
python
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for 1 in range(4):
turtle.circle(40, 80)
turtle.circle(-40, 80)
turtle.fd(40)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)

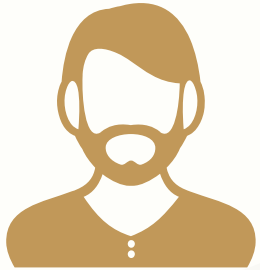
Python语言程序设计

代码复用与函数递归

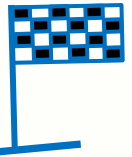
嵩 天
北京理工大学

单元开篇

代码复用与函数递归



- 代码复用与模块化设计
- 函数递归的理解
- 函数递归的调用过程
- 函数递归实例解析



python
语言程序设计

代码复用与模块化设计

代码复用

把代码当成资源进行抽象

- 代码资源化：程序代码是一种用来表达计算的"资源"
- 代码抽象化：使用函数等方法对代码赋予更高级别的定义
- 代码复用：同一份代码在需要时可以被重复使用

代码复用

函数 和 对象 是代码复用的两种主要形式

函数：将代码命名

在代码层面建立了初步抽象

对象：属性和方法

`<a>.<b>` 和 `<a>.<b>()`

在函数之上再次组织进行抽象

抽象级别

模块化设计

分而治之

- 通过函数或对象封装将程序划分为模块及模块间的表达
- 具体包括：主程序、子程序和子程序间关系
- 分而治之：一种分而治之、分层抽象、体系化的设计思想

模块化设计

紧耦合 松耦合

- 紧耦合：两个部分之间交流很多，无法独立存在
- 松耦合：两个部分之间交流较少，可以独立存在
- 模块内部紧耦合、模块之间松耦合

函数递归的理解

递归的定义

函数定义中调用函数自身的方式

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

递归的定义

两个关键特征

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

- **链条：计算过程存在递归链条**
- **基例：存在一个或多个不需要再次递归的基例**

递归的定义

类似数学归纳法

- 数学归纳法
  - 证明当n取第一个值n_0时命题成立
  - 假设当n_k时命题成立，证明当n=n_{k+1}时命题也成立
- 递归是数学归纳法思维的编程体现

函数递归的调用过程

递归的实现

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

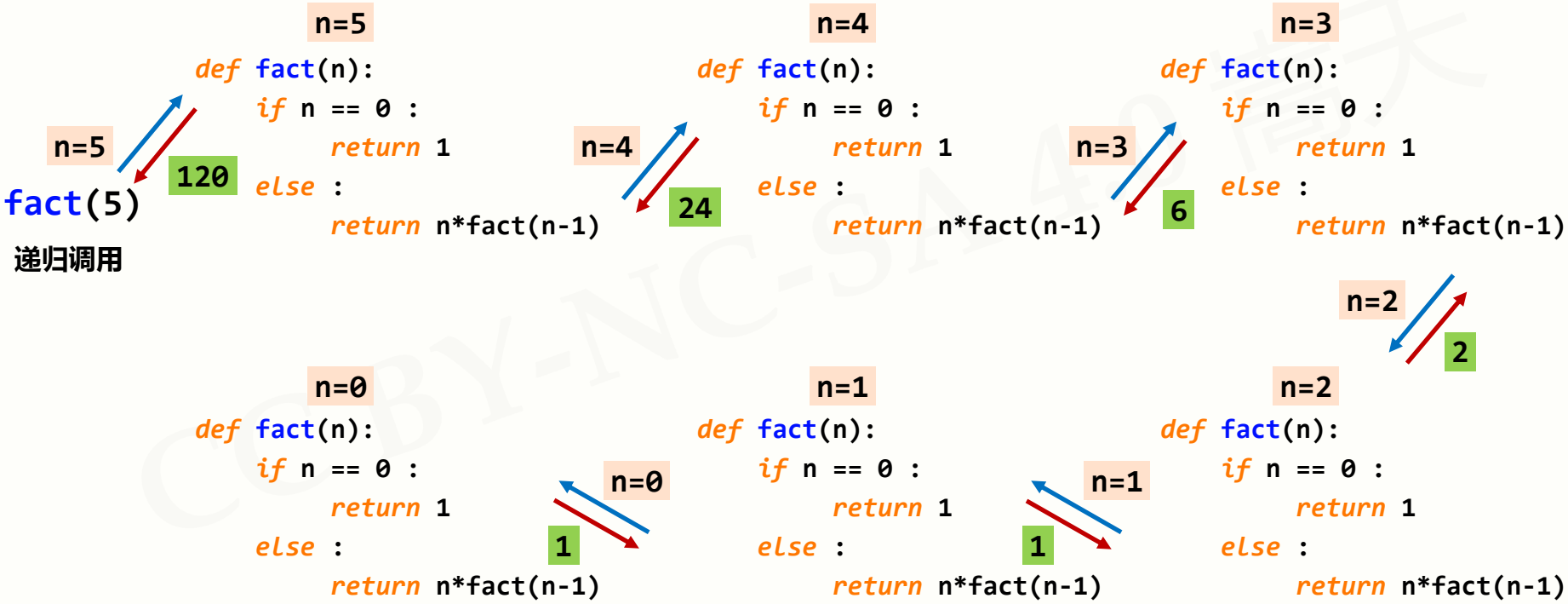
```
def fact(n):
    if n == 0 :
        return 1
    else :
        return n*fact(n-1)
```

