Python函数

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

资料整理
火龙果软件
荐

参考书目《Python 程序设计》

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1 定义函数

函数是组织好的，可重复使用的，用来实现单一或相关联功能的代码段，它能够提高应用的模块化和代码的重复利用率。

Python安装包、标准库中自带的函数统称为内置函数，用户自己编写的函数称为自定义函数，不管是哪种函数，其定义和调用方式都是一样的。

资料整理
火龙果软件
荐

1 定义函数

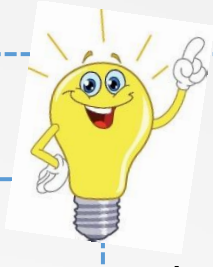
函数要有三个重要元素：函数名，以及函数参数和返回值。

定义函数的基本格式如下：

```
def 函数名(参数列表):
    '''函数注释字符串'''
    函数体
```

【例】定义一个打印信息的函数

```
def printInfo():
    '''定义一个函数，能够完成打印信息的功能
。
'''
    print('------------------------------------')
    print(' 不忘初心，牢记使命 ')
    print('------------------------------------')
```



- 如果有多个参数，各参数间使用逗号“,”分隔；函数没有参数时也必须保留一对空的小括号“()”；
- 如果函数有返回值，可以使用return语句返回；不带表达式的return相当于返回None；
- 函数体和注释相对于def关键字必须保持一定的缩进。

资料整理 火龙果软件 荐

2 调用函数

定义了函数之后，就相当于有了一段具有某些功能的代码，想要让这些代码能够执行，需要调用它。

函数名称([函数参数])

【例】调用5.2.1节中的printInfo函数。

```
printInfo ()
运行结果：
-----------------------------------
 不忘初心，牢记使命
-----------------------------------
```

函数名称：要调用的函数名称必须是已经创建好的。

可选参数：如果需要传递多个参数值，则各参数值间使用逗号“,”分隔。如果该函数没有参数，则直接写()。

资料整理 火龙果软件 荐

3 函数的返回值

“返回值”，就是程序中的函数完成一件事情后，最后给调用者的结果

return [value]

【例】return语句应用示例：定义一个两个数相加的函数

```
#定义函数
def add2num(a, b):
    c=a+b
    return c
    print("函数运行中")
#调用函数
sumValue = add2num(2,3)
print(sumValue)
```

运行结果：5

return作用：

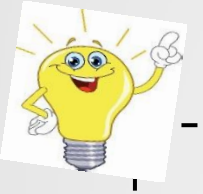
1. 负责函数返回值。
2. 退出当前函数：导致return下方的所有代码（函数体内部）不执行。

资料整理 火龙果软件 荐

3 函数的返回值

一般情况下，每个函数都有一个return语句，如果函数没有定义返回值，那么返回值就是None.

返回值个数	返回值类型
0	None
1	Object
大于1	Tuple



如果一个函数要有多个返回值，可以采取“return a, b”写法，返回多个数据的时候，默认是元组类型。return后面也可以连接列表、元组或字典，以返回多个值。

资料整理 火龙果软件 荐

4 函数的嵌套调用

在一个函数中调用了另外一个函数，这就是函数嵌套调用。其执行流程为如果函数A中，调用了另外一个函数B，则执行函数B中的任务后再回到上次函数A执行的位置。

【例】函数的嵌套调用示例。

```
#计算三个数之和
def sum_num(a, b, c):
    return a + b + c

#求三个数平均值
def average_num(a, b, c):
    sumResult = sum_num(a, b, c)
    return sumResult / 3

result = average_num(1, 2, 3)
print(result)
```

运行结果：

2.0

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1 函数的形参和实参

形式参数	实际参数
➢ 在定义函数时，函数名后面括号中的参数 ➢ 在函数内部使用的，函数外部并不可见。	➢ 在调用一个函数时，函数名后面括号中的参数 ➢ 函数的调用者提供给函数的参数

