浦东新区2023年Python教师培训项目

第5课-Python的基础算法

课程概要

本节课主要内容是学习Python中的基础算法，递推算法（顺推法、逆推法），排序算法（选择排序，冒泡排序），递归算法。

知识点

算法是解决问题的步骤，生活、学习、编程中充满了算法

算法在不断地改进、优化中

递推算法

递推算法思维

顺推法：从已知条件出发，逐步推算出要解决的问题的方法

逆推法：从已知问题的结果出发，推算出问题的开始的条件

tips：递推算法要确保规律的准确性，顺推、逆推相互验证

递推算法重在通过已知条件，发现问题中存在的规律，根据所发现的规律编写算法，推导出最终结果，十分锻炼推理能力；并使用逆推法验证最终结果，保证规律的准确性，培养严谨思维习惯。

场景重现：

斐波那契数列：在得到斐波那契数列：1、1、2、3、5、8.....后，如何编写算法，去计算出数列中的第 36 个数字？只需要使用三个变量，就能实现算法的编写。

先使用变量 a、b 记录数列的第一个与第二个数字，然后根据数列的规律：从数列的第三个项开始，每一项都等于前两项之和。使用变量 c 去记录 a+b 的值，得到数列中的第三个数字。然后将 a、b 中保存的数值，替换成斐波那契数列的第二项和第三项，去计算数列第四项。重复执行以上的操作，便能计算出最终结果。

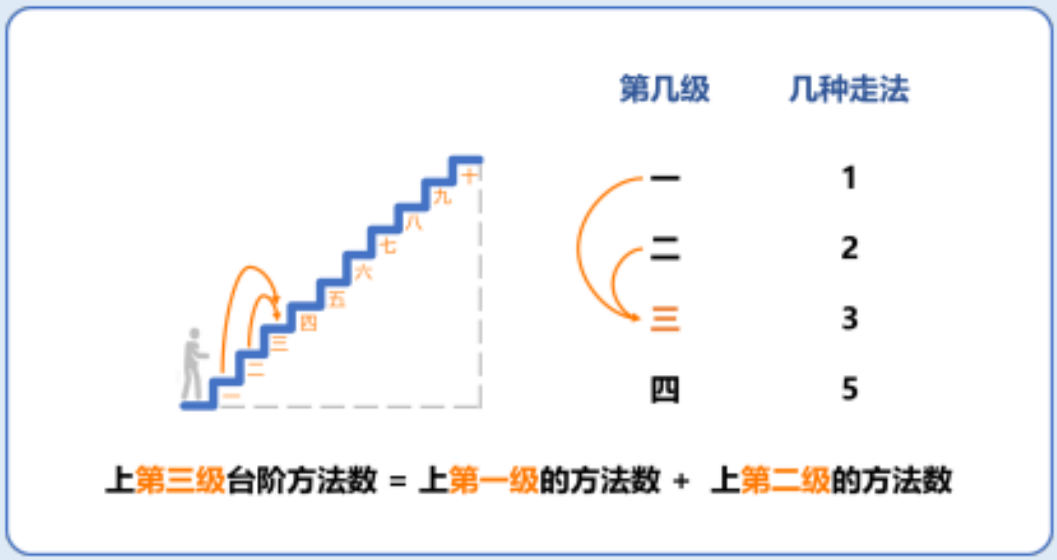
```
a = 1
b = 1
for i in range(34):
    c = a + b
    a = b
    b = c
print(c)
```

场景重现：

递推算法：假设你正在爬一段 10 级的楼梯。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

在“爬楼梯”问题中，我们发现当楼梯级数为 1、2、3、4、5 级时，爬楼梯的方法数分别为：1、2、3、5、8，符合斐波那契数列。但是，随着楼梯级数的增加，爬楼梯方法数量真的会符合斐波那契数列吗？

所以，我们使用了 逆推法 验证了这个规律。



当小人处于第三级台阶上时，它必然从第一级台阶或者第二级台阶上去的。那我们就只需要知道上到第一个级台阶与第二级台阶各自的方法数，然后相加，就能得到上第三级台阶有多少种方法了。这是符合斐波那契数列规律的。

