Python 程序设计教案

<table>
<tr><td>本次授课内容</td><td colspan="2">5.1 函数定义与使用
5.2 函数参数
5.3 变量作用域</td></tr>
<tr><td>本次课的
教学目的</td><td colspan="2">掌握函数定义和调用的用法
理解递归函数的执行过程
掌握位置参数、关键参数、默认值参数和长度可变参数的用法
理解函数调用时参数传递的序列解包用法
理解变量作用域</td></tr>
<tr><td>本次课教学
重点与难点</td><td colspan="2">递归函数定义与调用
默认值参数、关键参数、可变长度参数
传递参数时的序列解包
global 的两种作用</td></tr>
<tr><td>教学方法
教学手段</td><td colspan="2">PPT、边讲边练</td></tr>
<tr><td rowspan="3">课堂教学
时间分配</td><td>教学内容</td><td>时间分配（分）</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>课堂教学设计</td><td colspan="2">首先介绍函数定义与调用的基本语法以及递归函数的概念，然后重点介绍位置参数、默认值参数、关键参数、可变长度参数以及传递参数时的序列解包，最后介绍变量作用域和 global 语句的两种作用。</td></tr>
<tr><td>实　验</td><td colspan="2">函数定义与调用，几种参数传递方式的不同</td></tr>
<tr><td>思考题及作业题</td><td colspan="2">课后习题 1-4 题，第 10 题。</td></tr>
<tr><td>备　注</td><td colspan="2"></td></tr>
<tr><td>教学后记</td><td colspan="2"></td></tr>
</table>

课堂重点内容详解	第 一 节 课

在 Python 中，定义函数的语法如下：

```
def 函数名([参数列表]):
    '''注释'''
    函数体
```

其中，def 是用来定义函数的关键字。定义函数时在语法上需要注意的问题主要有：

1）不需要说明形参类型，Python 解释器会根据实参的值自动推断形参类型；

2）不需要指定函数返回值类型，这由函数中 return 语句返回的值来确定；

3）即使该函数不需要接收任何参数，也必须保留一对空的圆括号；

4）函数头部括号后面的冒号必不可少；

5）函数体相对于 def 关键字必须保持一定的空格缩进。

函数递归通常用来把一个大型的复杂问题层层转化为一个与原来问题本质相同但规模很小、很容易解决或描述的问题，只需要很少的代码就可以描述解决问题过程中需要的大量重复计算。在编写递归函数时，应注意：

- 每次递归应保持问题性质不变；
- 每次递归应使用更小或更简单的输入；
- 必须有一个能够直接处理而不需要再次进行递归的特殊情况来保证递归过程可以结束；
- 函数递归深度不能太大，否则会引起内存崩溃。

位置参数是比较常用的形式，调用函数时实参和形参的顺序必须严格一致，并且实参和形参的数量必须相同。

```
>>> def demo(a, b, c):                    #所有形参都是位置参数
    print(a, b, c)

>>> demo(3, 4, 5)
3 4 5
>>> demo(3, 5, 4)
3 5 4
>>> demo(1, 2, 3, 4)                      #实参与形参数量必须相同
TypeError: demo() takes 3 positional arguments but 4 were given
```

在定义函数时，Python 支持默认值参数，在定义函数时可以为形参设置默认值。在调用带有默认值参数的函数时，可以不用为设置了默认值的形参进行传值，此时函数将会直接使用函数定义时设置的默认值，当然也可以通过显式赋值来替换其默认值。

需要注意的是，在定义带有默认值参数的函数时，任何一个默认值参数右边都不能再出现没有默认值的普通位置参数，否则会提示语法错误。带有默认值参数的函数定义语法如下：

```
def 函数名(……，形参名=默认值):
    函数体
```

第 二 节 课

关键参数主要指调用函数时的参数传递方式，通过关键参数可以按参数名字传递值，明确指定哪个值传递给哪个参数，实参顺序可以和形参顺序不一致，但不影响参数值的传递结果，避免了用户需要牢记参数位置和顺序的麻烦，使得函数的调用和参数传递更加灵活方便。

```
>>> def demo(a, b, c=5):
    print(a, b, c)

>>> demo(3, 7)
3 7 5
>>> demo(a=7, b=3, c=6)
7 3 6
>>> demo(c=8, a=9, b=0)
9 0 8
```

可变长度参数在定义函数时主要有两种形式：*parameter 和**parameter，前者用来接收任意多个位置实参并将其放在一个元组中，后者接收多个关键参数并将其放入字典中。

