
>>> def square(x, y):
```

```
    """
```

```
    计算参数的和与差
```

```
    """
```

```
    return x + y, x - y
```

函数的调用

```
>>> printStr('Hi, Python!')
```

```
Hi, Python!
```

```
>>> r1, r2 = square(3, 5)
```

```
>>> r1
```

```
8
```

```
>>> r2
```

```
-2
```


例4.2 求2~100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

```
for i in range(2, 101):  
    if isprime(i):  
        print(i, end = ' ')
```

File

prog4-2 .py

```
from math import sqrt
```

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

```
    if x == 2:
```

```
        return True
```

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

```
        return False
```

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

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

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

```
            return False
```

```
    return True
```

例4.2 求1~100之间的所有素数

F_{ile}

```
if __name__ == "__main__":  
    for i in range(2, 101):  
        if isprime(i):  
            print(i, end = ' ')
```

```
>>> __name__  
'__main__'  
>>> import abc  
>>> abc.__name__  
'abc'
```

Python中的main函数



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

```
def main():  
    x = 3  
    print(foo(x))
```

```
if __name__ == "__main__":  
    main()
```

字符串循环移动

- 自定义函数move_substr(s, flag, n)，将传入的字符串s按照flag（1代表循环左移，2代表循环右移）的要求左移或右移n位，结果返回移动后的字符串，若n超过字符串长度则结果返回-1。__main__模块中从键盘输入字符串、左移和右移标记以及移动的位数，调用move_substr()函数若移动位数合理则将移动后的字符串输出，否则输出“the n is too large”。

```
def moveSubstr(s, flag, n):
```

```
    if n > len(s):
```

```
        return -1
```

```
    else:
```

```
        if flag == 1:
```

```
            return s[n:]
```

```
        else:
```

```
            return s[-n:]
```

```
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')
```

例 模拟一个简易的用户注册和登录系统

30

F_{ile}

prog4-4.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 模拟一个简易的用户注册和登录系统

31



```
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("Login success!")
```

例 模拟一个简易的用户注册和登录系统



```
if __name__ == '__main__':  
    while True:  
        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 模拟一个简易的用户注册和登录系统

33

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

Login success!

嵌套调用

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

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

嵌套定义

```
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

4.4

函数的参数

1. 位置参数

S
ource

```
>>> 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.
```

S
ource

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

2. 关键字参数

Source

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

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。调用时每个参数的含义更清晰。

2. 关键字参数

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(68, y = False)
```

```
68 is OK
```

3. 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

Source

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

Source

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

Source

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

Source

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


3. 默认参数

S

```
>>> 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

定义语法错误：非默认参数跟在了默认参数之后

3. 默认参数

Source

```
>>> 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.
```

4. 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)
```

4. 可变长参数

S

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

4. 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。



```
>>> 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

46

S
ource

```
>>> def greeting(args1, *args, **kwargs):  
    print(args1)  
    print(args)  
    print(kwargs)  
  
>>> names = ['Wangdachuan', 'Liuyun', 'Linling']  
>>> info = {'schoolName': 'NJU', 'City': 'Nanjing'}  
>>> greeting('Hello,', *names, **info)  
Hello,  
( 'Wangdachuan', 'Liuyun', 'Linling' )  
{ 'City': 'Nanjing', 'schoolName': 'NJU' }
```

例4.6 实现用户信息注册登记

- 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4.6 实现用户信息注册登记

F_{ile}

prog4-6.py

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


例4.6 实现用户信息注册登记



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

例4.6 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'}
```

- 自定义sum(x, y)参数个数确定的
- 自定义sum(x, *args)函数
- 自己实现sum()函数

```
In [1]: help(sum)
Help on built-in function sum in module builtins:

sum(iterable, start=0, /)
    Return the sum of a 'start' value (default: 0) plus an iterable of numbers

    When the iterable is empty, return the start value.
    This function is intended specifically for use with numeric values and may
    reject non-numeric types.
```

```
sorted(iterable, /, *, key=None, reverse=False)
```

- / 之前的参数都是 positional-only参数
- * 之后的参数都是 keyword-only参数

4.5

LAMBDA函数 与函数式编程

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
```

- 匿名函数

将一个列表中的所有元素按绝对值大小进行排序

```
list1 = [3,5,-4,-1,0,-2,-6]
sorted(list1, key=abs)
```

```
list1 = [3,5,-4,-1,0,-2,-6]
def get_abs(x):
    return abs(x)
sorted(list1, key=get_abs)
```

```
list1 = [3,5,-4,-1,0,-2,-6]
sorted(list1, key=lambda x: abs(x))
```

例4.3 编写函数计算平均成绩——lambda函数

55

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])  
[('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])]
```

例4.3 编写函数计算平均成绩——lambda函数

56



```
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]
```


- 函数式编程的主要由3个基本函数和1个算子构成

基本函数：

map()、reduce()、
filter()