值传递和引用传递的基本区别：

- ➢ 进行值传递后，改变形式参数的值，实际参数的值不变
- ➢ 进行引用传递后，改变形式参数的值，实际参数的值也一同改变。

资料整理 火龙果软件 荐

4 默认参数

【例】定义一个名称为printString的函数，然后为printString函数传递一个字符串类型的变量作为参数（代表值传递），并在函数调用前后分别输出该字符串变量，再为printString函数传递列表类型的变量作为参数（代表引用传递），并在函数调用前后分别输出该列表。

```
#定义函数
def printString(obj):
    print("原值：",obj)
    obj += obj
#调用函数
print("------------值传递------------")
strslogan = "不忘初心，牢记使命"
print("函数调用前：",strslogan)
printString(strslogan)  #采用不可变对象：字符串
print("函数调用后：",strslogan)
print("------------引用传递------------")
listslogan = ["不忘初心","牢记使命"]
print("函数调用前：",listslogan)
printString(listslogan) #采用可变对象：列表
print("函数调用后：",listslogan)
```

运行结果：

```
------------值传递------------
函数调用前： 不忘初心，牢记使命
原值： 不忘初心，牢记使命
函数调用后： 不忘初心，牢记使命
------------引用传递------------
函数调用前： ['不忘初心','牢记使命']
原值： ['不忘初心','牢记使命']
函数调用后： ['不忘初心','牢记使命','不忘初心','牢记使命']
```

资料整理 火龙果软件 荐

2 位置参数

位置参数也称必备参数，调用函数时根据函数定义的参数位置来传递参数。

数量必须与定义时一致

位置必须与定义时一致

```
division(6, 2)
```

```
def division(num_one, num_two):
    print(num_one / num_two)
```

资料整理 火龙果软件 荐

3 关键字参数

关键字参数是指使用形式参数的名字来确定输入的参数值。函数调用，通过“键=值”形式加以指定。

【例】关键字参数应用示例。

```
def user_info(name, age, gender):
    print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')
user_info('Rose', age=20, gender='女')
user_info('huayi', gender='男', age=6)
```

运行结果：

```
您的名字是Rose, 年龄是20, 性别是女
您的名字是huayi, 年龄是6, 性别是男
```

资料整理 火龙果软件 荐

4 默认参数

定义函数时可以指定形式参数的默认值。调用函数时，可分为以下两种情况：

未给默认参数传值	给默认参数传值
使用参数的默认值	使用实际参数的值

默认值放最后

```
def   函数名(…,[参数n=默认值]):
      "函数注释字符串"
      函数体
```

资料整理 火龙果软件 荐

4 默认参数

【例】函数定义时设置默认参数，调用时验证其功能。

```
def printinfo( name, age = 100 ):
  # 打印任何传入的字符串
  print("Name:", name)
  print("Age:", age)
# 调用printinfo函数
printinfo(name="cau" )
printinfo(name="cau",age=110 )
```

运行结果：

```
Name: cau
Age: 100
Name: cau
Age: 110
```

资料整理 火龙果软件 荐

5 不定长可变参数

不定长参数也叫可变参数。用于不确定调用的时候会传递多少个参数(不传参也可以)的场景。

```
def 函数名([formal_args,] *args, **kwargs):
        "函数_文档字符串"
        函数体
        [return语句]
```

*args和**kwargs都是不定长参数，它们可搭配使用，亦可单独使用。

资料整理 火龙果软件 荐

5 不定长可变参数

1.* args形式

这种形式表示接收任意多个实际参数并将其放到一个元组中。

```
def test(*args):
    print(args)
```

```
test(1, 2, 3, 'a', 'b', 'c')
```

(1, 2, 3, 'a', 'b', 'c')

资料整理 火龙果软件 荐

5 不定长可变参数

2. kwargs形式**

****kwargs这种形式表示接收任意多个类似关键字参数一样显式赋值的实际参数，并将其放到一个字典中。**

```
def test(**kwargs):
    print(kwargs)
```

```
test(a=1, b=2, c=3, d=4)
```

