人工智能程序设计

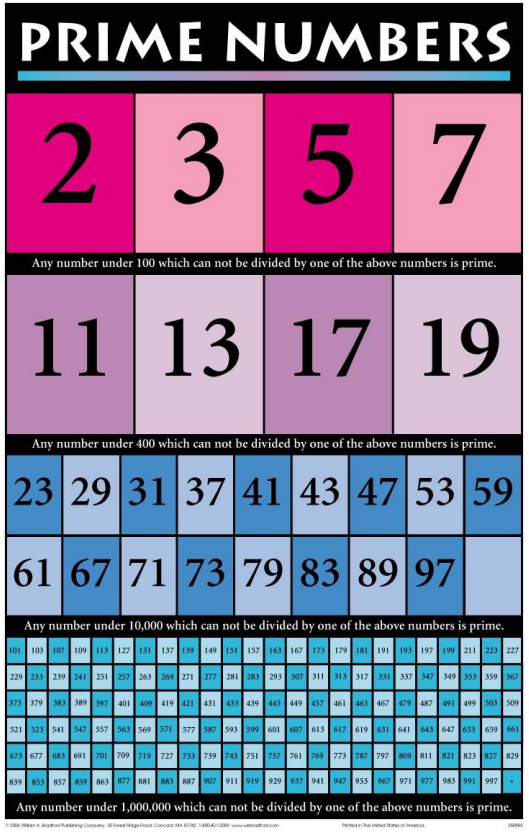
M1 Python程序设计基础

4 函数

张 莉

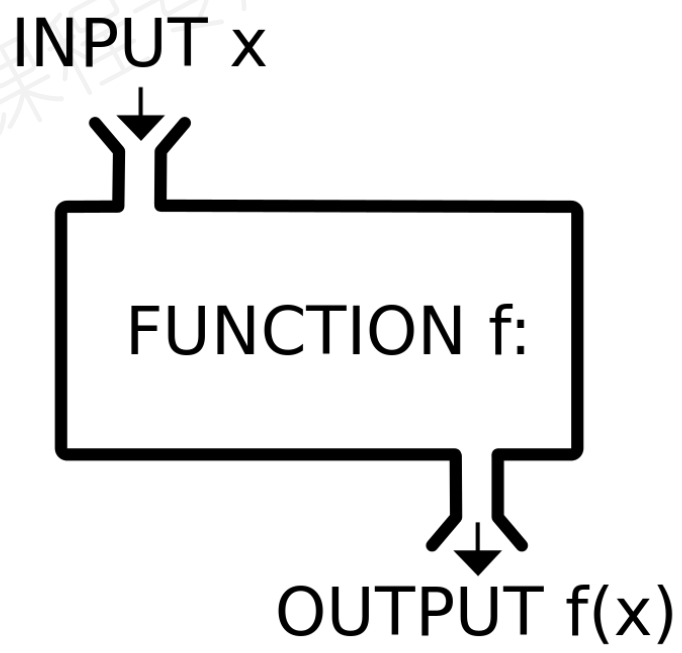
找前5个默尼森数

- 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。