算子(operator)：
lambda

```
>>> 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
```

1. 列表数值型与字符串类型元素切换
2. ['pku', 'nju', 'cmu']中的字符串转换成全大写

数字筛选

输入一个2（包含）至9（包含）之间的一位数字，输出1-100中剔除了包含该数字、该数字的倍数的所有数字，输出满足条件的数，要求一行输出10个数字（最后一行可能不足10个），数字之间用“,”分隔。

```
n = int(input())
r = list(map(str, filter(lambda x: x%n and str(n) not in str(x), range(1, 101))))
for i in range(0, len(r), 10):
    print(','.join(r[i:i+10]))
```

例 寻找数字朋友组

F

find_friends.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']

例 寻找数字朋友组

File

find_friends.py

def findNumFriends(s):

s = s.split(',')

d, result = {}, []

for num **in** s:

sumNum = 0

for ch **in** num:

sumNum += int(ch)

if sumNum **in** d:

d[sumNum] += [num]

else:

d[sumNum] = [num]

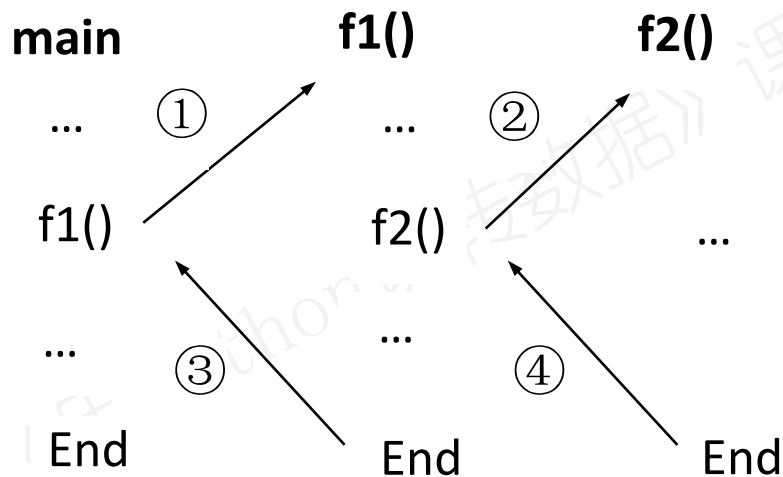
lst = sorted(d.items(), key = lambda d: d[0])

result = list(map(sorted, [x[1] **for** x **in** lst]))**return** result

4.6

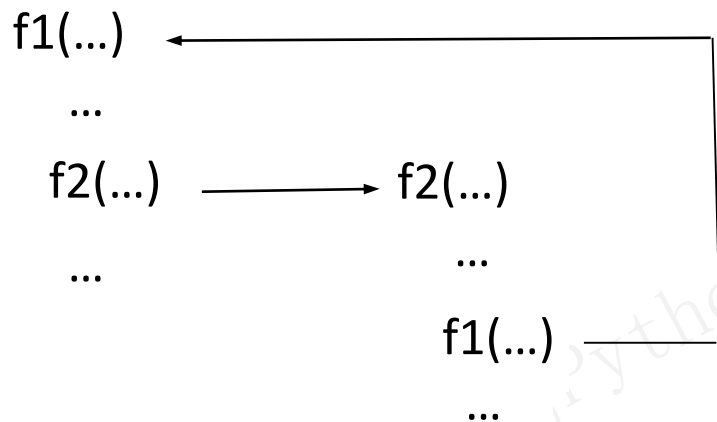
递归

函数的嵌套调用

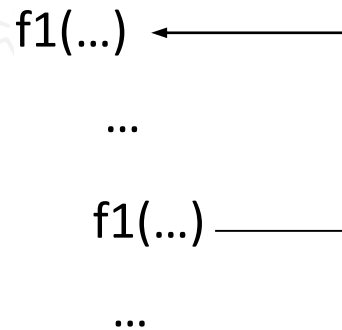


- 函数间可以相互调用，如果在 `_main_` 模块中调用了 `f1` 函数，在函数 `f1` 中又调用了函数 `f2`，就形成了函数的嵌套调用。

直接递归和间接递归



(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。



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

无穷递归 (infinite recursion)

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并结束递归调用，条件为假时简化问题执行副本继续递归调用。

