云南大学数学与统计学院
上机实践报告

课程名称： 数据结构与算法实验	**年级：** 2015 级	**上机实践成绩：**
指导教师： 陆正福	**姓名：** 刘鹏	
上机实践名称： 递归实验	**学号：** 20151910042	**上机实践日期：** 2017-04-10
上机实践编号： No.4	**组号：**	**上机实践时间：** 上午三、四节

一、实验目的

1. 熟悉递归算法设计模式
2. 熟悉 Python 递归程序设计
3. 熟悉主讲教材 Chapter 4 的代码片段

二、实验内容

1. 递归有关的数据结构设计与算法设计；
2. 调试主讲教材 Chapter 4 的 Python 程序；
3. （选做） 任选一个算法,写出其递归版本与非递归版本,总结不同设计与实现的优缺点。

三、实验平台

Windows 10 Enterprise 中文版；
Python 3.6.0；
Wing IDE Professional 6.0.2-1 集成开发环境。

四、实验记录与实验结果分析

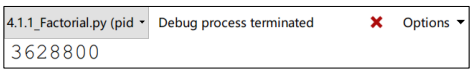
题 1.

4.1.1 The Factorial Function（递归函数）

程序代码：

```
1  # filename: 4.1.1
2
3  def factorial(n):
4      if n == 0:
5          return 1
6      else:
7          return n * factorial(n - 1)
8
9  print(factorial(10))
```

程序代码 1



运行结果 1

题 2

Drawing an English Ruler

*画一个英式直尺

In the case of computing a factorial, there is no compelling reason for preferring recursion over a direct iteration with a loop. As a more complex example of the use of recursion, consider how to draw the marking of a typical English ruler. For each inch, we placing a tick with a numeric label. We denote the length of the tick designating a whole inch as the major tick length. Between the marks for whole inches, the ruler contains a series minor ticks, placed at intervals of 1/2 inch, 1/4 inch, and so on. As the size of the interval decreases by half, the tick length decreases by one.

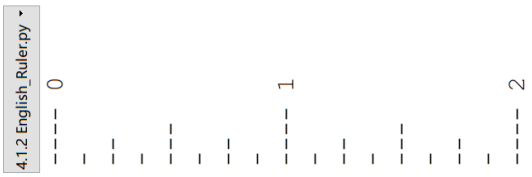
*在计算阶乘函数的例子中，我们选择递归而不选择循环是没有强制性理由的。现在给出一个使用递归的更加复杂的例子，那就是绘制典型的英文标尺的刻度标记。每增加一英尺，我们都要放置一个带有数字编号的刻度标签。我们将长度为一英尺的刻度设置为主刻度。在一个单位英尺的刻度线之间，还包含一些列小的

刻度线，如二分之一英尺、四分之一英尺等等。刻度线的间隔大小减少一半，刻度线的高度的减少 1。

程序代码：

```
# 4.1.2 English Ruler

def draw line(tick length,tick label=''):
    line = '-' * tick length
    if tick label:
        line += ' ' + tick label
    print(line)

def draw interval(center length):
    if center length > 0:
        draw interval(center length - 1)
        draw line(center length)
        draw interval(center length - 1)

def draw ruler(num inchs,major length):
    draw line(major length,'0')
    for j in range(1,1 + num inchs):
        draw interval(major length - 1)
        draw line(major length,str(j))

draw_ruler(2,4)
```

程序代码 2



运行结果 2（This photo has been rotated by 90 degrees）

The English ruler pattern is a simple example of a ***fractal***, that is, a shape that has a self-recursive structure at various levels of magnification.

