程序设计

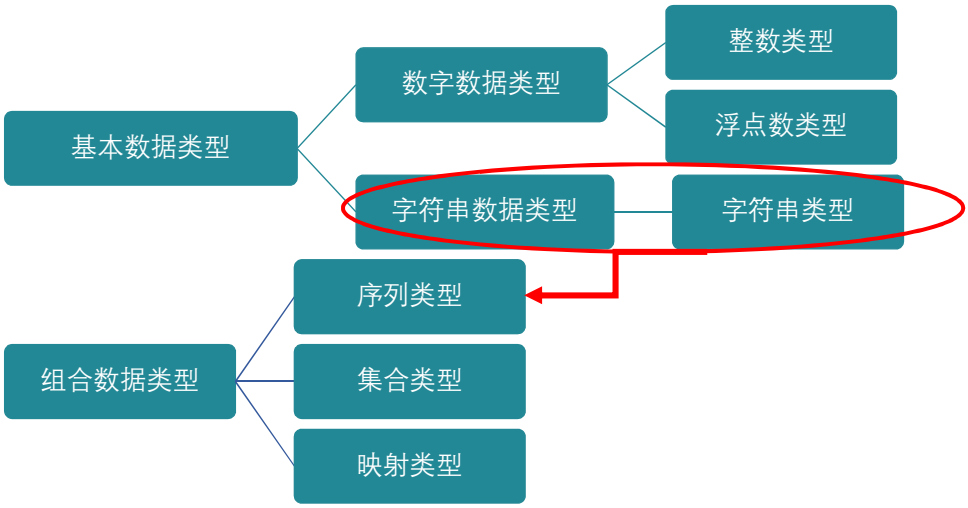
代码复用和模块化设计

import A

from A import B

import A as B

组合数据类型



序列类型

序列类型有12个通用的操作符和函数

操作符	描述
x in s	如果x是s的元素，返回True，否则返回False
x not in s	如果x不是s的元素，返回True，否则返回False
s + t	连接s和t
s * n 或 n * s	将序列s复制n次
s[i]	索引，返回序列的第i个元素
s[i: j]	分片，返回包含序列s第i到j个元素的子序列（不包含第j个元素）
s[i: j: k]	步骤分片，返回包含序列s第i到j个元素以j为步数的子序列
len(s)	序列s的元素个数（长度）
min(s)	序列s中的最小元素
max(s)	序列s中的最大元素
s.index(x[, i[, j]])	序列s中从i开始到j位置中第一次出现元素x的位置
s.count(x)	序列s中出现x的总次数

列表的定义

- 列表是值的序列。
- 在字符串中，这些值是字符，在列表中，它可以是任何类型。
- 列表中的值称为元素（element），有时也叫列表项（item）。
- 列表用中括号（[]）表示。

列表的操作——添加

在列表末尾添加，使用.append()方法；

在列表中插入元素，使用insert()方法，需指定新元素的索引和值；

列表的操作——修改

与字符串不同，列表元素的值是可变的：

```
a = ['Spam', 2.0, 5, [10, 20]]
a[1] = 3
```

列表的操作——删除

使用del语句删除元素；

使用pop()方法删除列表末尾的元素，并返回其值；

使用pop()方法删除指定位置的元素；

使用remove()方法，根据值删除元素；

列表的操作——其他方法

使用方法 sort()对列表进行永久性排序；

使用函数 sorted()对列表进行临时排序；

使用reverse()方法倒序打印列表；

使用len()函数，确定列表长度。

列表的遍历

如何进行列表的遍历？

```
a = ['Spam', 2.0, 5, [10, 20]]
for i in a:
    print(id)
```

元组

列表非常适合用于存储在程序运行期间可能变化的数据集。列表是可以修改的，这对处理网站的用户列表或游戏中的角色列表至关重要。然而，有时候你需要创建一系列不可修改的元素，元组可以满足这种需求。 Python将不能修改的值称为不可变的，而不可变的列表被称为元组。

元组

元组看起来犹如列表，但使用圆括号而不是方括号来标识。定义元组后，就可以使用索引来访问其元素，就像访问列表元素一样。

```
a = tuple()
a = ('student','teacher',12, [12,'a','c'])
a[1]
a[1] = '0'
```