{'c': 3, 'd': 4, 'a': 1, 'b': 2}

资料整理 火龙果软件 荐

6 可变参数的装包与拆包

装包：把未命名参数和命名参数分别放在元组或者字典中。

拆包：将一个序列类型的数据拆开为多个数据，分别赋值给变量，位置对应。

*args参数	**kwargs参数
- 用于接收多余的未命名参数 - 元组类型	- 用于接收形参中的命名参数 - 字典类型

资料整理
火龙果软件
荐

6 可变参数的装包与拆包

【例】参数*agrs装包与拆包过程示例。

```
def run(a, *args):
    # 第一个参数传给了a
    print(a)
    # args是一个元组，里面是第2和3两个参数
    print(args)
    # *args是将这个元组中的元素依次取出来
    print("对args拆包")
    print(*args)  # *args 相当于 a,b = args
    print("将未拆包的数据传给run1")
    run1(args)
    print("将拆包后的数据传给run1")
    run1(*args)
```

```
def run1(*args):
    print("输出元组")
    print(args)
    print("对元组进行拆包")
    print(*args)


run('Rose','Tome', 'King')
```

资料整理 火龙果软件 荐

6 可变参数的装包与拆包

【例】参数**kwargs装包与拆包过程示例。

```
def run(**kwargs): # 传来的 key = value 类型的实参会映射成kwargs里面的键和值
    # kwargs是一个字典，将未命名参数以键值对的形式
    print(kwargs)
    print("对kwargs拆包:")
    # 此处可以把**kwargs理解成对字典进行了拆包，{"k":2,"v":4}的kwargs字典又
    # 被拆成了k=2,v=4传递给run1,但是**kwargs是不能像之前*args那样被打印出来看的
    run1(**kwargs)
    # print(**kwargs)
def run1(k, v): # 此处的参数名一定要和字典的键的名称一致
    print(k, v)
run(k=2,v=4)
```

运行结果：

```
{'k': 2, 'v': 4}
对kwargs拆包:
2 4
```

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1 LEGB原则

变量的作用域

变量的作用域是指变量的作用范围。根据作用范围，Python中的变量分为局部变量与全局变量。

“LEGB”即Python中四层作用域范围如下，：

第一层是L(local)，表示变量在一个函数定义中，而且在这个函数里面没有再包含函数的定义

第二层是E(enclosing function)，表示变量在一个函数定义中，但这个函数里面还包含有函数的定义

第三层是G(global)，表示一个模块的命名空间，也就是说变量在一个.py文件中，且在函数或类外构成的一个空间，这一层空间对应的全局范围

第四层是B(builtin)，表示变量在Python解释器启动时就已经加载到当前编程环境中的范围

变量是采用L—>E—>G –>B的规则查找，即先是在局部中查找，如果找不到，便去局部外的局部找（例如闭包），再找不到就会去全局找，再者去内建命名空间中找。

资料整理 火龙果软件 荐

2 全局变量和局部变量

变量的作用域

1.局部变量

局部变量是指在函数内部定义并使用的变量，它只在函数内部有效。作用：在函数体内部，临时保存数据，即当函数调用完成后，则销毁局部变量。

【例】局部变量的使用

```
def test1():
    num=100
    print("test1中的num值为：%d"%num)
def test2 () :
    num=200
    print("test2中的num值为：%d"%num)
#函数调用
test1()
test2()
```

运行结果：

```
test1中的num值为：100
test2中的num值为：200
```

资料整理 火龙果软件 荐

2 全局变量和局部变量

变量的作用域

2.全局变量

全局变量可以在整个程序范围内访问。有以下两种情况：

(1)在函数外定义变量

如果一个变量在函数外定义，那么不仅可在函数外可以访问，在函数内也可以访问到。

```
result=100                    #全局变量
def  sum (a, b) :
    result=a+b                #局部变量
    print("函数内的result的值为： ",result)  #result在这里是局部变量
    return  result
#调用sum函数
sum(100, 200)
print("函数外的变量result是全局变量，等于",result)
```

运行结果：

```
函数内的result的值为： 300
函数外的变量result是全局变量，等于 100
```

资料整理 火龙果软件 荐

2 全局变量和局部变量

(2)在函数体内定义变量

在函数体内定义，并且使用global关键字修饰后，该变量也就变为全局变量。在函数体内还可以对其进行修改。

【例】global关键字的应用

```
a=100
def test():
    global a
    a+=100
    print (a)
test()
```

运行结果：200

在函数内部定义的变量即使与全局变量重名，也不影响全局变量的值。

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1 递归函数

如果一个函数在内部不调用其他的函数，而是调用自身的话，这个函数就是递归函数。

【例5-19】应用递归函数计算5！

reduce(lambda x, y: x + y,[1,2,3,4])

