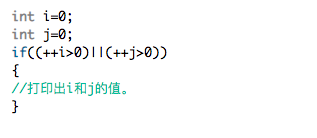
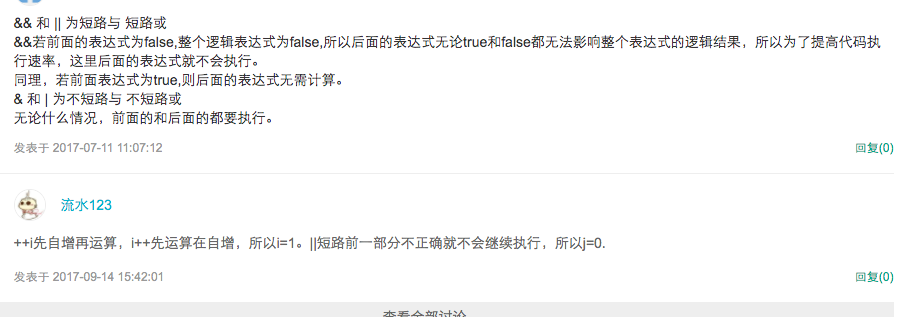
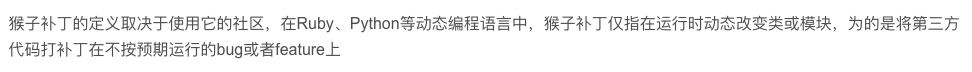
**陈中原：**

****

****

**苏建飞：**

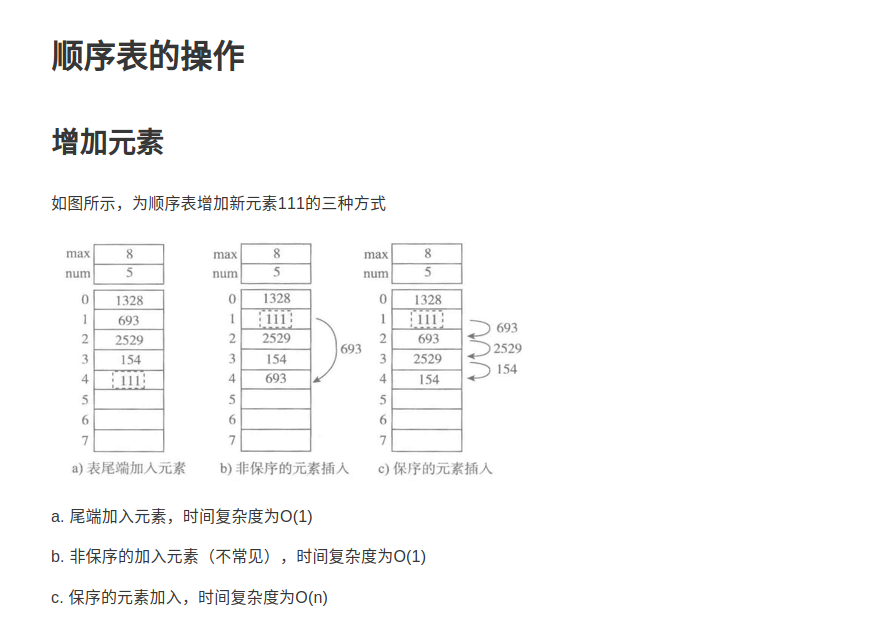
解释一下 什么是Python 的猴子补丁

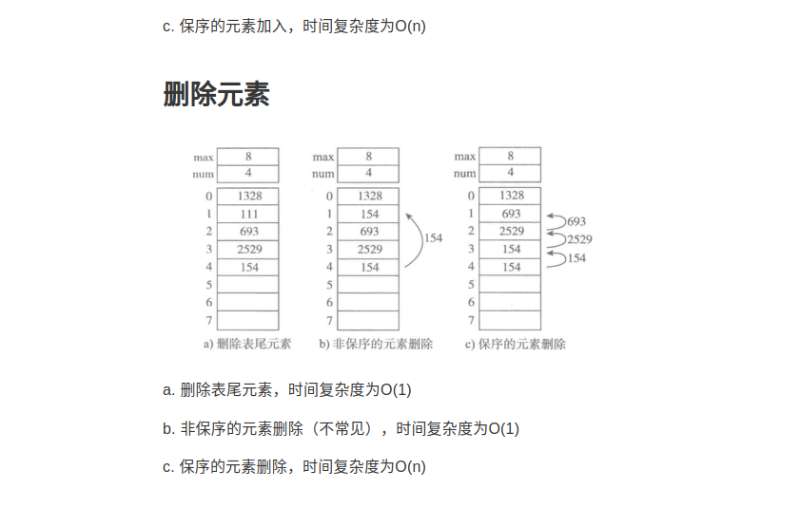
****

****

**丁昌茂：**

****

****

****

一个顺序表的完整信息包括两部分，一部分是表中的元素集合，另一部分是为实现正确操作而需记录的信息，即有关表的整体情况的信息，这部分信息主要包括元素存储区的容量和当前表中已有的元素个数两项,代表就是list和元祖