元组的作用

1. 函数多返回值
2. 多变量同步赋值
3. 遍历循环

字典概念

字典类似于列表，但更加通用；

在列表中，下标是整数，在字典中，下标可以是任意类型；

字典包含下标（键）集合和值集合；

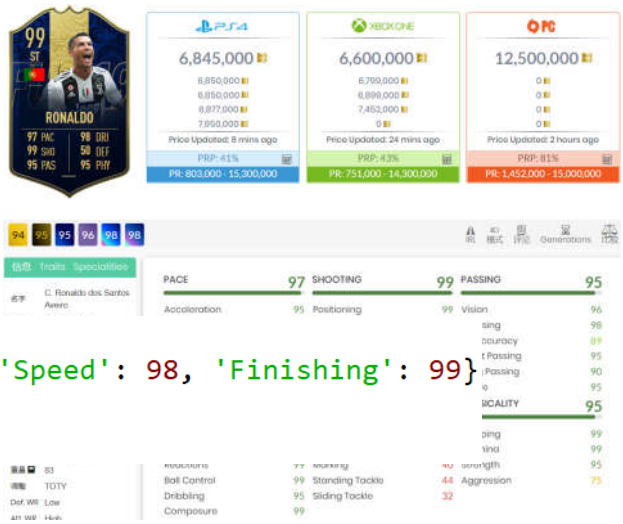
键与值之间的关联被称为键值对；

字典体现的是键值对的映射；

字典可以高效模拟现实世界的情景。

字典案例

游戏中，C罗的各个属性值是不同的，可以用字典存储部分属性的值：




```
# 字典案例
CR_7 = {'Club': 'Juventus', 'Speed': 98, 'Finishing': 99}
print(CR_7['Club'])
print(CR_7['Speed'])
```

字典的创建

```
# 字典案例
CR_7 = {'Club': 'Juventus', 'Speed': 98, 'Finishing': 99}
print(CR_7['Club'])
print(CR_7['Speed'])
```

用花括号{}中的键-值对表示；

键值之间用冒号分隔，键-值对之间用逗号分隔；

字典值的访问

```
# 字典案例
CR_7 = {'Club': 'Juventus', 'Speed': 98, 'Finishing': 99}
print(CR_7['Club'])
print(CR_7['Speed'])
```

要获取与键相关联的值，可依次指定字典名和放在方括号内的键

字典键-值对的添加、修改、删除

字典是一种动态结构，可随时在其中添加键—值对；

要添加键—值对，可依次指定字典名、用方括号括起的键和相关联的值；

```
# 字典案例
CR_7 = {'Club': '尤文图斯', 'Speed': 98, 'Finishing': 99}
print(CR_7['Club'])
print(CR_7['Speed'])

# 字典的添加（添加键值对）
CR_7['Heading'] = 95
CR_7['Nationality'] = '葡萄牙'
```

字典键-值对的添加、修改、删除

创建空字典：使用字典来存储用户提供的数据或在编写能自动生成大量键—值对的代码时，通常都需要先定义一个空字典。

```
# 创建空字典
Messi = {}
```

字典键-值对的添加、修改、删除

修改字典中的值，可依次指定字典名、用方括号括起的键以及与该键相关联的新值。

```
# 修改字典值
CR_7['Speed'] = 90
```

字典键-值对的添加、修改、删除

删除键值对，可使用del语句将相应的键—值对彻底删除；

使用del语句时，必须指定字典名和要删除的键；

字典遍历

遍历字典有多种方式：可遍历字典的所有键—值对、键或值：

1. 要编写用于遍历字典的for循环，可声明两个变量，用于存储键—值对中的键和值。对于这两个变量，可使用任何名称。使用items()方法访问键值对
2. 在不需要使用字典中的值时，使用方法keys()遍历键。
3. 遍历值可使用方法values()，返回一个值列表，而不包含任何键。

字典遍历

```
for key,value in CR_7.items():
    print(key)
    print(value)
```

```
for value in CR_7:
    print(value)
```

```
for value in CR_7.values():
    print(value)
```

```
for key in CR_7.keys():
    print(key)
```

字典嵌套

将一系列字典存储在列表中，或将列表作为值存储在字典中，这称为嵌套。

1. 在列表中嵌套字典
2. 在字典中嵌套列表
3. 在字典中嵌套字典

安装三方库

pip命令

算法

算法概念

名人名言

尼古拉斯·沃斯，瑞士计算机科学家。1958年，Niklaus从苏黎世工学院取得学士学位后来到加拿大的莱维大学深造，之后进入美国加州大学伯克利分校获得博士学位。于1984年获得图灵奖。