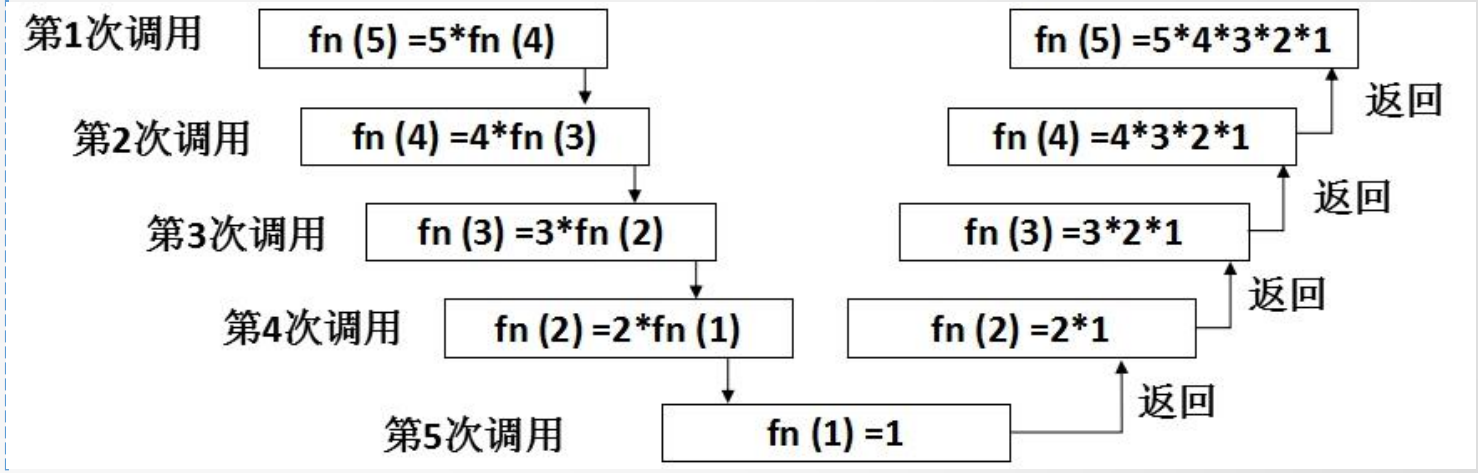
执行步骤：

先将1，2传入：1+2 = 3

再将3，3传入：3+3 = 6

再将6，4传入：6+4 = 10

最终结果为：10



资料整理 火龙果软件 荐

1 递归函数

递归函数具有如下特征：

递归回归

递归的递推（调用）和回归（返回）过程，跟入栈和出栈类似

结束条件

结束条件即终止条件，用于终止递归。

资料整理 火龙果软件 荐

2 匿名函数

匿名函数（lambda)是指没有名字的函数，即不再使用def语句定义的函数。通常情况下，这样的函数只使用一次。

```
result = lambda [arg1 [,arg2,.., argn]]:expression
```

【例】匿名函数的应用。

```
sum = lambda arg1, arg2: arg1 + arg2
# 调用sum函数
print("运行结果：", sum(10, 20))
print("运行结果：", sum(20, 20))
```

```
运行结果：
运行结果： 30
运行结果： 40
```

用途

- 指定短小的回调函数
- 作为内置函数的参数

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

高阶函数

高阶函数就是一个函数可以用来接收另一个函数作为参数，这样的函数叫做高阶函数。

利用函数定义调用方式，求和的方法1

```
def add_num(a, b):
    return abs(a) + abs(b)

result = add_num(-1, 2)
print(result)
```

运行结果：

3

利用定义高阶函数，求和的方法2

```
def sum_num(a, b, f):
    return f(a) + f(b)

result = sum_num(-1, 2, abs)
print(result)
```