下面的代码演示了第一种形式可变长度参数的用法，无论调用该函数时传递了多少实参，一律将其放入元组中：

```
>>> def demo(*p):
    print(p)

>>> demo(1, 2, 3)
(1, 2, 3)
>>> demo(1, 2, 3, 4, 5, 6, 7)
(1, 2, 3, 4, 5, 6, 7)
```

下面的代码演示了第二种形式可变长度参数的用法，在调用该函数时自动将接收的多个关键参数转换为字典中的元素：

```
>>> def demo(**p):
    for item in p.items():
        print(item)

>>> demo(x=1, y=2, z=3)
('y', 2)
('x', 1)
('z', 3)
```

与可变长度的参数相对应，这里的序列解包是指实参，同样也有*和**两种形式。调用含有多个位置参数的函数时，可以使用 Python 列表、元组、集合、字典以及其他可迭代对象作为实参，并在实参名称前加一个星号，Python 解释器将自动进行解包，然后把序列中的值分别传递给多个形参变量。

```
>>> def demo(a, b, c):                    #可以接收多个位置参数的函数
    print(a+b+c)

>>> seq = [1, 2, 3]
>>> demo(*seq)                            #对列表进行解包
6
>>> tup = (1, 2, 3)
>>> demo(*tup)                            #对元组进行解包
6
>>> dic = {1:'a', 2:'b', 3:'c'}
>>> demo(*dic)                            #对字典的键进行解包
6
>>> demo(*dic.values())                   #对字典的值进行解包
abc
>>> Set = {1, 2, 3}
>>> demo(*Set)                            #对集合进行解包
6
```

如果实参是个字典，可以使用两个星号**对其进行解包，会把字典转换成类似于关键参数的形式进行参数传递。对于这种形式的序列解包，要求实参字典中的所有键都必须是函数的形参名称，或者与函数中两个星号的可变长度参数相对应。

```
>>> p = {'a':1, 'b':2, 'c':3}             #要解包的字典
>>> def f(a, b, c=5):                     #带有位置参数和默认值参数的函数
    print(a, b, c)

>>> f(**p)
1 2 3
>>> def f(a=3, b=4, c=5):                 #带有多个默认值参数的函数
    print(a, b, c)

>>> f(**p)                                #对字典元素进行解包
1 2 3
>>> def demo(**p):                        #接收字典形式可变长度参数的函数
    for item in p.items():
        print(item)

>>> p = {'x':1, 'y':2, 'z':3}
>>> demo(**p)                             #对字典元素进行解包
('y', 2)
('z', 3)
('x', 1)
```

变量起作用的代码范围称为变量的作用域，不同作用域内同名变量之间互不影响。如果想要在函数内部修改一个定义在函数外的变量值，必须要使用 global 明确声明，否则会自动创建新的局部变量。

在函数内如果只引用某个变量的值而没有为其赋新值，该变量为（隐式的）全局变量。如果在函数内有为变量赋值的操作，该变量就被认为是（隐式的）局部变量，除非在函数内赋值操作之前显式地用关键字 global 进行了声明。

Python程序设计教案

<table>
<tr><td>本次授课内容</td><td colspan="2">5.4 lambdabds
5.5 生成器函数
5.6综合案例解析</td></tr>
<tr><td>本次课的
教学目的</td><td colspan="2">掌握lambda表达式的定义与用法
理解生成器函数工作原理</td></tr>
<tr><td>本次课教学
重点与难点</td><td colspan="2">lambda表达式语法、功能与应用场景
生成器函数工作原理、生成器对象的惰性求值特点</td></tr>
<tr><td>教学方法
教学手段</td><td colspan="2">PPT、边讲边练</td></tr>
<tr><td rowspan="3">课堂教学
时间分配</td><td>教学内容</td><td>时间分配（分）</td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td>课堂教学设计</td><td colspan="2">介绍lambda表达式语法、功能和应用场景，然后介绍生成器函数工作原理，最后通过几个例题演示本章语法和应用。</td></tr>
<tr><td>实　验</td><td colspan="2">lambda函数定义与应用场景，生成器函数定义与使用</td></tr>
<tr><td>思考题及作业题</td><td colspan="2"></td></tr>
<tr><td>备　注</td><td colspan="2"></td></tr>
<tr><td>教学后记</td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2">第　一　节　课</td></tr>
</table>