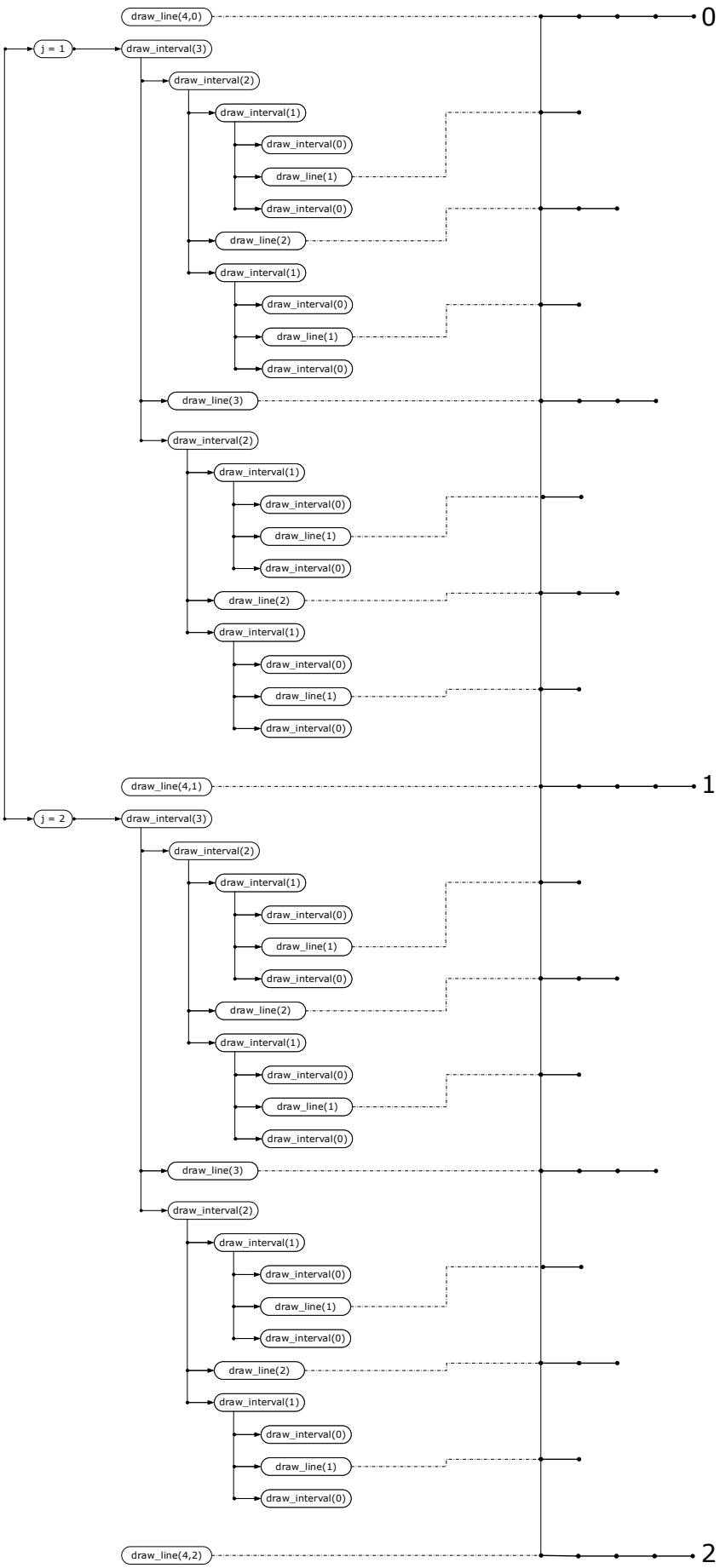
*英式直尺上的刻度排布是一个分形的简单例子，而所谓的分形，指的就是一个在不同的放大倍率有着自我递归结构的图形。

The execution of the recursive `draw_interval` function can be visualized using a recursion trace. The trace for `draw_interval` is more complicated than in the factorial example, however, because each instance makes two recursive calls. To illustrate this, we will show the recursion trace in a form that is reminiscent of an outline for a document.

*递归式函数 `draw_interval` 的执行可以通过一个递归追踪而变得可视化。但是在这个函数里的追踪比阶乘函数里的要复杂，因为每递归一次都产生两个递归调用。为了表述清楚这件事，我们使用一种类似于文档大纲的形式来展示递归追踪。

分析：

可以很明显看出，递归就是不断地铺张内存，而循环就不会。一次递归就展开一段内存，当到了最后不满足判断条件，就开始收敛内存区域。正因如此，流程图也很好画。


draw_line(4,0)
0
j = 1
draw_interval(3)
draw_interval(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(3)
draw_interval(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(4,1)
1
j = 2
draw_interval(3)
draw_interval(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(3)
draw_interval(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(2)
draw_interval(1)
draw_interval(0)
draw_line(1)
draw_interval(0)
draw_line(4,2)
2

题 3

In this section, we describe a classic recursive algorithm, binary search, that is used to efficiently locate a target value within a sorted sequence of n elements. This is among the most important of computer algorithms, and it is the reason that we so often store data in sorted order.

