函数和递归

礼 欣

北京理工大学

6.6 函数和程序结构

- 函数可以简化程序，函数可以使程序模块化
- 用函数将较长的程序分割成短小的程序段，可以方便理解
- 程序例子：

```
print("This program plots the growth of a 10-year investment.")
# 输入本金和利率
principal = eval(input("Enter the initial principal: "))
apr = eval(input("Enter the annualized interest rate: "))
# 建立一个图表，绘制每一年银行帐户的增长数据
for year in range(1, 11):
    principal = principal * (1 + apr)
    print("%2d" %year, end='')
    # 计算星号的数量
    total = int(principal*4/1000)
    print ("*" * total)
print( " 0.0K      2.5K      5.0K      7.5K      10.0K")
```

6.6 函数和程序结构

- 将部分功能从程序中移出作为独立函数
  - 星号绘制函数

```
def createTable(principal, apr):
    #为每一年绘制星号的增长图
    for year in range(1, 11):
        principal = principal * (1 + apr)
        print("%2d"%year, end = '')
        total = caculateNum(principal)
        print("*" * total)
    print(" 0.0K       2.5K       5.0K       7.5K       10.0K")
```

6.6 函数和程序结构

- 星号数量计算函数

```
def caculateNum(principal):
    # 计算星号数量
    total= int(principal*4/1000.0)
    return total
```

- 整体控制函数

```
def main():
    print("This program plots the growth of a 10-year investment.")
    # 输入本金和利率
    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annualized interest rate: "))
    # 建立图表
    createTable(principal,apr)
main()
```

6.6 函数和程序结构

- 完整程序可以写为。

```
def createTable(principal, apr):
    #为每一年绘制星号的增长图
    for year in range(1, 11):
        principal = principal * (1 + apr)
        print("%2d"%year, end = '')
        total = caculateNum(principal)
        print("*" * total)
    print(" 0.0K      2.5K      5.0K      7.5K      10.0K")
def caculateNum(principal):
    # 计算星号数量
    total= int(principal*4/1000.0)
    return total
def main():
    print("This program plots the growth of a 10-year investment.")
    # 输入本金和利率
    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annualized interest rate: "))
    # 建立图表
    createTable(principal,apr)
main()
```

6.6 函数和程序结构

- 整个程序可读性很强。使用函数的思想编写程序，可以大大增加程序的模块化程度
- 程序运行结果为：

```
This program plots the growth of a 10-year investment.
Enter the initial principal: 1000
Enter the annualized interest rate: 0.2
 1****
 2*****
 3******
 4********
 5*********
 6**********
 7***********
 8************
 9*************
10**************
 0.0K      2.5K      5.0K      7.5K      10.0K
```

6.7.1 递归的定义

- 递归：函数定义中使用函数自身的方法
- 经典例子：阶乘

$$n! = n(n-1)(n-2)...(1)$$

- 举例：5！=5(4)(3)(2)(1) =5 * 4！
- 推广： n!=n(n-1)!

python 语言程序设计

6.7.1 递归的定义

- 阶乘的递归定义：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & otherwise \end{cases}$$

- 0的阶乘：定义为1
- 其他数字：定义为这个数字乘以比这个数字小1的数的阶乘

6.7.1 递归的定义

- 递归不是循环
- 最后计算基例：0!。0!是已知值
- 递归定义特征 ：
  - 有一个或多个基例是不需要再次递归的；
  - 所有的递归链都要以一个基例结尾。