程序=算法+数据结构

学习的算法意义

将科学问题通过算法+数据结构，转换成一个程序，并通过计算机求解。

学习的算法意义

算法=内功

编程语言=招式

算法特性

- 确定性
- 有穷性
- 有效性
- 输入（0或多个）
- 输出（至少一个）

算法分类

- 数值算法
  - 对数值进行求解
- 非数值算法
  - 排序、查找等“处理”问题

算法结构

20世纪70年代，荷兰科学家，迪杰斯特拉提出结构化程序设计思想，包括：

1. 一般程序由三种基本结构组成
2. 程序设计自顶向下进行

算法表示

算法的表示是指以某种形式来表达算法，有以下几种常用表示方法：

- 自然语言
- 传统的流程图
- 结构流程图
- 伪代码
- PAD图

算法表示——自然语言

求解N！的算法：

Step 1: P=1

Step 2: I=1

Step 3: 输入N

Step 4: 使P*I，乘积仍放在P中

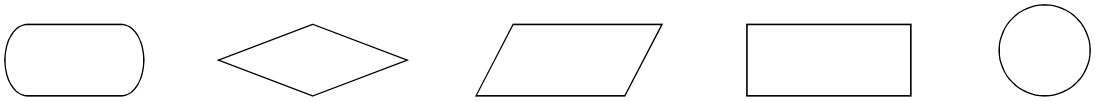
Step 5: 使I 的值加1，再放回到I中

Step 6: 如果I小于等于N，返回Step 4执行，否则，进入下一步Step 7

Step 7: 输出P中存放的N！值

算法表示——流程图

- 使用几何图形表示算法开始到结束的过程
- 基本流程图符号如下图所示



开始/结束　条件判断　输入/输出　过程　连接

算法表示——流程图

求解N！的算法：

Step 1: P=1

Step 2: I=1

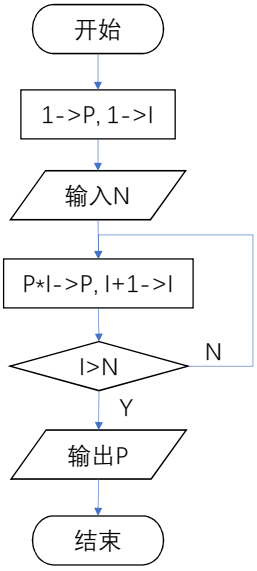
Step 3: 输入N

Step 4: 使P*I，乘积仍放在P中

Step 5: 使I 的值加1，再放回到I中

Step 6: 如果I小于等于N，返回Step 4执行，否则，进入下一步Step 7

Step 7: 输出P中存放的N！值



算法表示——伪代码

- 一种类似于英语的表示法
- 没有固定的标准
- 包含以下部分：
  - 算法头
  - 输入值和返回值

Algorithm 1: GND Algorithm

Input: $G(V,E)$, $R^{\#}$, $\mathcal{E}$
Output: $\hat{R}$

$\hat{R} \leftarrow \emptyset$;
$\mathcal{V} \leftarrow V$;
for $i \leftarrow 1$ **to** $R^{\#}$ **do**
  for *each* $j \in \mathcal{V}$ **do**
    $\mathcal{E}_j \leftarrow \mathcal{E}_j + \beta$;
    $F_j \leftarrow runLP(\mathcal{E}_j)$;
    $\mathcal{E}_j \leftarrow \mathcal{E}_j - \beta$;
  end
  $n^* \leftarrow maxLocation(\{F_j \mid j \in \mathcal{V}\})$;
  $\mathcal{E}_{n^*} \leftarrow \mathcal{E}_{n^*} + \beta$;
  $\hat{R}.add(n^*)$;
  $\mathcal{V}.delete(n^*)$;
end

算法表示——伪代码

求解N！的算法：

Step 1: P=1

Step 2: I=1

Step 3: 输入N

Step 4: 使P*I，乘积仍放在P中

Step 5: 使I 的值加1，再放回到I中

Step 6: 如果I小于等于N，返回Step 4执行，否则，进入下一步Step 7

Step 7: 输出P中存放的N！值

```
Start
    set p = 1
    set i = 2
    input n
    while I <= n do
        p = p*i
        i = i+1
    end while
    print p
End
```

算法的发现

- 杨辉三角
- 九章算术
- 欧几里德辗转相除法

排序与查找

排序算法

排序是迭代的延续;

将一组数据按递增或递减重新排列;

不仅用在数值方面，也用在文本方面;

常用方法:

