

The background features a light gray field with several decorative elements: teal diamonds, thin gray lines, and colorful abstract shapes in orange, red, and blue. Some of these shapes have a white dotted pattern. The title text is centered in the upper half of the image.

# Python 语言程序设计

# 第一章

## 程序与算法



01 程 序

02 算 法



# 01 程序



# 语言的演变

## 程序语言的演变

- 编程其实就是用计算机语言把人类的需求表达出来。
- 计算机语言（Computer Language）是人与计算机之间交流的媒介。
- 计算机语言经历了从机器语言、汇编语言，再到高级语言的演变过程。

机器语言

汇编语言

高级语言

操作：寄存器 BX 的内容送到 AX 中

1	1000100111011000	机器指令
2	MOV AX, BX	汇编指令
3	AX = BX	高级语言

# 语言的演变

计算机语言	编写方式及要素	特点
机器语言	二进制编码 操作码、地址码	速度快，效率高，占用内存少 直观性差，难以纠错，编写需要很强的专业性
汇编语言	助记符号 操作码、地址码	速度快，效率高，占用内存少，直观性较强 编写专业性较强
高级语言	接近自然语言的语法 源程序，编译或解释程序	占用内存多，执行需要编译 易于掌握，可读性强 独立性、共享性及通用性强

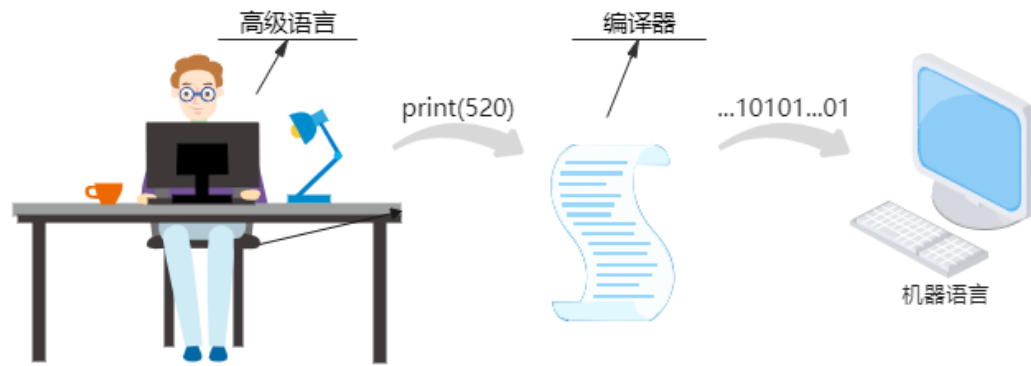
# 高级语言的运行机制

## 高级语言按照执行方式可以分为 编译型

- 编译程序对源程序进行解释的方法相当于日常生活中的“整文翻译”。

## 解释型

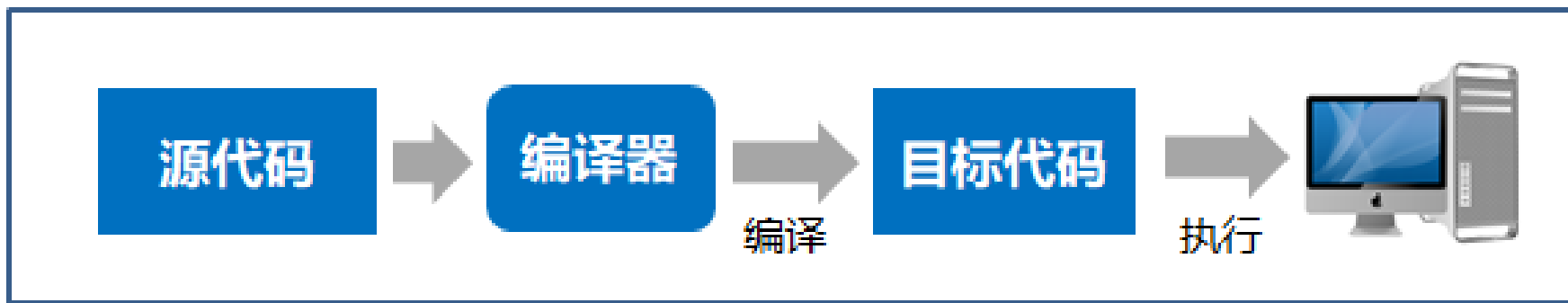
- 解释程序对源程序进行翻译的方法相当于日常生活中的“同声传译”。



# 高级语言的运行机制

编译型语言具有如下优点：

- 可独立运行，源代码经过编译形成的目标程序可脱离开发环境独立运行；
- 运行效率高，编译过程包含程序的优化过程，编译的机器码运行效率较高。