运行结果：

3

方法2的代码会更加简洁，函数灵活性更高。函数式编程大量使用函数，**减少了代码的重复**。

资料整理 火龙果软件 荐

1 内置高阶函数：map()

map函数：根据提供的函数对指定的序列做映射。将传入的函数变量func作用到iterables变量的每个元素中，并将结果组成新的迭代器返回。

map(func, *iterables) --> map object

【例】对于list [1,2,3,4,5,6,7,8,9]的每个元素都作平方，只需要向map()函数传入函数f(x)=x*x即可。

```
# 1. 准备列表数据
list_01 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# 2. 准备2次方计算的函数
def f(x):
    return x*x
# 3. 调用map
result = map(f,list_01 )
```

```
# 4. 验收成果
print(list_01)
print(result)
print(list(result))
```

运行结果：

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
<map object at 0x00000000021DF668>
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

资料整理 火龙果软件 荐

2内置高阶函数：reduce()

reduce()函数：实现对func计算的结果继续和序列的下一个元素做累积计算。

reduce(function, iterable[, initializer])

用传给reduce中的函数 func()（必须是一个二元操作函数）先对集合中的第1，2个数据进行操作，得到的结果再与第三个数据用func()函数运算。

reduce(lambda x, y: x + y,[1,2,3,4])执行步骤：

先将1，2传入：1+2 = 3

再将3，3传入：3+3 = 6

再将6，4传入：6+4 = 10

最终结果为：10

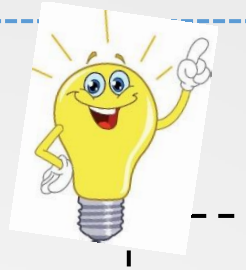
资料整理 火龙果软件 荐

2内置高阶函数：reduce()

【例】reduce()的应用示例

```
# 1. 导入模块
from functools import reduce
# 2. 定义功能函数
def myadd(x,y):
    return x+y
# 3. 调用reduce，作用：功能函数计算的结果和序列的下一个数据做累计计算
sum=reduce(myadd,(1,2,3,4,5,6,7))
print(sum)
```

运行结果：28



reduce（）函数被放置在fucntools 模块里，使用前要先引入模块

资料整理 火龙果软件 荐

3 内置高阶函数：filter()

fliter()函数：对指定的序列执行过滤操作。其接收一个函数 f 和一个 list，这个函数 f 的作用是对每个元素进行判断，返回 True或 False，filter()根据判断结果自动过滤掉不符合条件的元素。

filter(function, iterable) -> filter object

【例5-25】要从一个list [1, 4, 6, 7, 9, 12, 17]中删除偶数，保留奇数。

```
list1 = [1, 4, 6, 7, 9, 12, 17]
# 1. 定义功能函数：过滤序列中的偶数
def is_odd(x):
    return x % 2 == 1
# 2. 调用filter，利用filter()过滤掉偶数
print(list(filter(is_odd, list1)))
```

运行结果：

[1, 7, 9, 17]

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1函数的引用

采用不同的方式函数的调用，所起到的功能也不同。

```
def infoout():
    print("Work hard!")

infoout()
print(infoout)
```

运行结果：

```
Work hard!
<function infoout at 0x00000000020AC268>
```

使用“infoout()”和“infoout”是不同的，前者表示调用infoout()函数，执行函数中的代码，而后者是引用的函数块

资料整理 火龙果软件 荐

2闭包概述

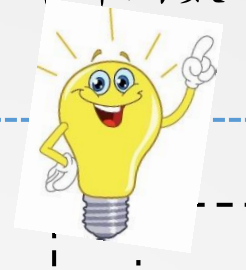
若每次在某变量的基础上和其它数字进行求和计算，怎么办?

闭包可解决这个需求。闭包条件：

1）在函数嵌套(函数里面再定义函数)的前提下。

2）内部函数使用了外部函数的变量(还包括外部函数的参数)。

3）外部函数返回了内部函数。



inner函数就是一个闭包

```
#1.函数嵌套
def outer(num1):
        #外部函数
    def inner(num2):
                #内部函数
                #2.内部函数必须使用了外部函数的变量
        print(num1+num2)
        #外部函数返回了内部函数，这里返回的内部函数就是闭包
    return inner
```

资料整理 火龙果软件 荐

2闭包概述


```
print(outer(10))
# 创建闭包实例
outer_new = outer(10)
#执行闭包
outer_new(20)
```