选择排序

冒泡排序

排序算法—选择排序

- 把表中最大的数找到并放入第一个位置
- 比较余下的数，找到次大的数放到第二个位置
- 直到对所有数据全部扫描过

排序算法—选择排序

- 按从大到小排序下列数字：2, 12, 5, 56, 34, 78

2	12	5	56	34	78	从大到小排序一组数
78	12	5	56	34	2	第一次扫描，78和2互换
78	56	5	12	34	2	第二次扫描，56和12互换
……						
78	56	34	12	5	2	第五次扫描，得到排序结果

排序算法—冒泡排序

- 开始比较相邻的两个数，将较小的向前移动，所有数比较完毕，得到最大的数在第最后一个位置
- 剩下的数，持续这个过程，找到的次大的数排到列表的倒数第二个位置
- 依次类推，直到结束

排序算法—冒泡排序

- 按从小到大排序下列数字：2, 12, 5, 56, 34, 78

2	12	5	56	34	78	2和12比，不交换
2	5	12	56	34	2	5和12比，交换
2	5	12	56	34	2	56和12比，不交换
2	5	12	34	56	2	56和34比，交换
2	5	12	34	2	56	56和2比，交换

查找算法

- 把一个特定的数据从列表中找到并提供它所在的位置;
- 常用方法:
  - 顺序查找
  - 折半查找

查找算法—顺序查找

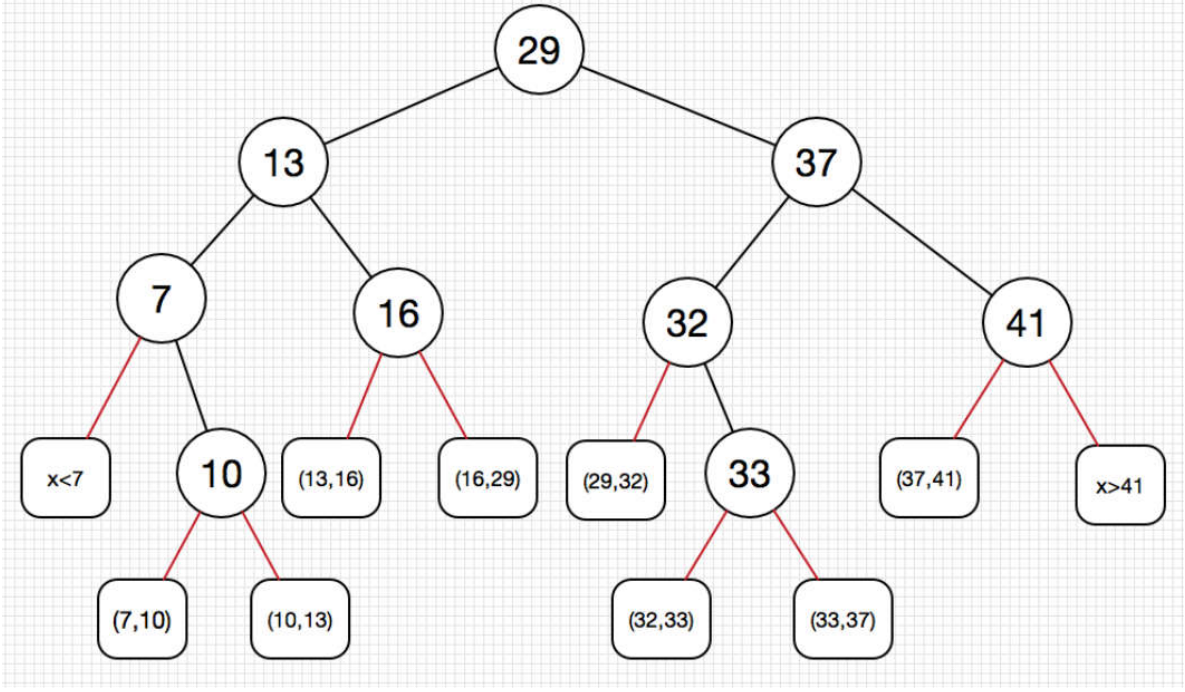
- 从列表中的第一个数据开始，当给定的数据和表中的数据匹配时，查找过程结束，给出这个数据所在表中的位置。

查找算法—折半查找

- 也叫二分法，从列表的一半开始，比较列表处于中间位置的数据，判断是在前半部分还是后半部分（根据列表的排序确定）

折半查找平均查找次数

7,10,13,16,29,32,33,37,41　　查找33 ？



折半查找平均查找次数

查找成功：

(1+2+2+3+3+3+3+4+4)/9=25/9

查找不成功：

(3+4+4+3+3+3+4+4+3+3)/10=3.4