# 高级语言的运行机制

解释型语言的优点如下：

- 易于修改和测试，逐句解释过程中便于对代码的修改和测试；
- 可移植性较好，只要有解释环境，可在不同的操作系统上运行。



# 02 算法

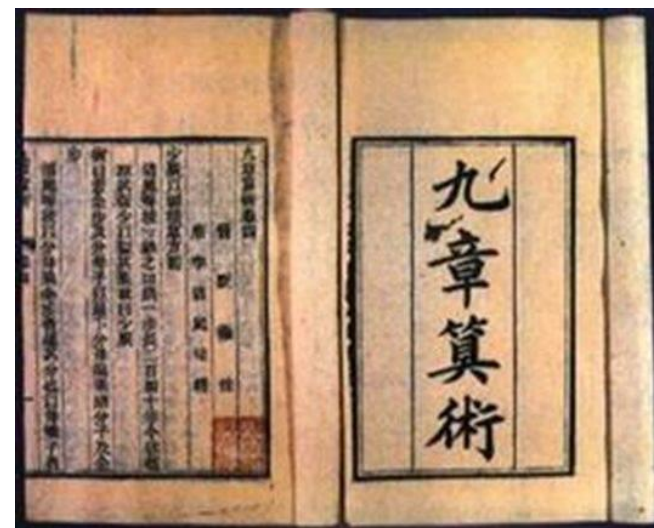
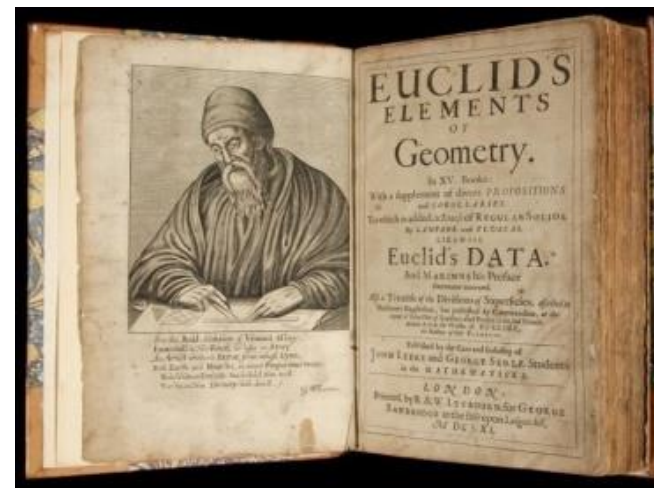


# 什么是算法

算法 (Algorithm) 一词最早出现于波斯数学家花拉子米(al-Khwarizmi)的著作中，他在书中使用“算法”来描述一种新的计算方法。

在数学史上，“欧几里得算法”被西方人认为是人类史上第一个算法。

算法在中国古代文献中称为“术”，自古以来就有许多专门论述“算法”的专著，最早出现在《九章算术》（公元一世纪左右）中。



# 算法的概念

计算机科学中的算法是指对问题求解方法准确而完整的描述，是为解决一个特定问题所采取的确定的有穷的运算序列。通俗地说，算法是对某一问题求解步骤的准确描述。

【例1.1】人类历史上第一个算法“欧几里得算法”。

**算法描述：**给定两个正整数 $a$ ， $b(a \geq b)$ ，求它们的最大公约数，即能够同时整除 $a$ 和 $b$ 的最大正整数。可表示为 $\text{gcd}(a,b)$ 。

**步骤1：**输入两个正整数 $a$ 和 $b$ ；

**步骤2：**计算 $a$ 除以 $b$ 的余数 $r$ ；

**步骤3：**若 $r$ 等于0，则最大公约数为 $b$ ，算法结束；若 $r$ 不等于0，则将 $b$ 值给 $a$ ， $r$ 值给 $b$ ，返回步骤2。

欧几里得算法（又称辗转相除法），古希腊数学家欧几里得在其著作《几何原本》（大约公元前300年）中最早描述了这种算法。

# 算法的基本特征

值得注意的是，并不是任意一个问题的求解步骤都可以称之为算法。计算机的算法还应该具备以下五个基本的特征：

- 有穷性：算法必须在合理的有限时间内，执行有限次步骤后结束。
- 确定性：算法中的每一个步骤都必须有确切的定义，不能有二义性。
- 输入项：一个算法有0个或多个输入，用以描述运算对象的初始情况，所谓0个输入是指算法本身定义了初始条件。
- 输出项：一个算法有一个或多个输出，是一组与“输入项”有确定关系的量值，是算法对输入数据加工后的结果。
- 可行性：算法中描述的操作可以通过已经实现的基本运算有限次地执行来完成。