运行结果：

```
<function outer.<locals>.inner at 0x00000000021E66A8>
30
```

资料整理
火龙果软件
荐

3闭包的应用

闭包主要在面向对象、装饰器以及实现单例模式三个方面应用居多。

【例5-28】利用闭包计算函数式：y=ax+b的函数值。

```
def calc_y(a,b):
    def calc(x):
        print(a*x+b)
    return calc

calc_1 = calc_y(2,5)
calc_2 = calc_y(6,8)
calc_1(2)
calc_2(3)
```

运行结果：

```
9
26
```

闭包具有**提高代码可复用性**的作用

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1装饰器的概念

python装饰器（Fuctional Decorators）是用于拓展原来函数功能的一种函数，目的是在不改变原函数名(或类名)的情况下，给函数增加新的功能

。【例】利用装饰器实现统计函数执行时间。

```
import time
def timeit(func):
    def result():
        start_time = time.time()
        func()
        end_time = time.time()
        print('函数运行时间为：%.2fs' %
(end_time - start_time))
    return result
```

```
@timeit
def func_0():
    time.sleep(3)
# 省略4个
@timeit
def func_5():
    time.sleep(3)
#调用函数
func_0()
func_5()
```

```
运行结果：
函数运行时间为：3.00s
函数运行时间为：3.00s
```

资料整理 火龙果软件 荐

2装饰器的应用

1.无参数的装饰器应用

【例】利用测试函数运行时间的应用，演示无参数的装饰器应用。

```
import time
# 装饰器
def timeit(func): # func为装饰器绑定的方法(绑定装饰器后自动传入)
    def result():
        start_time = time.time()
        func() # 调用test方法
        end_time = time.time()
        print('函数运行时间为：%.2fs' %(end_time - start_time))
    return result # 返回result方法
```

```
@timeit # 添加装饰器
def test():
    time.sleep(1)
    print("函数运行测试")

test() # 调用函数
```

运行结果：
函数运行测试
函数运行时间为：1.00s

资料整理 火龙果软件 荐

2装饰器的应用

2.有参数的装饰器应用

【例】利用测试函数运行时间的应用，演示有参数的装饰器应用。

```
import time

def timeit(func): # func为装饰器绑定的方法(绑定装饰器后自动传入)
    def result(arg1): # 传入test方法的参数
        start_time = time.time()
        func(arg1) # 调用test方法
        end_time = time.time()
        print('函数运行时间为：%.2fs' %(end_time - start_time))

    return result # 返回result方法
```

```
@timeit # 添加装饰器
def test(mStr):
    time.sleep(2)
    print("函数运行时间:" + mStr)

test("传入参数") # 调用函数
```

运行结果：
函数运行时间:传入参数
函数运行时间为：2.00s

资料整理 火龙果软件 荐

2装饰器的应用

3.装饰器既可以装饰带参函数也可以装饰不带参函数

【例】利用测试函数运行时间的应用，演示有或无参数的装饰器应用

```
import time
# 装饰器
def timeit(func): # func为装饰器绑定的方法(绑定装饰器后自动传入)
    def result(*arg1, **kwargs): # (传入非固定参数)这样即使装饰函数不带参数也可被装饰
        start_time = time.time()
        func(*arg1, **kwargs)
        end_time = time.time()
        print('函数运行时间为：%.2fs' % (end_time - start_time))
    return result # 返回result方法
```

```
@timeit # 添加装饰器
def test():
    time.sleep(2)
    print("函数运行")

test() # 调用函数
```

运行结果：
函数运行
函数运行时间为：2.00s

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

1迭代器的概念

1. 迭代

迭代是通过重复执行的代码处理相似的数据集的过程，并且本次迭代的处理数据要依赖上一次的结果继续往下做，上一次产生的结果为下一次产生结果的初始状态，如果中途有任何停顿，都不能算是迭代。

【例】迭代应用示例。

```
loop = 0
while loop < 3:
    print(loop)
loop += 1
```

运行结果：

```
0
1
2
```

资料整理 火龙果软件 荐

1迭代器的概念

2. 容器

容器是一种把多个元素组织在一起的数据结构，容器中的元素可以逐个地迭代获取，可以用in，not in关键字判断元素是否包含在容器中。

3. 可迭代对象(Iterable)