*在这一节中，我们将会描述一个经典的递归算法，那就是用于在有序序列中高效定位目标元素的二元搜索算法。该算法是计算机算法中最重要的一个之一，而这也是我们总是将数据有序存放的原因。

When the sequence is ***unsorted***, the standard approach to search for a target value is to use loop to examine every element, until either finding the target or exhausting the data set. This is known as the ***sequential search*** algorithm runs in $O(n)$ time (i.e., linear time) since every element is inspected in the worst case.

*当序列处于无序状态时，标准的做法是通过循环检查每一个元素是否匹配，直到找到目标元素或者穷尽整个序列（译者按：当然不排除最后一个元素才找到）。这被称为序列式搜索算法，因为每一个元素都需要被检视，所以时间复杂度是$O(n)$，也就是线性时间复杂度。

When the sequence is ***sorted*** and ***indexable***, there is a much more efficient algorithm. (For intuition, think about how you would accomplish this task by hand!) For any index j, we know that all the values stored at indices $0, \cdots, j-1$ are less than or equal to the value at index j, and all the values stored at indices $j+1, \cdots, n-1$ are greater than or equal to that at index j. This observation allows us to quickly "home in" on a search target using a variant of the children's game "high-low." We call an element of the sequence a ***candidate*** if, at the current stage of the search, we cannot rule out that this item matches the target. The algorithm maintains two parameters, low and high, such that all the candidate entries have index at least low and at the most high. Initially, low = 0 and high = $n-1$. We then compare the target value to the median candidate, that is, the item data[mid] with index

$$\text{mid} = \lfloor (\text{low} + \text{high})/2 \rfloor$$

*当序列是有序的而且是有下标的那种时，我们就可以采取一个更加高效的算法。（从直观上想一想你可以采取什么措施手动完成这个任务吧！）对于随便一个下标j，我们知道所有以$0, \cdots, j-1$为下标的元素数值都不大于下标为j的元素，而所有以$j+1, \cdots, n-1$为下标的元素数值都不小于下标为j的元素。这个发现使我们也已采取一种类似于小孩子那种“高还是低”的游戏进行快速定位。在搜索进程的当前状态下，我们把序列中的一个元素设置为候选值，当然我们肯定不能保证这就是我们需要的那个数值。这个算法包含两个参数，那就是 low 和 high，所有候选值的下标最小为 low，最大为 high。我们把 low 的初始值设置为 0，high 的初始值设置为$n-1$。在这之后，我们将目标值与下标处于 low 与 high 的中间值的那个数值进行比较。

程序代码：

```
1   # 4.1.3 Binary Search
2
3   def binary search(data,target,low,high):
4       if low > high:
5           return False
6       else:
7           mid = (low + high) // 2
8           if target == data[mid]:
9               return True
10          elif target < data[mid]:
11              return binary search(data,target,low,mid-1)
12          else:
13              return binary search(data,target,mid + 1, high)
14
15  a = [1,2,3,5,8,15,45,666,3333,6222,9111]
16  print(binary_search(a,9111,0,11))
```

程序代码 3

```
4.1.3_Bianry_Search.py
True
```

运行结果 3

4 题

Modern operating system defines file-system directories (which are also sometimes called "folders") in a

recursive way. Namely, a file system consists of a top-level directory, and the contents of this directory consists of files and other directories, which in turn can contain files and other directories, and so on. The operating system allows directories to be nested arbitrarily deep (as long as there is enough space in memory), although there must necessarily be some base directories that contain only files, not further subdirectories.

*现代操作系统都用递归的方式定义了文件系统目录，或者称之为文件夹。换句话说，一个文件系统包含一个顶层目录，这个顶层目录的内容包括文件以及其他的目录，而这些被顶层目录包含的子目录也可以包含其他的文件与目录。文件系统允许这样的目录结构可以无限制地嵌套下去，当然前提是有足够的存储空间。但是总会有一些基本的底层目录，它们之下只有文件而没有子目录。（译者按：这也不一定吧，毕竟有空目录这种存在）

Python's os Module

To provide a Python implementation of a recursive algorithm for computing disk usage, we rely on Python's os module, which provides robust tools for interacting with the operating system during the execution of a program. This is an extensive library, but we will only need the following four function:

*为了能用 Python 实现这个计算磁盘用量的递归算法，我们需要 Python 下的一个名为 os 的模块。这个模块可以为我们的程序执行提供一种稳健的与操作系统交互的工具。这是一个扩展库，但是我们仅仅需要它下面的四个函数。