6.7.2递归函数

- 通过一个累计器循环计算阶乘
- 阶乘的递归定义函数 ：

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

python 语言程序设计

6.7.2递归函数

- 运行递归函数fact()计算阶乘：

```
>>> fact(4)
24
>>> fact(10)
3628800
```

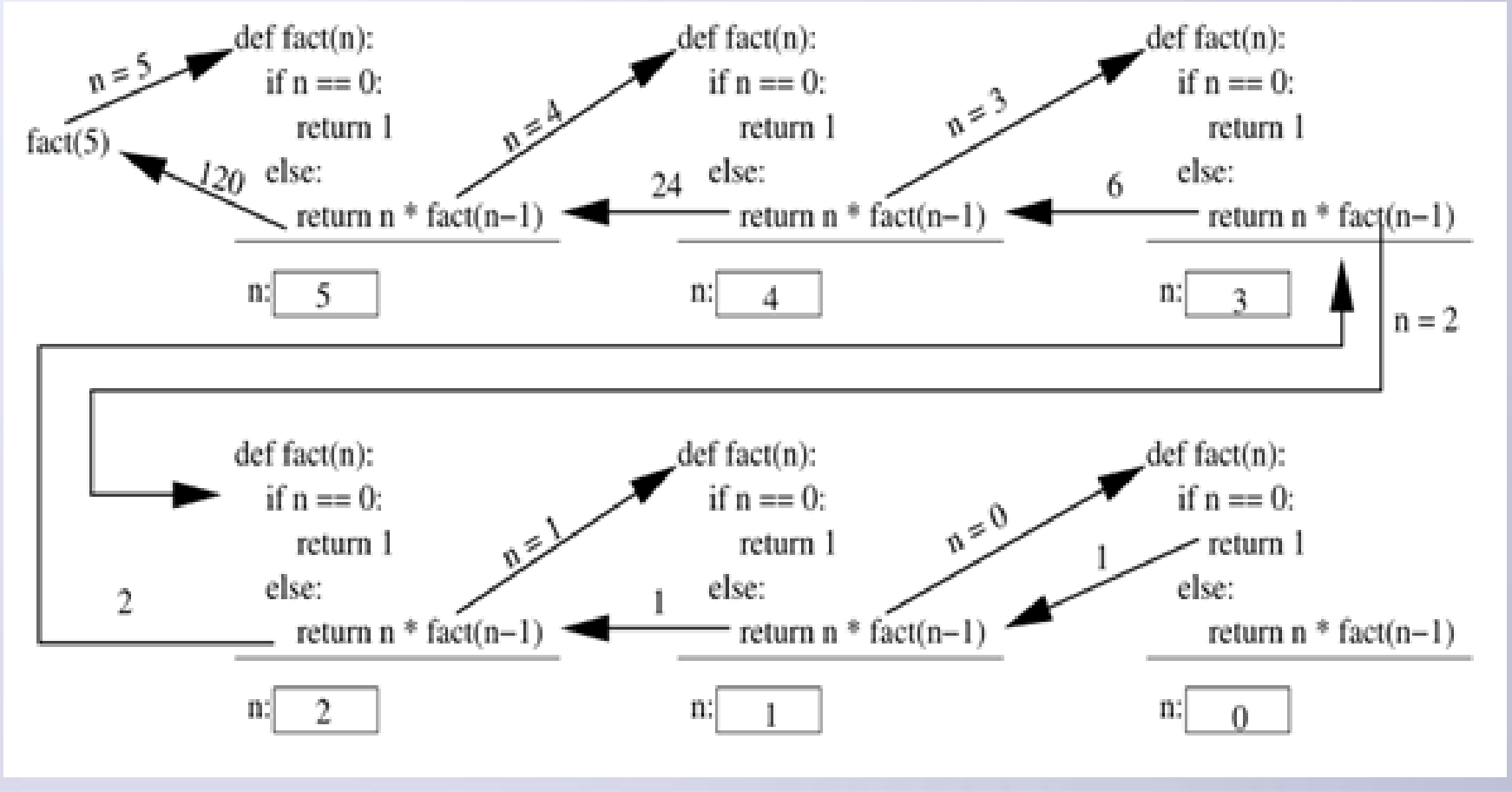
6.7.2递归函数

- 递归每次调用都会引起新函数的开始
- 递归有本地值的副本，包括该值的参数
- 阶乘递归函数中：每次函数调用中的相关n值在中途的递归链暂时存储，并在函数返回时使用。

python 语言程序设计

6.7.2递归函数

- 5!的递归调用过程图



6.7.3示例程序：字符串反转

- Python列表有反转的内置方法
  - 方法1：字符串转换为字符列表，反转列表，列表转换回字符串
  - 方法2：递归

6.7.3示例程序：字符串反转

- 此问题的IPO模式：
  - 输入：字符串
  - 处理：用递归的方法反转字符串
  - 输出：反转后的字符串
- 基本思想：把字符串看做递归对象

6.7.3示例程序：字符串反转

- 将字符串分割成首字符和剩余子字符串
- 反转了剩余部分后把首字符放到末尾，整个字符串反转就完成了
- 字符串反转算法(常犯错误版)：

```
def reverse(s):
    return reverse(s[1:]) + s[0]
```

6.7.3示例程序：字符串反转

- 此算法运行结果出错 ：

```
>>> reverse("Hello")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    reverse("Hello")
  File "C:\Python34\recursion_string.py", line 2, in reverse
    return reverse(s[1:]) + s[0]
  File "C:\Python34\recursion_string.py", line 2, in reverse
    return reverse(s[1:]) + s[0]
```

.....

```
  File "C:\Python34\recursion_string.py", line 2, in reverse
    return reverse(s[1:]) + s[0]
RuntimeError: maximum recursion depth exceeded
```

6.7.3示例程序：字符串反转

- 构造递归函数，需要基例
- 基例不进行递归，否则递归就会无限循环执行
- Python在900余次调用之后，到达默认的“递归深度的最大值”，终止调用
- 此递归调用以字符串形式执行，应设置基例为空串

6.7.3示例程序：字符串反转

- 正确的字符串反转代码（新版）：

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

python 语言程序设计

6.7.3示例程序：字符串反转

- 运行结果：

```
>>> reverse("Hello")
'olleH'
```