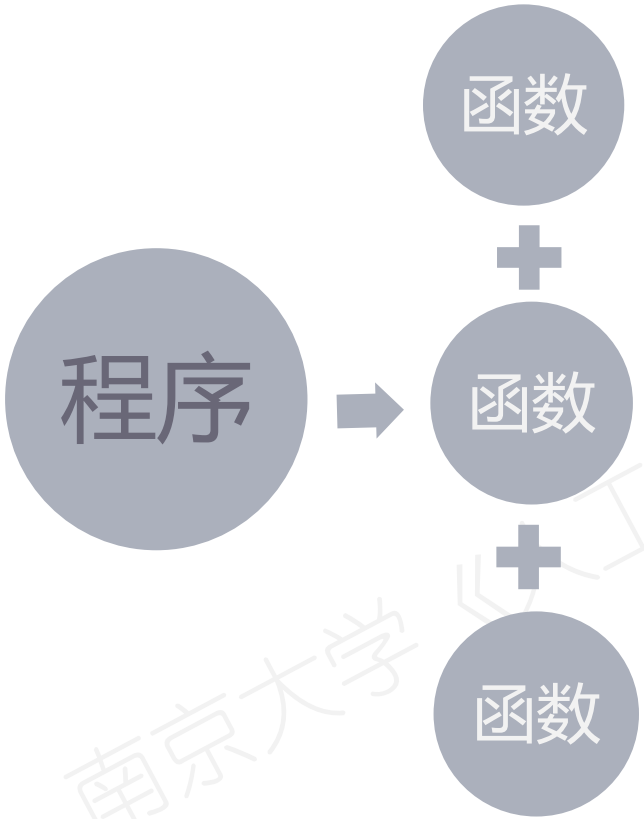


函数

- 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



函数

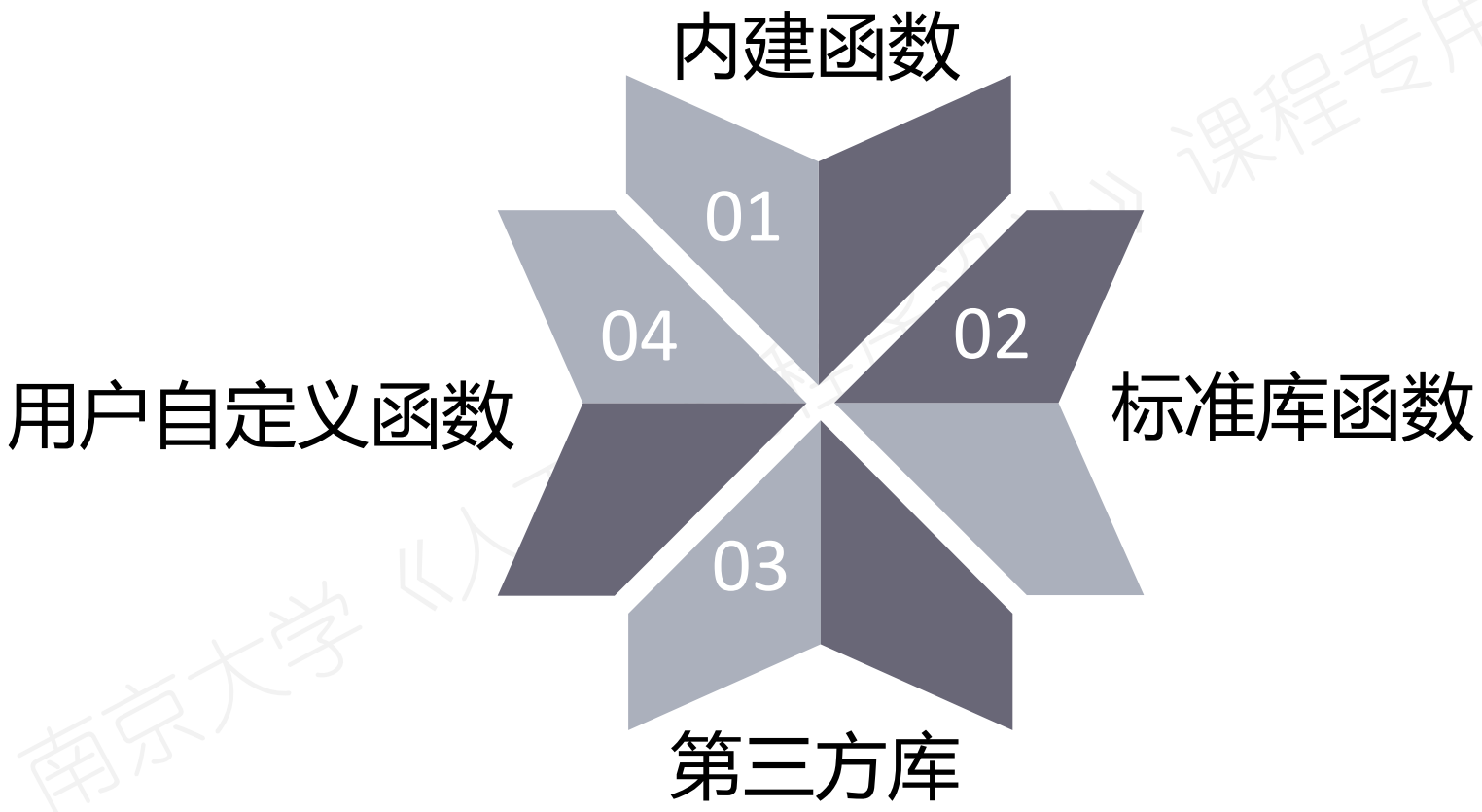