**王福成：**

package day0323;

public class IntegerBreak {

/\*\*

\* leetcode 343 Integer Break

\* 给定一个正整数n，可以将其分割成多个数字的和，若要让这些数字的乘机最大，求分割的方法（至少要分成两个数）。

\* 算法返回这个最大的乘积。

\*

\* 问题分析：

\* 最优子结构+重叠子问题

\* 1+... 2+... 3+... n-1+...

\*/

// 递归方式：

public Integer break1(int n){

assert n > 1;

Integer max = -1;

for(int i=1;i<n;i++){

// 比较 当前最大乘积、不继续分割的最大乘积、继续分割子结构的最大乘积

max = max3(max,i \* (n - i),i \* break1(n - i));

}

return max;

}

private int[] breaks;

public IntegerBreak(int initN){

breaks = new int[initN];

}

/\*\*

\* 递归 + 记忆化搜索

\* 在递归的基础上增加一个列表，保存每个数的最大break后的乘积，避免重复计算

\* @param n

\* @return

\*/

public Integer break2(int n){

assert n > 1;

Integer max = -1;

for(int i=1;i<n;i++){

// 比较 当前最大乘积、不继续分割的最大乘积、继续分割子结构的最大乘积

if(this.breaks[n-i-1] == 0)

max = max3(max,i \* (n - i),i \* break1(n - i));

else

max = max3(max,i \* (n - i),i \* this.breaks[n-i-1]);

this.breaks[n-1] = max;

}

return max;

}

// 动态规划方式自底向上的求解

public Integer break3(int n){

return null;

}

public static Integer max3(int a, int b, int c){

return Math.max(a, Math.max(b, c));

}

public static void main(String[] args) {

IntegerBreak integerBreak = new IntegerBreak(10);

System.out.println(integerBreak.break1(10));

System.out.println(integerBreak.break2(10));

}

}

package day0323;

public class IntegerBreak {

/\*\*

\* leetcode 343 Integer Break

\* 给定一个正整数n，可以将其分割成多个数字的和，若要让这些数字的乘机最大，求分割的方法（至少要分成两个数）。

\* 算法返回这个最大的乘积。

\*

\* 问题分析：

\* 最优子结构+重叠子问题

\* 1+... 2+... 3+... n-1+...

\*/

// 递归方式：

public Integer break1(int n){

assert n > 0;

if(n == 1)

return 1;

Integer max = -1;

for(int i=1;i<n;i++){

// 比较 当前最大乘积、不继续分割的最大乘积、继续分割子结构的最大乘积

max = max3(max,i \* (n - i),i \* break1(n - i));

}

return max;

}

private int[] break2Arr;

private int[] break3Arr;

public IntegerBreak(int initN){

break2Arr = new int[initN];

break3Arr = new int[initN];

}

/\*\*

\* 递归 + 记忆化搜索

\* 在递归的基础上增加一个列表，保存每个数的最大break后的乘积，避免重复计算

\* @param n

\* @return

\*/

public Integer break2(int n){

assert n > 0;

if(n == 1)

return 1;

Integer max = 0;

for(int i=1;i<n;i++){

// 比较 当前最大乘积、不继续分割的最大乘积、继续分割子结构的最大乘积，记录n的最大break值

if(this.break2Arr[n-i-1] == 0)

max = max3(max,i \* (n - i),i \* break1(n - i));

else

max = max3(max,i \* (n - i),i \* this.break2Arr[n-i-1]);

this.break2Arr[n-1] = max;

}

return max;

}

// 动态规划方式自底向上的求解

public Integer break3(int n){

assert n > 0;

if(n == 1)

return 1;

// 从1开始算起

break3Arr[0] = 1;

for(int i=2;i<=n;i++){

int max = 0;

for(int j = 1;j<i;j++){

// 比较 当前最大乘积、不继续分割的最大乘积、继续分割子结构的最大乘积，记录n的最大break值

max = max3(max, j\*(i-j), j\*break3Arr[i-j-1]);

}

break3Arr[i-1] = max;

}

return break3Arr[n - 1];

}

public static Integer max3(int a, int b, int c){

return Math.max(a, Math.max(b, c));

}

public static void main(String[] args) {

IntegerBreak integerBreak = new IntegerBreak(10);

System.out.println(integerBreak.break1(10));

System.out.println(integerBreak.break2(10));

System.out.println(integerBreak.break3(10));

}

}