技能小贴士：

递推算法：顺推法、逆推法

递推算法要确保规律的准确性，顺推、逆推相互验证

递归算法

描述：在函数的定义中，函数内部的语句调用函数本身（通俗点说，递归就是在函数内部调用自己的函数）

递归的特点之一：必须有一个明确的终止条件

递归函数的定义主要有以下要点：

（1）def：表示函数的关键字；

（2）函数名：函数的名称，日后根据函数名调用函数；

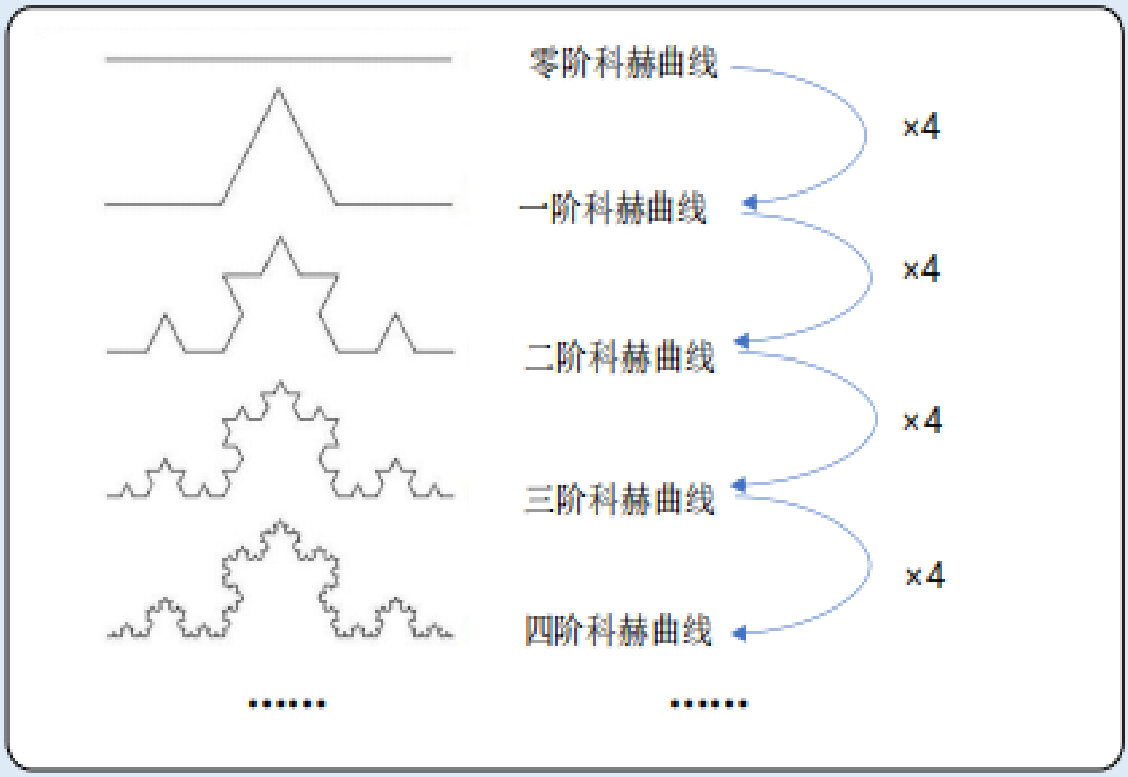
（3）函数体：函数中进行的一系列的逻辑运算；

（4）参数：为函数体提供数据；

（5）返回值：当函数执行完毕后，可以给调用者返回数据

场景重现：

科赫曲线：我们了解到一个科赫雪花是由三条科赫曲线组成的，为了画出一个科赫雪花，我们首先需要了解了科赫曲线：科赫曲线一开始是一条确定的线段（零阶科赫曲线）；将这条线段三等分，取中间一段为底边向外作等边三角形，并把底边擦掉，就会得到一条曲线（一阶科赫曲线）；将所得到的曲线的每条线段都三等分，以每条线段的中间一段为底边向外作等边三角形，并把底边都擦掉，就会得到一条曲线（二阶科赫曲线）……重复上述步骤，就会得到不同阶的科赫曲线。同时，不同阶的科赫曲线之间有一些关系：N 阶的科赫曲线可以看做是由 4 个 N-1 阶的科赫曲线组成的。不同阶的科赫曲线及其关系如下图所示：



技能小贴士：

科赫曲线的由来及简介：

瑞典数学家——海里格冯·科赫（1870-1924）在 1904 年著作了一篇名为《关于一条连续而无切线，可由初等几何构作的曲线》（法语原标题：Sur une courbe continue sans tangente, obtenue par une construction géométrique élémentaire）的论文，在这篇论文中，他描述了一种分形曲线，而这种曲线之后就以他的名字命名，即科赫曲线。

科赫曲线是最早被描述出来的分形曲线之一，这是一种类似于雪花的几何曲线，所以科赫曲线又称为雪花曲线。

场景重现：

递归算法：我们将绘制科赫曲线的代码自定义为一个 koch()函数，并在函数内部调用 koch()函数之后，运行这段代码，发现此时的程序每调用一次 koch()函数，就会向内构造一层 for 循环。