简化程序结构

降低程序开发和修改复杂度

提高程序可读性

Python中的函数



函数

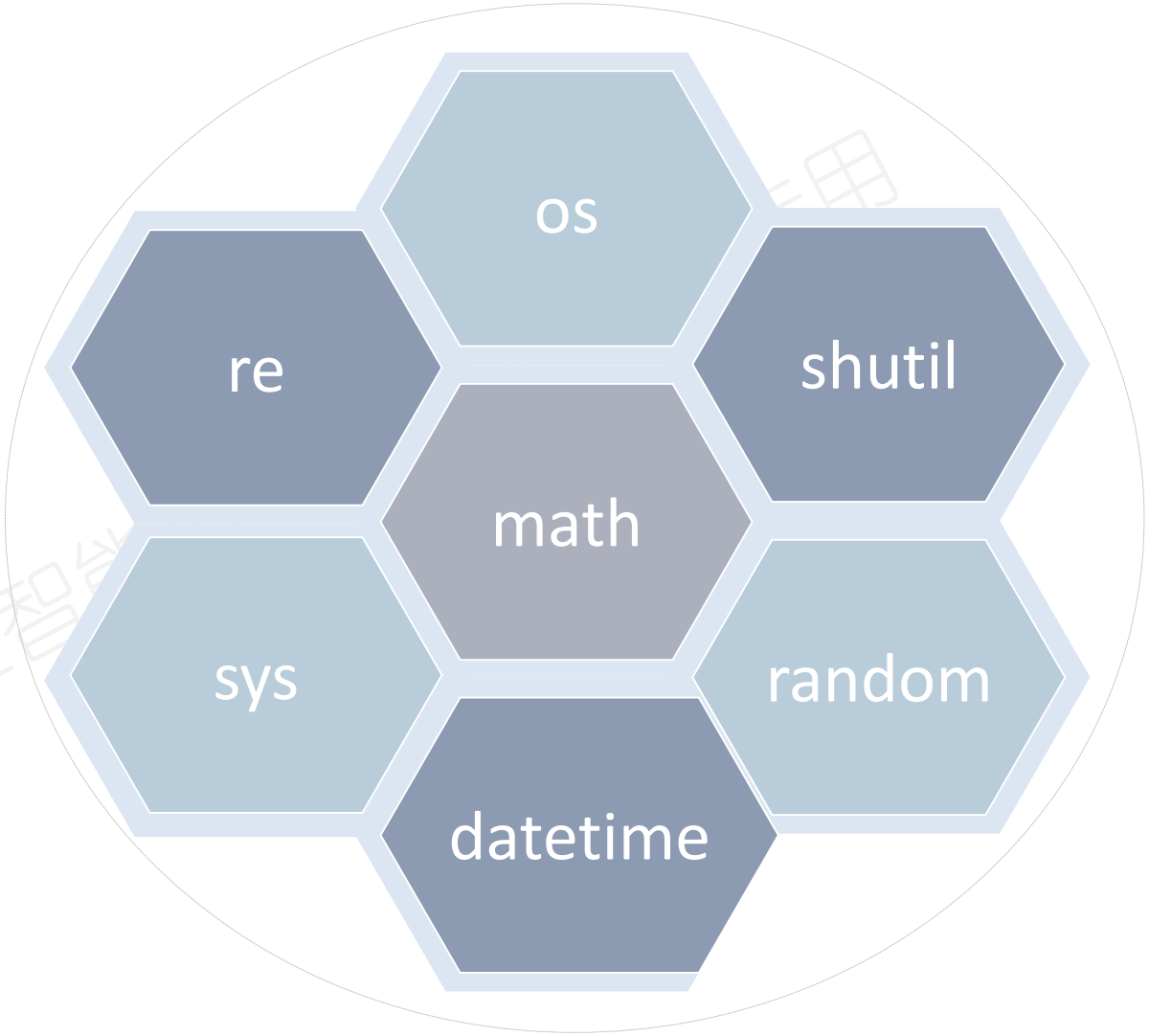
1. 常用Python标准库函数
2. 自定义函数
3. 函数的参数
4. 递归函数
5. lambda函数与函数式编程
6. 变量作用域