- 递归的执行

系统资源消耗

比循环 **大**

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

例4.7 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4.7 编写递归函数计算n的阶乘

F_{ile}

```
# prog6-7.py
```

```
def fac(n):
```

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

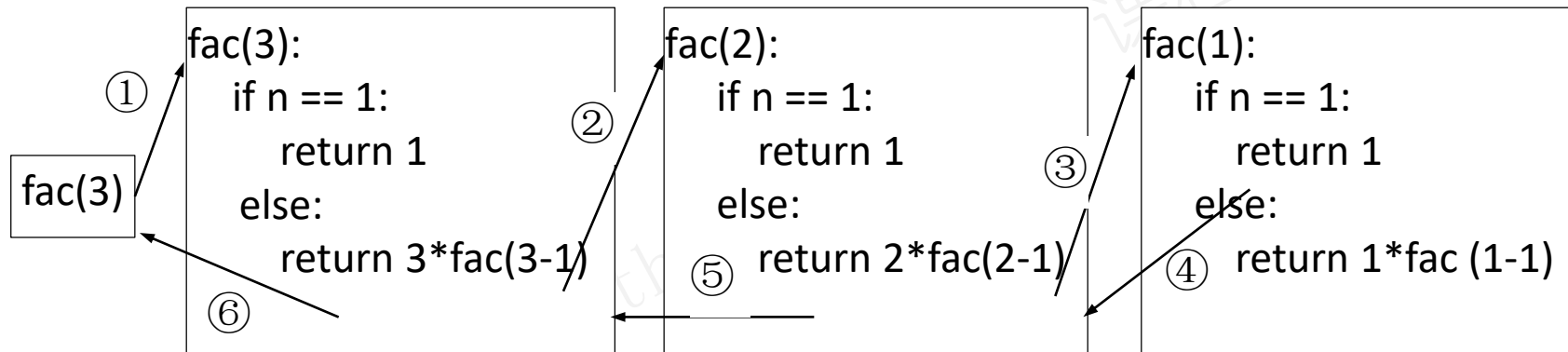
```
        return 1
```

```
    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4.7 编写递归函数计算n的阶乘



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。

递归和迭代

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

经典的Hanoi（汉诺塔）游戏

八皇后



适合用递归实现

```
def proc(s):  
    if s == "":  
        return s  
    else:  
        return proc(s[1:])+s[0]
```

```
s = input("input a string: ")  
print(proc(s))
```

输入step


```
def f(n):  
    if n < 0:  
        print('-', end = '')  
        n = -n  
    if n // 10:  
        f(n//10)  
    print(n%10, end = '')
```

f(-345)

折半查找法

```
def b_search(x, low, high, key):  
    mid = (low+high) // 2  
    if x[mid] == key:  
        return mid  
    elif low > high:  
        return -1  
    elif key < x[mid]:  
        return b_search(x, low, mid-1, key)  
    else:  
        return b_search(x, mid+1, high, key)
```

10进制数转成2进制数

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

递归深度的设定

- 查看递归深度

```
>>> import sys
```

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

```
1000
```

- 手工修改默认值

```
sys.setrecursionlimit(2000)
```

4.7

变量作用域



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

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

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

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

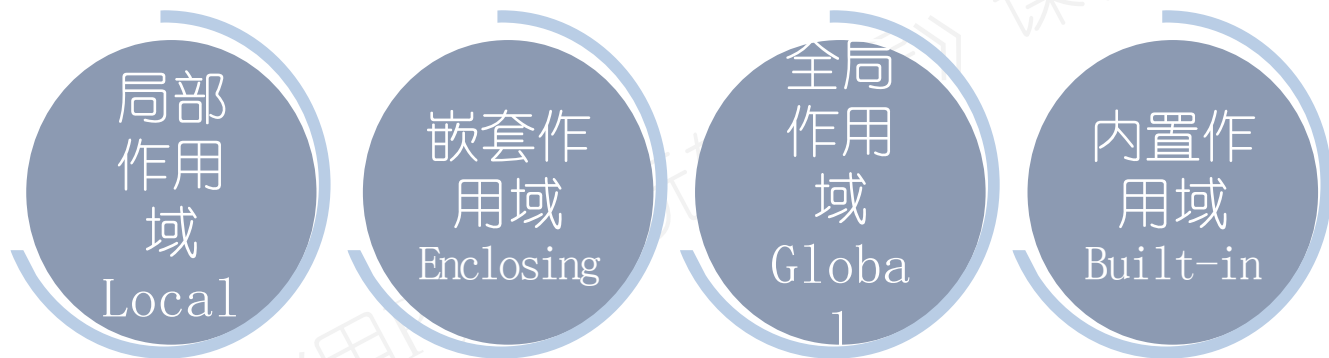
print(x)

NameError: name 'x' is not defined

- 作用域会生成一个命名空间(namespace)
- 命名空间是从名称（标识符）到对象的映射
- 不同命名空间中的名字之间没有关系



变量作用域



当在函数中使用未确定的变量名时，搜索变量名的顺序遵循LEGB法制

对不同作用域同名变量的处理

S_{source}

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

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

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

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

3

S_{source}

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

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

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

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

3

局部变量和全局变量同名时


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



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


函数内部同时出现同名局部变量和全局变量

83

 Source

```
>>> 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

 Source

```
>>> 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

慎用全局变量