课堂重点内容详解

```
>>> f = lambda x, y, z: x+y+z          #也可以给 lambda 表达式起个名字
>>> print(f(1, 2, 3))                   #把 lambda 表达式当作函数使用
6
>>> g = lambda x, y=2, z=3: x+y+z      #支持默认值参数
>>> print(g(1))
6
>>> print(g(2, z=4, y=5))               #调用时使用关键参数
11
>>> L=[1, 2, 3, 4, 5]
>>> list(map(lambda x: x+10, L))       #lambda 表达式作为函数参数
[11, 12, 13, 14, 15]
>>> data = list(range(20))
>>> import random
>>> random.shuffle(data)                #随机打乱顺序
>>> data
[4, 3, 11, 13, 12, 15, 9, 2, 10, 6, 19, 18, 14, 8, 0, 7, 5, 17, 1, 16]
>>> data.sort(key=lambda x: len(str(x)))
                                        #使用 lambda 表达式指定排序规则
                                        #按所有元素转换为字符串后的长度排序
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> data.sort(key=lambda x: len(str(x)), reverse=True)
                                        #reverse=True 表示降序排序
>>> data
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

例 5-3 编写并使用能够生成斐波那契数列的生成器函数。

```
>>> def f():
    a, b = 1, 1              #序列解包，同时为多个元素赋值
    while True:
        yield a              #暂停执行，需要时再产生一个新元素
        a, b = b, a+b        #序列解包，继续生成新元素

>>> a = f()                  #创建生成器对象
>>> for i in range(10):     #斐波那契数列中前 10 个元素
    print(a.__next__(), end=' ')

1 1 2 3 5 8 13 21 34 55
>>> for i in f():           #斐波那契数列中第一个大于 100 的元素
    if i > 100:
        print(i, end=' ')
        break
144
```

第 二 节 课

例 5-4 编写函数，接收字符串参数，返回一个元组，该元组中第一个元素为大写字母个数，第二个元素为小写字母个数。

```
def demo(s):
    result = [0, 0]
    for ch in s:
```

```
        if ch.islower():              #当前字符为小写字母
            result[1] += 1            #小写字母个数加1
        elif ch.isupper():            #当前字符为大写字母
            result[0] += 1            #大写字母个数加1
    return tuple(result)

print(demo('Beautiful is better than ugly.'))
```

例 5-5 编写函数，接收一个整数 t 为参数，打印杨辉三角前 t 行。

```
def yanghui(t):
    print([1])                                  #输出第一行
    line = [1, 1]
    print(line)                                 #输出第二行
    for i in range(2, t):
        r = []                                  #存储当前行除两端之外的数字
        for j in range(0, len(line)-1):
            r.append(line[j]+line[j+1])    #第 i 行除两端之外其他的数字
        line = [1]+r+[1]                        #第 i 行的全部数字
        print(line)                             #输出第 i 行
```

例 5-6 编写函数，接收一个正偶数为参数，输出两个素数，并且这两个素数之和等于原来的正偶数。如果存在多组符合条件的素数，则全部输出。

```
def demo(n):
    def IsPrime(p):                 #该函数用来判断 p 是否为素数
        if p == 2:
            return True
        if p%2 == 0:
            return False
        for i in range(3, int(p**0.5)+1, 2):
            if p%i==0:
                return False
        return True

    if isinstance(n, int) and n>0 and n%2==0:
        for i in range(2, n//2+1):
            if IsPrime(i) and IsPrime(n-i):
                print(i, '+', n-i, '=', n)
```

例 5-7 编写函数，计算字符串匹配的准确率。

```
def Rate(origin, userInput):
    if not (isinstance(origin, str) and isinstance(userInput, str)):
        print('The two parameters must be strings.')
        return
    #统计对应位置上打对的字符数量
    right = sum((1 for o, u in zip(origin, userInput) if o==u))
    return round(right/len(origin), 2)

s1 = 'Readability counts.'
s2 = 'readability count.'
```

```
print(Rate(s1, s2))
```

例 5-8 编写函数模拟猜数游戏。系统随机产生一个数，玩家最多可以猜 3 次，系统会根据玩家的猜测进行提示，玩家则可以根据系统的提示对下一次的猜测进行适当调整。