1

人工智能程序设计

常用PYTHON标准库函数

常用Python标准库函数



os模块中常用的处理文件及目录的函数

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

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

sys模块中标准输入和输出属性

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

Source

```
>>> 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
```

sys.agrv命令行参数

random模块中常用函数的功能和使用方法

Source

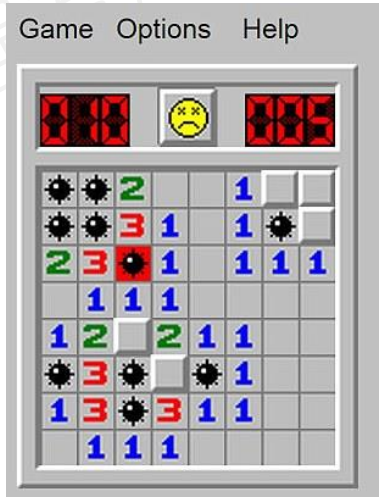
```
>>> import random
>>> random.seed(100)
>>> random.random()
0.1456692551041303
>>> random.random()
0.3885991408219421 4
>>> random.choice(['C++', 'Java', 'Python'])
'Java'
>>> random.randint(1, 100)
37
>>> random.randrange(0, 10, 2)
 4
>>> random.uniform(5, 10)
5.776718084305783
```

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

random模块中常用函数的功能和使用方法

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'
```

生成符合要求的学号

要求：

1. 函数func()的功能是利用班级信息的字典数据随机选择班级并生成一个随机的学号。注意：学号共有6位，前4位为班级编号，后2位为某同学在班级中的序号，如A00101，序号从01开始顺序编号，并且不能超过该班学生总数；
2. 主模块中包含班级信息字典，调用func()生成10个不重复的学生学号并输出。其中，班级信息字典的键为班级编号，值为对应班级的学生总数。例如，当给定的班级信息为data={"A001":32,"A002":47,"B001":39,"B002":42}时，表示A001班共有32位同学，依此类推。

生成符合要求的学号

```
import random

def func(data):
    cls_no = random.choice(list(data.keys()))
    stu_no = random.randint(1, data[cls_no])
    return "{}{:02}".format(cls_no, stu_no)