- **os.path.getsize(path)**
  Return the immediate disk usage (measured in bytes) for the file or directory that is identified by the string path (e.g., `/user/rt/courses`)
  *os.path.getsize(path)这个函数可以返回一个文件夹或者一个文件的磁盘使用量。而这个文件夹或者这个文件是通过路径的字符串形式给出的。

- **os.path.isdir(path)**
  Return True if entry designated by string path is a directory; False otherwise.
  *当 path 这个字符串所指定的目标是一个目录的时候，这个函数就会返回 True，否则就会返回 False。

- **os.listdir(path)**
  Return a list of strings that are the names of all entries within a directory designated by string path. In our sample file system, if the parameter is `/user/rt/courses`, this returns the list `['cs016','cs 252'].`
  *该函数返回的是 path 目录下的所有文件或者子目录。在我们的样本文件系统里，如果 path 是/user/rt/courses，那么就返回 ['cs016','cs 252'] 。

- **os.path.join(path,filename)**
  Compose the path string and filename string using an appropriate operating system separator between the two (e.g., the / character for a Unix/Linux system, and the \ character for Windows) Return the string that represents the full path to the file.
  *这个函数使用与操作系统相匹配的分隔符（比如 Unix/Linux 系统下的"/"，Windows 操作系统下的"\"），将路径与文件名组合成一个完成的文件路径，然后返回该完整路径所对应的字符串。

程序代码：

```
1   # 4.1.4 File System
2
3   import os
4
5   def disk usage(path):
6       total = os.path.getsize(path)
7       if os.path.isdir(path):
8           for filename in os.listdir(path):
9               childpath = os.path.join(path,filename)
10              total += disk usage(childpath)
11      print('{0:<7}'.format(total),path)
12      return total
13
14  path = input('Please input the path\n')
15  print(disk usage(path),' byte')
```

程序代码 4

```
4.1.4_File_System.py ( Debug I/O (stdin, stdout, stderr) appears below
Please input the path
D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1
58014    D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验1-高级语言基本编程实验.docx
226987   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验2-高级语言面向对象编程实验.docx
318170   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验3-算法分析.docx
622107   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验3-算法分析.pdf
164907   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验4-递归实验.docx
465663   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验4-递归实验.pdf
45588    D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验5-数组序列实验.docx
23515    D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验6-栈与队列实验.docx
23519    D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\20151910042-刘鹏-DSA实验7-链表与表的抽象化实验.docx
46       D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\desktop.ini
3826     D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image\Binary_Search.png
10581    D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image\BubbleSort.png
5947     D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image\English_Ruler.png
8810     D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image\Factorial.png
11061    D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image\QuickSort.png
193867   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image\QuickSort_many.png
238188   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\image
162      D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\~$151910042-刘鹏-DSA实验4-递归实验.docx
162782   D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\~WRL0005.tmp
2353744 D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1
2353744  byte
```

运行结果 4

```
Windows PowerShell
Windows PowerShell
版权所有 (C) 2016 Microsoft Corporation。保留所有权利。

PS C:\Users\newton> cd D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1\
PS D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1> tree /f >dir.txt
PS D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1> type .\dir.txt
卷 WorkStation 的文件夹 PATH 列表
卷序列号为 0000006D 5CF6:07DF
D:.
    20151910042-刘鹏-DSA实验1-高级语言基本编程实验.docx
    20151910042-刘鹏-DSA实验2-高级语言面向对象编程实验.docx
    20151910042-刘鹏-DSA实验3-算法分析.docx
    20151910042-刘鹏-DSA实验3-算法分析.pdf
    20151910042-刘鹏-DSA实验4-递归实验.docx
    20151910042-刘鹏-DSA实验4-递归实验.pdf
    20151910042-刘鹏-DSA实验5-数组序列实验.docx
    20151910042-刘鹏-DSA实验6-栈与队列实验.docx
    20151910042-刘鹏-DSA实验7-链表与表的抽象化实验.docx
    dir.txt

└─image
        Binary_Search.png
        BubbleSort.png
        English_Ruler.png
        Factorial.png
        File_System.png
        QuickSort.png
        QuickSort_many.png

PS D:\myNutshellCloud\myStudyMaterial\Grade_2_Term_2\#Data_Structure_And_Algorithm_Experiment\version1>
```