```
from random import randint

def guess(maxValue=10, maxTimes=3):
    #随机生成一个整数
    value = randint(1,maxValue)
    for i in range(maxTimes):
        #第一次猜和后面几次的提示信息不一样
        prompt = 'Start to GUESS:' if i==0 else 'Guess again:'
        #使用异常处理结构，防止输入不是数字的情况
        try:
            x = int(input(prompt))
        except:
            #如果输入的不是数字，就输出下面的这句话
            #然后直接进入下一次循环，不会执行下面 else 子句的代码
            print('Must input an integer between 1 and ', maxValue)
        else:
            #如果上面 try 中的代码没有出现异常，继续执行这个 else 中的代码
            #猜对了，退出游戏
            if x == value:
                print('Congratulations!')
                break
            elif x > value:
                print('Too big')
            else:
                print('Too little')
    else:
        #次数用完还没猜对，游戏结束，提示正确答案
        print('Game over. FAIL.')
        print('The value is ', value)
```

例 5-9 编写函数模拟报数游戏。有 n 个人围成一圈，顺序编号，从第一个人开始从 1 到 k（假设 k=3）报数，报到 k 的人退出圈子，然后圈子缩小，从下一个人继续游戏，问最后留下的是原来的第几号。

```
from itertools import cycle

def demo(lst, k):
    #切片，以免影响原来的数据
    t_lst = lst[:]
    #游戏一直进行到只剩下最后一个人
    while len(t_lst)>1:
        #创建 cycle 对象
        c = cycle(t_lst)
        #从 1 到 k 报数
```

```
        for i in range(k):
            t = next(c)
        #一个人出局，圈子缩小
        index = t_lst.index(t)
        t_lst = t_lst[index+1:] + t_lst[:index]
    #游戏结束
    return t_lst[0]

lst = list(range(1,11))
print(demo(lst, 3))
```

例 5-10 汉诺塔问题基于递归算法的实现。

```
def hannoi(num, src, dst, temp=None):
    '''参数含义：把 src 上的 num 个盘子借助于 temp 移动到 dst'''
    #声明用来记录移动次数的变量为全局变量
    global times
    #确认参数类型和范围
    assert type(num) == int, 'num must be integer'
    assert num > 0, 'num must > 0'
    #只剩最后或只有一个盘子需要移动，这也是函数递归调用的结束条件
    if num == 1:
        print('The {0} Times move:{1}==>{2}'.format(times, src, dst))
        times += 1
    else:
        #递归调用函数自身，
        #先把除最后一个盘子之外的所有盘子移动到临时柱子上
        hannoi(num-1, src, temp, dst)
        #把最后一个盘子直接移动到目标柱子上
        hannoi(1, src, dst)
        #把除最后一个盘子之外的其他盘子从临时柱子上移动到目标柱子上
        hannoi(num-1, temp, dst, src)
#用来记录移动次数的变量
times = 1
#A 表示最初放置盘子的柱子，C 是目标柱子，B 是临时柱子
hannoi(3, 'A', 'C', 'B')
```

例 5-11 编写函数计算任意位数的黑洞数。

```
def main(n):
    '''参数 n 表示数字的位数，例如 n=3 时返回 495，n=4 时返回 6174'''
    #待测试数范围的起点和结束值
    start = 10**(n-1)
    end = 10**n
    #依次测试每个数
    for i in range(start, end):
        #由这几个数字组成的最大数和最小数
        big = ''.join(sorted(str(i),reverse=True))
        little = ''.join(reversed(big))
        big, little = map(int,(big, little))
        if big-little == i:
```

```
            print(i)
n = 4
main(n)
```

例 5-12 编写函数，实现冒泡排序算法。

```
from random import randint

def bubbleSort(lst, reverse=False):
    length = len(lst)
    for i in range(0, length):
        flag = False
        for j in range(0, length-i-1):
            #比较相邻两个元素大小，并根据需要进行交换
            #默认升序排序
            exp = 'lst[j] > lst[j+1]'
            #如果 reverse=True 则降序排序
            if reverse:
                exp = 'lst[j] < lst[j+1]'
            if eval(exp):
                lst[j], lst[j+1] = lst[j+1], lst[j]
                #flag=True 表示本次扫描发生过元素交换
                flag = True
        #如果一次扫描结束后，没有发生过元素交换，说明已经按序排列
        if not flag:
            break

lst = [randint(1, 100) for i in range(20)]
print('排序前:\n', lst)
bubbleSort(lst, True)
print('排序后:\n', lst)
```

例 5-13 编写函数，实现选择法排序。

```
from random import randint

def selectSort(lst, reverse=False):
    length = len(lst)
    for i in range(0, length):
        #假设剩余元素中第一个最小或最大
        m = i
        #扫描剩余元素
        for j in range(i+1, length):
            #如果有更小或更大的，就记录下它的位置
            exp = 'lst[j] < lst[m]'
            if reverse:
                exp = 'lst[j] > lst[m]'
            if eval(exp):
                m = j
        #如果发现更小或更大的，就交换值
        if m!=i:
```