为了构造 2 层 for 循环，我们又自定义了一个变量 n，用来记录构造的 for 循环的层数，并把它作为自定义的 koch()函数的一个参数。当构造出一层 for 循环之后，就需要再次调用 koch()函数；而现在已经构造了一层 for 循环了，所以给内部的 koch()函数传递的参数就应该是 n+1。当 n 等于 2 时，就已经构造了 2 层 for 循环了，就可以不再调用 koch()函数，而是画一条直线了。n==2 就是递归终点。

```
import turtle
def koch(n):
    if n == 2:
        turtle.forward(400/3/3)
    else:
        for i in [0,60,240,60]:
            turtle.left(i)
            koch(n+1)
```

技能小贴士：

递归之所以叫做递归，是因为有不断调用函数的“递”，还要有在满足终止条件的时候的“归”；也就是说，递归必须有一个终止条件，当终止条件不满足时，递归前进；当终止条件满足时，递归返回。

递归的一些特性：

1、必须要有函数的定义；

2、必须在函数内部调用自身；

3、必须有一个明确的终止条件，否则会无限循环

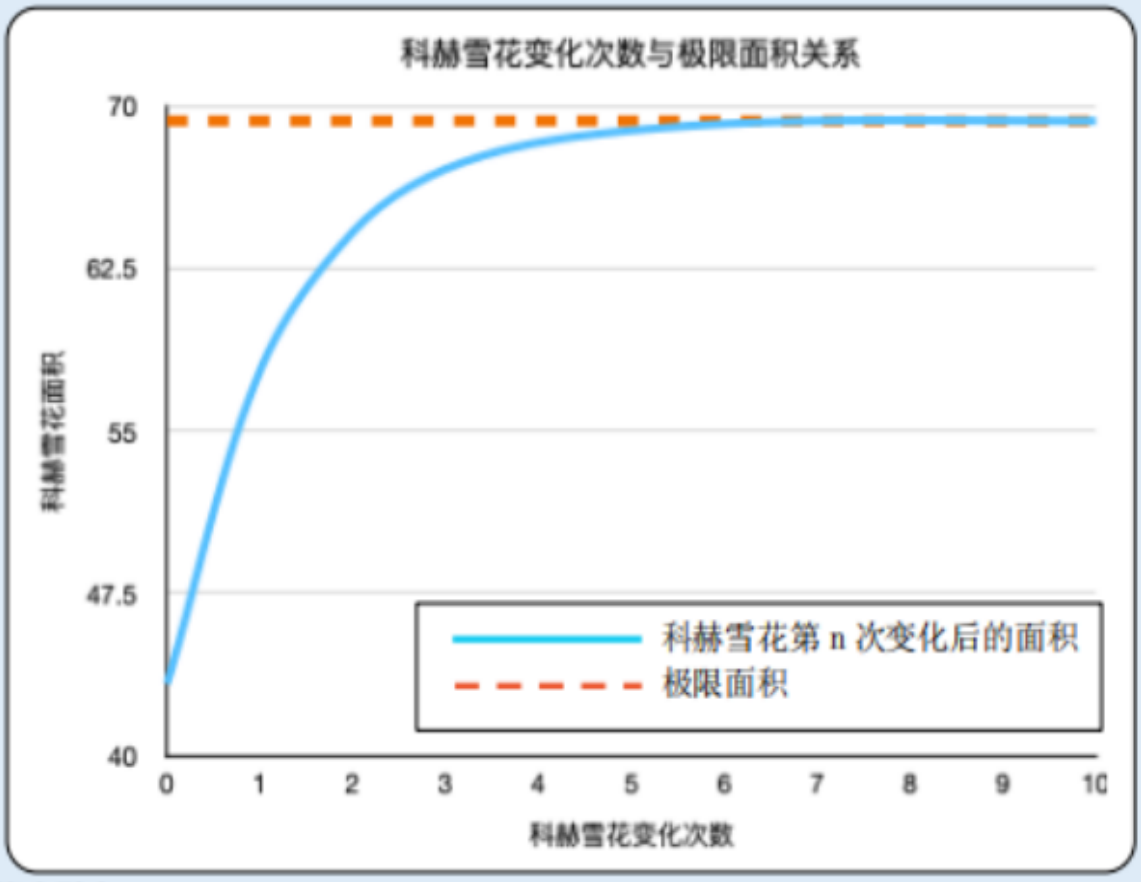
场景重现：

科赫雪花的周长与面积：假设零阶科赫曲线是 100 像素，那么经过第一次变化，得到的一阶科赫曲线是$\frac{400}{3}$像素，长度变为了原来的$\frac{4}{3}$倍；经过第二次变化，得到的二阶科赫曲线是$\frac{1600}{9}$像素，长度变为了原来的$\frac{4}{3}$倍……按照这个规律，科赫曲线的阶数每增加一阶，得到的科赫曲线的长度就会增加为原来的$\frac{4}{3}$倍。