五、实验体会

Translation

Chapter 4 Recursion
*第四章 递归

One way to describe repetition within a computer program is the use of loops, such as Python's while-loop and for-loop constructs described in Section 1.4.2. An entirely different way to achieve repetition is through a process known as ***recursion***.
*循环是一种在计算机程序中描述重复行为的方法，正如在 1.4.2 节描述的，Python 中就有 while 循环与 for 循环。另一种完全不同的但却同样可以实现重复性操作的方式叫做递归。

Recursion is a technique by which a function makes one or more calls to itself during execution, or by which a data structure relies upon smaller instances of the very same type of structure in its representation. There are many examples of recursion in art and nature. For example, fractal patterns are naturally recursive. A physical example of recursive used in art is in the Russian Matryoshka dolls. Each doll is either make of solid wood, or is hollow and contains another Matryoshka doll inside it.
*通过递归这种方法，一个函数可以进行一次或者多次的自我调用，一个数据结构也可以依靠与其同类型的、但是规模较小的实例来实现自我构建。在艺术与自然的领域里，有很多递归的例子。举个例子，分形艺术就是自然递归。另一个在艺术中的递归的实例是俄罗斯套娃。套娃的一个，要么是实心的，要么是空心的，而且空心的这种里面还有一个套娃。

In computing, recursion provides an elegant and powerful alternative for performing repetitive tasks. In fact, a few programming languages (e.g., Scheme, Smalltalk) do not explicitly support looping constructs and instead rely directly on recursion to express repetition. Most modern programming languages support functional recursion using the identical mechanism that is used to support traditional forms of function calls. When one invocation of the function make a recursive call, that invocation is suspended until the recursive call completes.
*在计算中，递归为执行重复性任务提供了一种优雅而强有力的选择。事实上，有一小部分编程语言并不直接支持循环结构，比如说 Scheme 或 Smalltalk，反而是直接靠递归来表达重复性操作。大多数的现代编程语言都采用了与传统的函数调用上的相同的机制，实现了函数性递归。当函数的一个调用进行递归式调用时，这个调用就被暂时停下，知道递归调用完成。

Recursion is an important technique in the study of data structures and algorithms. We will use it prominently in several later chapters of this book (most notably, Chapter 8 and 12). In this chapter, we begin with the following four illustrative examples of the use of recursion, providing a Python implementation for each.
*对于数据结构与算法的学习而言，递归这种方法十分重要。我们将会在接下来的几个章节中（特别是第八章与第十二章）突出使用它。着这一章中，我们从这四个说明性的例子开始学习递归的使用，而且我们给出了它们在 Python 下的实现。

- The ***factorial function*** (commonly denoted as $n!$) is a classic mathematical function that has a natural recursive definition.
  *阶乘函数是一个拥有自然递归定义的经典数学函数。
- An ***English ruler*** has a recursive pattern that is a simple example of a fractal structure.
  *英式直尺的刻度线有着递归模式，这种模式是分形结构的一个很简单的例子。
- ***Binary search*** is among the most important computer algorithms. It allows us to efficiently locate a desired value in a data set with upwards of billions of entries.
  *二元搜索是计算机算法领域里最重要的算法之一。这个算法使得我们可以在无数数据中高效定位目标值。
- The ***file system*** for a computer has a recursive structure in which directories can be nested arbitrarily deeply within other directories. Recursive algorithms are widely used to explore and manage these system.
  *计算机中的文件系统也具有递归结构，可以在其他目录中任意嵌套目录。递归算法被广泛用于探索以及管理这些系统。

We then describe how to perform a formal analysis of the running time of a recursive algorithm and we discuss some potential pitfalls when defining recursions. In the balance of the chapter, we provide many more examples of recursive algorithm, organized to highlight some common forms of design.

*在这之后，我们将叙述一下如何对一个递归算法的时间复杂度进行正式的分析，而且我们也将会讨论许多在定义递归的时候所面临的陷阱。在本章的剩余部分，我们将会提供递归算法的更多实例，借此突出许多常见的设计形式。

END

在本章的学习中，我们重点讨论了递归算法的实现，讲了几个重点例子。其中包括 English Ruler 的复杂递归实例。递归的核心在内存的铺展上。明白了这一点，就可以很快画出递归展开的踪迹图，然后就可以理解这个算法。

六、参考文献

[1] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, *Data Structures and Algorithms in Python*, Chapter 4
[2] 实验教材：汪萍，陆正福等编著，数据结构与算法的问题与实验 第 1 章