```
            lst[i], lst[m] = lst[m], lst[i]

lst = [randint(1, 100) for i in range(20)]
print('排序前:\n', lst)
selectSort(lst, True)
print('排序后:\n', lst)
```

例 5-14 编写函数，实现二分法查找。

```
from random import randint

def binarySearch(lst, value):
    start = 0
    end = len(lst)
    while start < end:
        #计算中间位置
        middle = (start + end) // 2
        #查找成功，返回元素对应的位置
        if value == lst[middle]:
            return middle
        #在后面一半元素中继续查找
        elif value > lst[middle]:
            start = middle + 1
        #在前面一半元素中继续查找
        elif value < lst[middle]:
            end = middle - 1
    #查找不成功，返回 False
    return False

lst = [randint(1,50) for i in range(20)]
lst.sort()
print(lst)
result = binarySearch(lst, 30)
if result != False:
    print('Success, its position is:',result)
else:
    print('Fail. Not exist.')
```

例 5-15 编写函数，查找给定序列的最长递增子序列。

```
from itertools import combinations
from random import sample

def subAscendingList(lst):
    '''返回最长递增子序列'''
    for length in range(len(lst), 0, -1):
        #按长度递减的顺序进行查找和判断
        for sub in combinations(lst, length):
            #判断当前选择的子序列是否为递增顺序
            if list(sub) == sorted(sub):
                #找到第一个就返回
                return sub
```

```
def getList(start=0, end=1000, number=20):
    '''生成随机序列'''
    if number > end-start:
        return None
    return sample(range(start, end), number)

def main():
    #生成一个包含 10 个随机数的列表进行测试
    lst = getList(number=10)
    if lst:
        print(lst)
        print(subAscendingList(lst))

main()
```

例 5-16 编写函数，寻找给定序列中相差最小的两个数字。

```
import random

def getTwoClosestElements(seq):
    #先进行排序，使得相邻元素最接近
    #相差最小的元素必然相邻
    seq = sorted(seq)
    #无穷大
    dif = float('inf')
    for i,v in enumerate(seq[:-1]):
        d = abs(v - seq[i+1])
        if d < dif:
            first, second, dif = v, seq[i+1], d
    #返回相差最小的两个元素
    return (first, second)

seq = [random.randint(1, 10000) for i in range(20)]
print(seq)
print(sorted(seq))
print(getTwoClosestElements(seq))
```

例 5-17 利用蒙特.卡罗方法计算圆周率近似值。

```
from random import random

def estimatePI(times):
    hits = 0
    for i in range(times):
        x = random()*2 - 1          #random()生成介于 0 和 1 之间的小数
        y = random()*2 - 1          #该数字乘以 2 再减 1，则介于-1 和 1 之间
        if x*x + y*y <= 1:          #落在圆内或圆周上
            hits += 1
    return 4.0 * hits/times

print(estimatePI(10000))
print(estimatePI(1000000))
```

```
print(estimatePI(100000000))
print(estimatePI(1000000000))
```

例 5-18 模拟蒙蒂霍尔悖论游戏。

```
from random import randrange

def init():
    '''返回一个字典，键为 3 个门号，值为门后面的物品'''
    result = {i: 'goat' for i in range(3)}
    r = randrange(3)
    #在某个随机的门后面放一辆汽车，其他两个门后面仍然是山羊
    result[r] = 'car'
    return result

def startGame():
    #获取本次游戏中每个门的情况
    doors = init()
    #获取玩家选择的门号
    while True:
        try:
            firstDoorNum = int(input('Choose a door to open:'))
            assert 0<= firstDoorNum <=2
            break
        except:
            print('Door number must be between {} and {}'.format(0, 2))

    #主持人查看另外两个门后的物品情况
    #字典的 keys()方法返回结果可以当作集合使用，支持使用减法计算差集
    for door in doors.keys()-{firstDoorNum}:
        #打开其中一个后面为山羊的门
        if doors[door] == 'goat':
            print('"goat" behind the door', door)
            #获取第三个门号，让玩家纠结
            thirdDoor = (doors.keys()-{door, firstDoorNum}).pop()
            change = input('Switch to {}?(y/n)'.format(thirdDoor))
            finalDoorNum = thirdDoor if change=='y' else firstDoorNum
            if doors[finalDoorNum] == 'goat':
                return 'I Win!'
            else:
                return 'You Win.'
while True:
    print('='*30)
    print(startGame())
    r = input('Do you want to try once more?(y/n)')
    if r == 'n':
        break
```