可迭代对象并不是指某种具体的数据类型，它是指存储了元素的一个容器对象，且容器中的元素可以通过__iter__( )方法或__getitem__( )方法访问。

资料整理
火龙果软件
荐

1迭代器的概念

4. 迭代器（Iterator）

迭代器可以看作是一个特殊的对象，每次调用该对象时会返回自身的下一个元素。一个迭代器对象必须是定义了__iter__()方法和next()方法的对象。

__iter__( )方法	__next__( )方法
让对象可以用for ... in循环遍历时找到	让对象可以通过next(实例名)访问下一个元素。

迭代器优点：节约内存、不依赖索引取值、实现惰性计算

资料整理 火龙果软件 荐

2迭代器的应用

1. 用for...in方式访问迭代器

```
vList=[1,2,3,4]
vIter=iter(vList) #从列表生成迭代器对象
for i in vIter:
    print('第一次：',i)  #输出迭代器中的数据1、2、3、4
for i in vIter:
    print('第二次:',i) #再次输出没有数据，因为迭代器已经空了
```

资料整理 火龙果软件 荐

2迭代器的应用

2. 用next方式访问

```
s = {'唐僧','悟空','悟能','悟净'}
it = iter(s)
try:
    while True:
        x = next(it)
        print(x)
except StopIteration:
    print('遍历结束')
```

资料整理 火龙果软件 荐

目录

- 函数的定义和调用
- 函数的参数与值传递
- 变量的作用域
- 递归函数和匿名函数
- 高阶函数
- 闭包及其应用
- 装饰器及其应用
- 迭代器及其应用
- 生成器及其应用

这里说明
图片内容

资料整理
火龙果软件
荐

生成器的概念

生成器是能够动态提供数据的可迭代对象。

调用生成器函数将返回一个生成器对象，不执行函数体。yield翻译为“产生”或“**生成**”。

定义	含有yield语句的函数，返回值为生成器对象
语法	def 函数名(): … yield 数据 …
调用	for 变量名 in 函数名(): 语句

资料整理 火龙果软件 荐

生成器的概念

执行过程

1）调用生成器函数会自动创建迭代器对象。
2）在调用迭代器对象的__next__()方法时才执行生成器函数。
3）每次执行到yield语句时返回数据，暂时离开。
4）待下次调用__next__()方法时继续从离开处继续执行。

生成迭代器对象的规则

1）将yield关键字以前的代码放在next方法中。
2）将yield关键字后面的数据作为next方法的返回值。

资料整理 火龙果软件 荐

本章小结

本章主要针对函数进行了讲解函数的作用有提高应用的模块性，最小化代码冗余以及流程分解，内容包括**函数的定义、函数的调用、函数的参数、函数的返回值，函数的嵌套、递归函数、匿名函数、日期时间函数和随机数函数**。函数作为关联功能的代码段，可以很好地提高应用的模块化，希望大家能用好这些函数，并学会查询相关的函数手册。

资料整理
火龙果软件
荐

思考与练习

1.定义一个getMax函数，返回三个数（从键盘输入的整数）中的最大值。例如：

请输入第1个整数：10

请输入第2个整数：15

请输入第3个整数：20

其中最大值为：20

2.定义能够计算斐波那契数列的函数，并调用。

3.回文数是一个正向和逆向都相同的整数，如123454321、9889。编写函数判断一个整数是否是回文数。

4.编写函数，判断输入的三个数字是否能构成三角形的三条边。

5.编写函数，求两个正整数的最小公倍数。

资料整理
火龙果软件
荐

思考与练习

6.编写一个学生管理系统，要求如下。

1)使用自定义函数，完成对程序的模块化。

2)学生信息至少包含：姓名、性别及手机号。

3)该系统具有的功能：添加、删除、修改、显示、退出系统。

设计思路如下。

1)提示用户选择功能操作。

2)获取用户选择的功能。

3)根据用户的选择，分别调用不同的函数，执行相应的功能。

7.创建装饰器，要求如下：

（1）创建add_log装饰器， 被装饰的函数打印日志信息；

（2）日志格式为: [字符串时间] 函数名: xxx，运行时间：xxx, 运行返回值结果:xxx。

资料整理
火龙果软件
荐