data = {"A001":32, "A002":47, "B001":39, "B002":42}
result = set()
while len(result) < 10:
    result.add(func(data))
print(result)
```

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

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

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

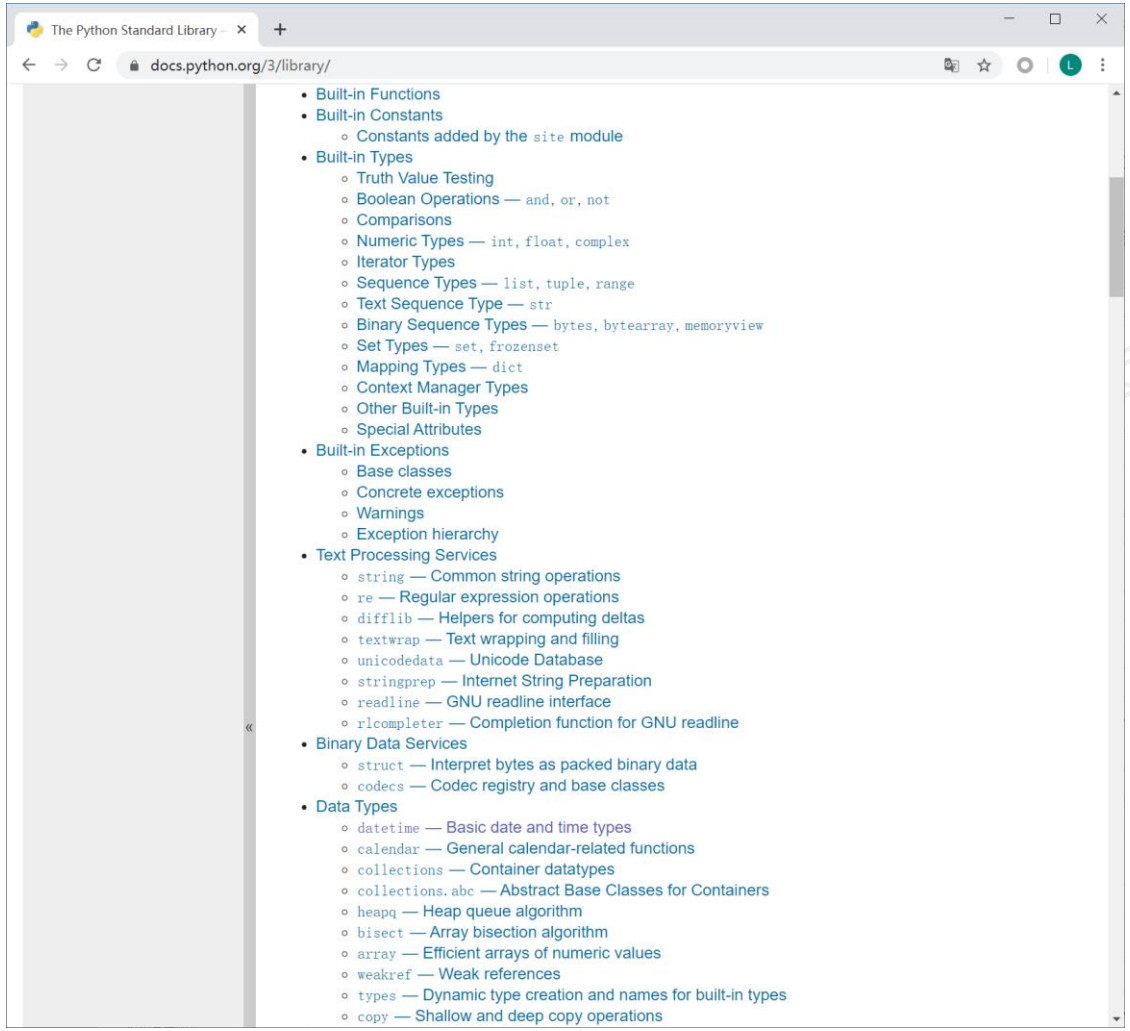
year | month | day | hour | minute | second

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



参考链接：
https://docs.python.org/3/library/

人工智能程序设计

2 自定义函数

自定义函数的创建

Source

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

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

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

函数的返回

Source

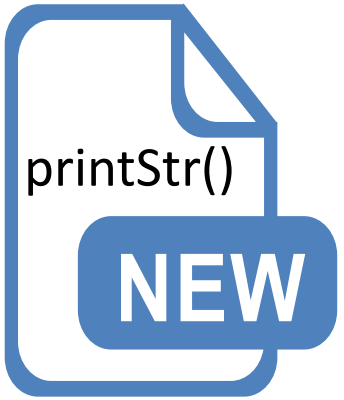
```
>>> def square(x, y):
        '''
        计算参数的和与差
        '''
        return x + y, x - y
```

return 表达式1, 表达式2, ..., 表达式n

函数的调用

```
>>> printStr('Hi, Python!')
Hi, Python!
>>> x, y = square(3, 5)
>>> x
8
>>> y
-2
```

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



pStr.py

例 求2~100之间的所有素数

- 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
import math

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

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

例 求2~100之间的所有素数

File

```
# Filename: prime.py
import math

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

File

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

Python中的__main__函数

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

File

```
#test.py
def foo(x):
    return x * x

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

Source

```
>>> 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(5))
25
```

Python中的main函数

File

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

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

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

例 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”。

例 字符串循环移动

```
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]
```

```
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

File

```
# Filename: account.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:
        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

File

```
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.get(user_name) != password:
            count += 1
            if count >= 3 :
                print("Bye - bye")
                break
            password = input("Wrong password, please input again: ")
        if account.get(user_name) == password:
            print("Login success!")
```

例 模拟一个简易的用户注册和登录系统

File

```
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

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
```

人工智能程序设计

3 函数的参数

1. 位置参数

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.
```

Source

```
>>> 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.
```

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

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

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

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

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

3. 默认参数

Source

```
>>> 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

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—可变长位置参数

- 允许传递一组数据给一个形参，形参前“*”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

Source

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

4. 可变长参数—可变长位置参数

Source

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

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。

Source