# 算法的复杂度

时间复杂度：算法的时间复杂度是指执行算法所需要的计算工作量。一般来说，算法是问题规模 $n$ 的函数 $f(n)$ ，算法的时间复杂度也因此记做， $T(n)=O(f(n))$ 。算法执行的时间的增长率与 $f(n)$ 的增长率正相关，称作渐进时间复杂度。

空间复杂度：算法的空间复杂度是指算法需要消耗的内存空间。其计算和表示方法与时间复杂度类似，一般都用复杂度的渐近性来表示，记作 $S(n)=O(f(n))$ 。算法执行期间所需要的存储空间包括三个部分：算法程序所占的空间、输入的初始数据所占的存储空间、算法执行过程中所需要的额外空间。

# 算法的要素与表示

算法有两个基本要素：

一是算法中对数据的运算和操作

二是算法的控制结构。

算法的常用表示方法有：自然语言描述法、图形表示法、伪代码表示法等。

# 算法的要素与表示

运算和操作：

算术运算：加减乘除等运算

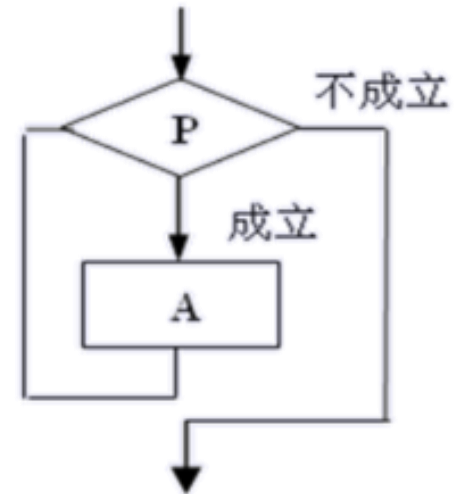
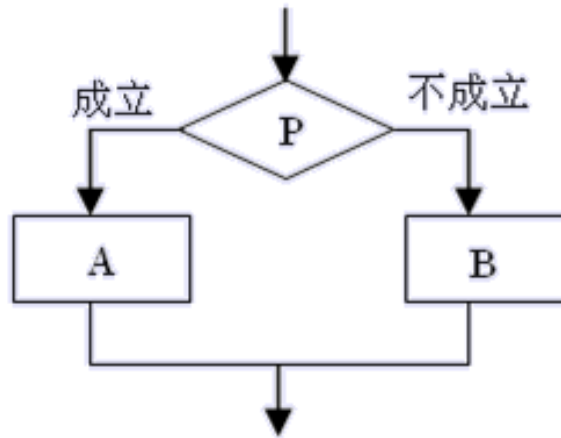
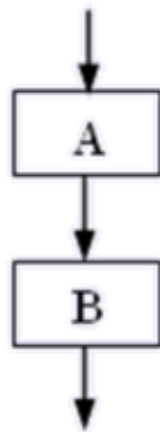
逻辑运算：与、或、非等运算

关系运算：大于、小于、等于、不等于等运算

数据传输：输入、输出、赋值等运算

# 算法的要素与表示

控制结构：



# 算法的要素与表示

算法的表示：

- 自然语言描述法
- 流程图表示法
- N-S图表示法
- 伪代码表示法

# 常用的算法策略

- 穷举法
- 递推法
- 递归法

# 常用的算法策略

- 穷举法

【例1.6】利用穷举法求解水仙花数。

定义如下：若一个n位自然数的各位数字的n次方之和等于它本身，则称这个自然数为阿姆斯特朗数。三位数中的阿姆斯特朗数被称为“水仙花数”，四位数的阿姆斯特朗数被称为“四叶玫瑰数”。如： $153 = 1^3 + 5^3 + 3^3$ ， $8208 = 8^4 + 2^4 + 0^4 + 8^4$ 。