上述规律说明：阶数越多，科赫曲线的长度就越长；也就是说，科赫曲线的长度可以无限长。既然科赫曲线的长度可以无限长，那么，由科赫曲线构成的科赫雪花的周长也是无限长的。

但是科赫雪花的面积却是固定的：通过高等数学方法，计算出：当科赫曲线的阶数趋近于无穷大时，科赫雪花的面积是初始零阶科赫雪花面积的 1.6 倍。

下图是根据边长为 10 的零阶科赫雪花经过不同次的变化的面积制成的。从下图可以看出，变化次数越多，科赫雪花的面积越趋近于一个固定值。



技能小贴士：

科赫曲线和科赫雪花有如下有趣的性质：

1、科赫曲线上任意两点之间的距离可以是无限的；

2、一个悖（bei，四声）论：无限的长度包围着有限的面积

排序算法

选择排序算法

描述：经典的排序算法之一；该算法既可以进行升序排序，也可以进行降序排序

排序原理：在升序排序中，在未排序部分中选择出来的最小值与未排序部分的第一个元素互换位置；重复上述操作，直到排序完成

冒泡排序算法

描述：经典的排序算法之一；该算法既可以进行升序排序，也可以进行降序排序

排序原理：比较未排序部分的每一对相邻元素，在升序排序中，如果前面一个比后面一个小，两者交换位置；重复上述操作，直到这一轮的未排序部分的最大值出现在未排序部分末尾；重复上述两步，直到排序完成

场景重现：

选择排序算法：在不使用min()函数、max()函数的前提下，要对列表进行排序，我们使用了选择排序算法与冒泡排序算法。其中，选择排序算法的排序思想是：选出列表未排序部分的最小值元素，然后与未排序部分的第一个元素互换位置（此为一轮）；重复上述操作，直到列表排序完成（升序）。 利用这种排序思想编写代码时，我们使用内层for循环来遍历未排序部分中，从对照项到最后的元素；又使用了外层for循环一轮一轮地遍历未排序部分，取出对照项。

```
lst = [60,80,65,54,55,78,58,74,63,70]
for j in range(len(lst)-1):
    comp_idx = j
    for i in range( j+1,len(lst)):
        if lst[comp_idx] > lst[i]:
            comp_idx = i
    lst[comp_idx],lst[j]=lst[j],lst[comp_idx]
print(lst)
```

技能小贴士：

选择排序算法的关键点：“选择”，也就是选出未排序部分的最小值。程序是怎么选择的呢？通过和对照项遍历比较选择出来的。

对照项：（升序）

（1）选择排序算法取未排序部分的第一个元素做对照项；

（2）如果后面的元素小于对照项，那么对照项就变成后面的那个元素；

（3）当对照项和未排序部分的其他元素全部比较完之后，此时的对照项就是未排序部分的最小值元素

场景重现：

冒泡排序算法：冒泡排序的排序思想是：对比未排序部分中的每一对相邻元素，如果前面的元素比后面的元素大，两个元素就交换位置，到一轮结束，未排序部分中的最大值会出现在未排序部分的最后面；重复上述操作，直到列表排序完成。（升序）
利用这种排序思想编写代码时，我们使用内层for循环来遍历未排序部分的每个元素，然后比较每个元素与它后面一个元素的大小；又使用了外层for循环一轮一轮地遍历未排序部分。

```
lst1 = [9,8,7,6,5,4,3,2,1,0]
for idx in range(len(lst1)-1):
    for i in range(0,len(lst1) -1 - idx):
        if lst1[i] > lst1[i+1]:
                lst1[i],lst1[i+1]=lst1[i+1],lst1[i]
print(lst1)
```

技能小贴士：

冒泡排序小知识：

冒泡排序为什么叫这个名字呢?

这是因为，在升序排序中，未排序部分的最大元素会慢慢交换到列表的后面，直到出现在未排序部分最后；就像水中的大气泡上升最终浮出水面一样。

相关学习网站

菜鸟教程：https://www.runoob.com/python3/python3-class.html

turtle：https://www.cnblogs.com/chen0307/articles/9645138.html