```
>>> 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

Source

```
>>> def greeting(x, *args, **kwargs):
        print(x)
        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 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。

File

```
# Filename: register.py
def register(name, gender, phonenum, **otherinfo):
    ''' register users information '''
    print('name: ', name, 'gender: ', gender, 'phone num: ', phonenum)
    print('other information: ', otherinfo)
```

例 实现用户信息注册登记

Source

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

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'}
```

元素求和

- 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.
```

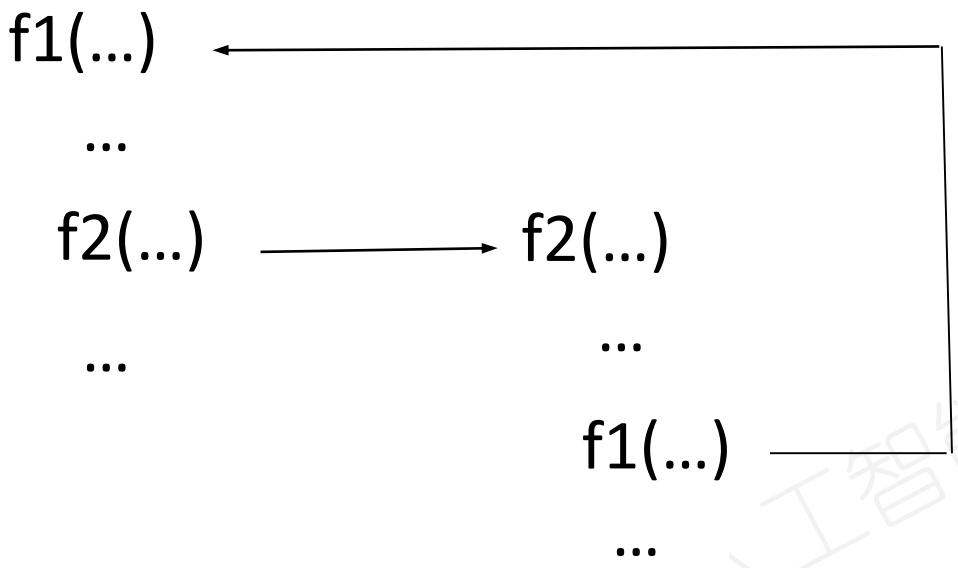
```
sorted(iterable, /, *, key=None, reverse=False)
```

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

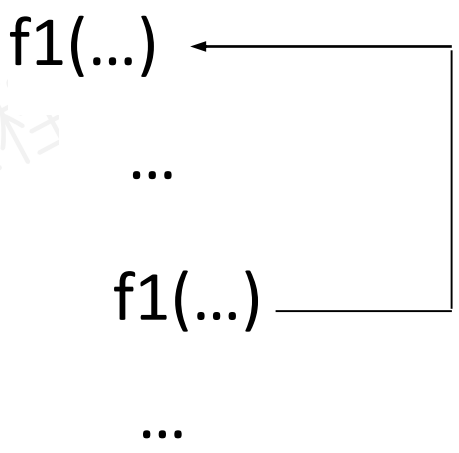
人工智能程序设计

4 递归函数

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

Source

```
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。

例 编写递归函数计算n的阶乘

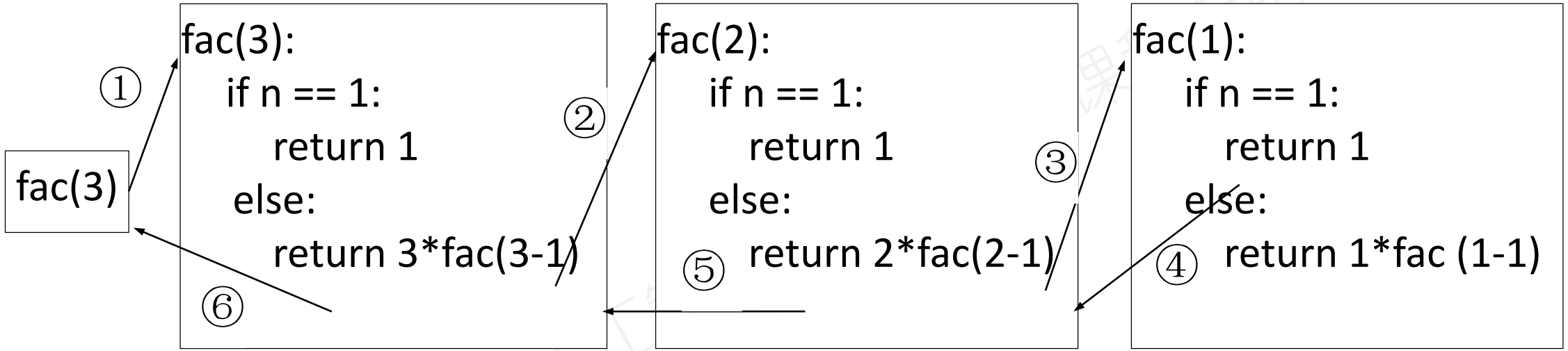
$$n! = \begin{cases} 1 & (\text{当} n=1) \\ n \times (n-1)! & (\text{当} n>1) \end{cases}$$