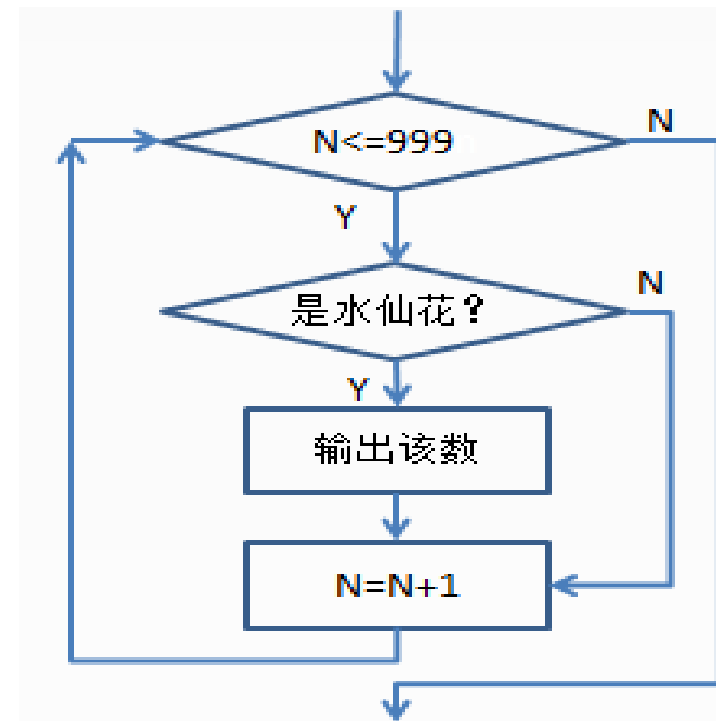
算法分析：

(1) 确定解的范围。水仙花数为符合条件的三位自然数N，N的范围为1~999。

(2) 确定解的条件。假设N的百位数字为i，十位数字为j，个位数字为k，则：

根据定义可知水仙花数的条件为： $N = i^3 + j^3 + k^3$

(3) 优化搜索范围，用循环结构实现。为提高效率可以缩小N的范围为100~999。



# 常用的算法策略

## • 递推法

【例1.7】递推法求自然数1~N的累加和。

假设 $i$ 为任意一个自然数, 且 $1 \leq i \leq N$ , 累加和为 $S$ ,  
 $S=1+2+3+4+\dots+N$ 。

算法分析:

(1) 求出问题规模为1的解; 当 $i=1$ 时,  $S=1$ 。

(2) 找出变化规律, 写出递推公式。求累加和的递推关系如下:

当 $i=1$ 时,  $S=1$ ,  $S_1=1$

当 $i=2$ 时,  $S=1+2$ ,  $S_2=S_1+2$

当 $i=3$ 时,  $S=1+2+3$ ,  $S_3=S_2+3$

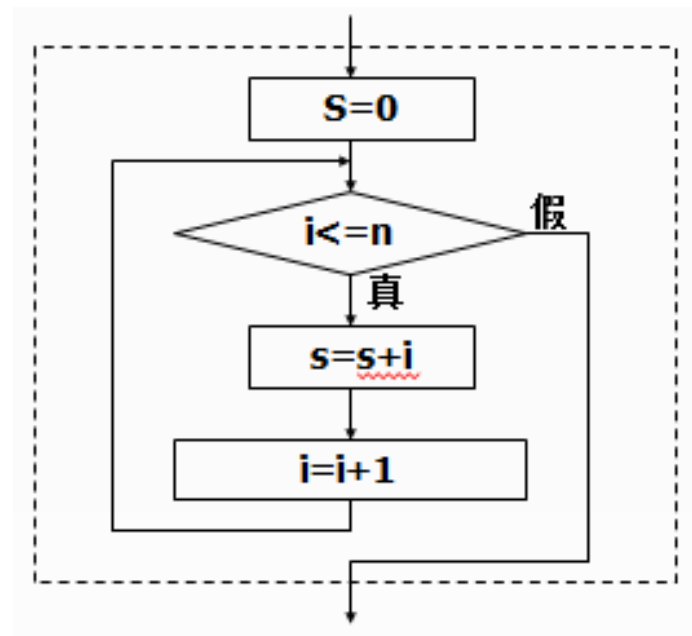
.....

当 $i=i$ 时,  $S=1+2+\dots+i$ ,  $S_i=S_{i-1}+i$

得到求累加和递推公式:

$$S_i = S_{i-1} + i$$

(3) 确定递推次数, 用循环结构实现。这里可以确定递推的次数为 $N$ 。



# 常用的算法策略

- 递归法

【例1.8】若将汉诺塔的64片圆盘全部移动完成，利用递归法求解共需移动多次？

这里不考虑程序实现的语言环境，给出汉诺塔的递归算法描述如下：

调用函数：

Call TowerofHanoi(64)

函数定义：TowerofHanoi(n)

If  $n = 1$  Then

Return 1

Else

Call TowerofHanoi( $n - 1$ ) \* 2 + 1

End If

## 递归算法-汉诺塔

圆盘个数	移动次数
1	1
2	3
3	$7 = 3 + 1 + 3$
4	$15 = 7 + 1 + 7$
...	...
n	$F(n) = 2 * F(n-1) + 1$