递归的实现

函数 + 分支语句

- 递归本身是一个函数，需要函数定义方式描述
- 函数内部，采用分支语句对输入参数进行判断
- 基例和链条，分别编写对应代码

递归的调用过程



函数递归实例解析

字符串反转

将字符串s反转后输出

```
>>> s[::-1]
```

- **函数 + 分支结构**
- **递归链条**
- **递归基例**

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

斐波那契数列

一个经典数列

$$F(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ F(n-1) + F(n-2) & otherwise \end{cases}$$

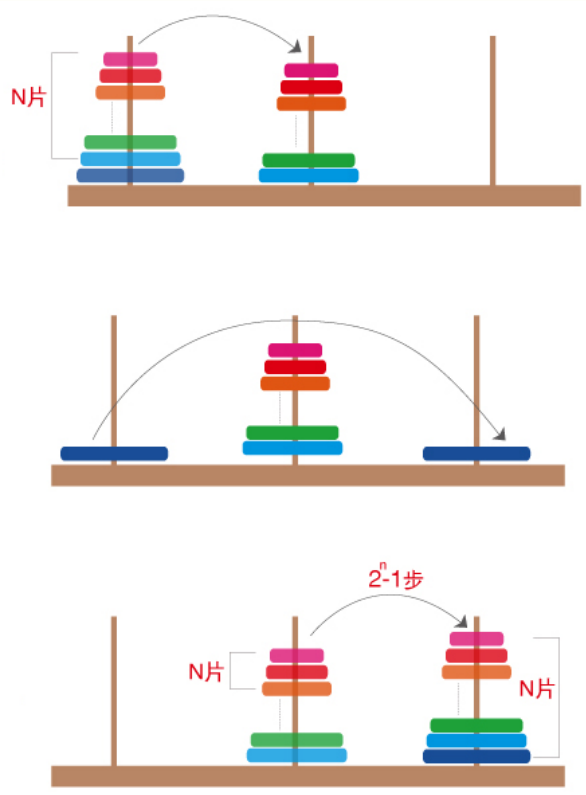
斐波那契数列

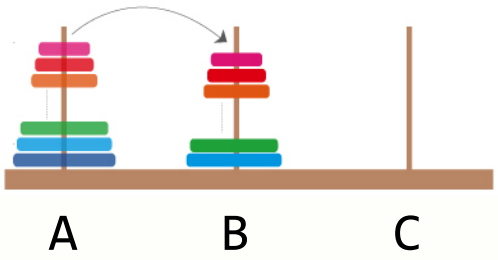
$$F(n) = F(n-1) + F(n-2)$$

- 函数 + 分支结构
- 递归链条
- 递归基例

```
def f(n):
    if n == 1 or n == 2 :
        return 1
    else :
        return f(n-1) + f(n-2)
```

汉诺塔

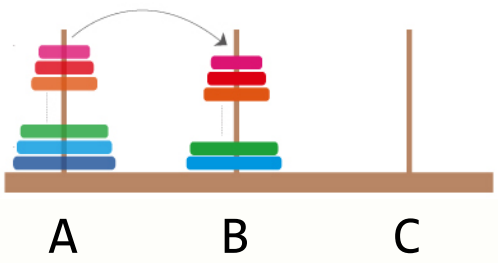




汉诺塔

- 函数 + 分支结构
- 递归链条
- 递归基例

```
count = 0
def hanoi(n, src, dst, mid):
    global count
    if n == 1 :
        print("{}:{}->{}".format(1,src,dst))
        count += 1
    else :
        hanoi(n-1, src, mid, dst)
        print("{}:{}->{}".format(n,src,dst))
        count += 1
        hanoi(n-1, mid, dst, src)
```



汉诺塔

```
count = 0
def hanoi(n, src, dst, mid):
    … (略)
hanoi(3, "A", "C", "B")
print(count)
```

```
>>>
1:A->C
2:A->B
1:C->B
3:A->C
1:B->A
2:B->C
1:A->C
7
```

单元小结

代码复用与函数递归

- **模块化设计：松耦合、紧耦合**
- **函数递归的2个特征：基例和链条**
- **函数递归的实现：函数 + 分支结构**

python
语言程序设计