n的阶乘的定义是一种递归定义

File

```
# Filename: fac.py
def fac(n):
    if n == 1:
        return 1
    else:
        return n * fac(n–1)
```

例 编写递归函数计算n的阶乘



阅读程序1

```
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

阅读程序2

```
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)
```

5

人工智能程序设计

LAMBDA函数和函数式编程

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
```

lambda函数

Source

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

Source

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

例 编写函数计算平均成绩—lambda函数

Source

```
>>> 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 编写函数计算平均成绩——lambda函数

File

```
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]
```

例 寻找数字朋友组

File

```
# 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])
for item in lst:
    itemTemp = item[1]
    itemTemp.sort()
    result.append(itemTemp)
return result
```

那些Python中的一行代码

```
foo = lambda array: array if len(array) <= 1 else foo([item for item in array[1:] if item <= array[0]]) + [array[0]] + foo([item for item in array[1:] if item > array[0]])
```

sorting

函数式编程

- 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
```

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]))
```

6

人工智能程序设计

变量的作用域

变量作用域

Source

```
>>> 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

- 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

Source

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

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

Source

```
>>> 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

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

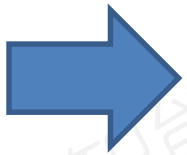
Python装饰器（decorator）

```
def f1():
    print('f1 was called')

def f2():
    print('f2 was called')

...
```

增加计时功能



```
import time
def f1():
    start_time = time.process_time()
    print('f1 was called')
    end_time = time.process_time()
    print('time=', end_time-start_time)

def f2():
    start_time = time.process_time()
    print('f2 was called')
    end_time = time.process_time()
    print('time=', end_time-start_time)
```

Python装饰器

- 功能：拓展原函数功能的一种函数，这个函数的返回值也是一个函数
- 优势：不更改原函数代码给函数增加新的功能
- 应用：缓存、日志和权限校验等

@ decorators

```
import time

def deco(fun):
    def wrapper():
        start_time = time.process_time()
        fun()
        end_time = time.process_time()
        print('time=', end_time-start_time)

    return wrapper

@deco
def f1():
    print('f1 was called')

@deco
def f2():
    print('f2 was called')

f1()
f2()
```

原功能为输出参数和，写一个打印参数乘积的装饰器

```
def deco(fun):
    def wrapper(a, b):
        fun(a, b)
        print(a*b)

    return wrapper

@deco
def f1(a, b):
    print(a+b)

@deco
def f2(a, b):
    print(a+b)

f1(3,4)
f2(5,8)
```

*args

```python
from functools import wraps

def logit(func):
    @wraps(func)
    def with_logging(*args, **kwargs):
        print(func.__name__ + " was called")
        return func(*args, **kwargs)
    return with_logging

@logit
def addition_func(x):
   """Do some math."""
   return x + x


result = addition_func(4)
# Output: addition_func was called
```

- https://realpython.com/primer-on-python-decorators/
- https://python101.pythonlibrary.org/chapter25_decorators.html

- “Logging is another area where the decorators shine”
- https://book.pythontips.com/en/latest/decorators.html

M1.4 小结

01 常用Python标准库函数

02 自定义函数

03 函数的参数

04 递归函数

05 lambda函数与函数式编程

06 变量作用域