

Distribution Info:

Distributor: B5-Software
Release Date: 7 May, 2025
Version: 0.0.1

自由的教輔
平等地学习

*For Freedom,
For Human Rights,
For Tomorrow.*

信
息
技
术
重
构



第一章 计算机基础

一、计算机组成

1. 硬件组成

计算机的五个基本组成部分是：

- a. **运算器**：负责执行算术运算（如加减乘除）和逻辑运算（如与、或、非）。它通常包含寄存器和算术逻辑单元（ALU），用于临时存储和处理数据。

--拓展 1：逻辑门

- b. **控制器**：负责从存储器中取出指令，解释指令，并协调计算机各部分的工作。控制器通常包括程序计数器（PC）、指令寄存器（IR）和控制单元（CU）。
- c. **存储器**：用于存储数据和程序。主要分为**主存储器**（RAM 内存、ROM 只读存储器）和**辅助存储器**（硬盘、固态硬盘等）。主存储器-内存存取速度快，但断电后数据会丢失，而辅助存储器用于长期存储数据。

--拓展 2：RAM、ROM、硬盘的区分

- d. **输入设备**：用于将外部信息输入计算机，如**键盘、鼠标、扫描仪**等。输入设备负责将用户指令和数据转换为计算机可以识别的格式。
- e. **输出设备**：用于将计算机处理后的结果输出给用户，如**显示器、打印机、扬声器**等。它们将计算机内部的二进制数据转换为人类可理解的信息。

2. 软件组成

软件是指使计算机硬件执行特定任务的**程序和相关文档**，主要分为两类：

- a. **系统软件**：管理和控制计算机硬件，支持应用软件运行的程序，包括**操作系统**（如 Windows、Linux、macOS、BSD、Unix、RTOS）和**系统工具**。
- b. **应用软件**：为满足用户特定需求而开发的程序，如**文字处理软件**、**浏览器**、**媒体播放器**等。

硬件和软件相互依赖，共同构成完整的计算机系统。硬件提供运行平台，软件指挥硬件完成各种任务，实现计算机的功能。

二、进制及其转换

1. 进制规则

- a. 先用阿拉伯数字（0~9）表示，不够用则引入字母（A~Z）；
- b. n 进制则有（0,1,2,...,n-1）的数字（0 也算一个数字）；
- c. 相同的值，n 进制表示，n 越大，位数越短；
- d. 不同的值，不同的进制表示，看上去可能一样；
- e. 对于 n 进制，描述时的 n 是使用十进制描述的。

2. 重要的进制法举例

进制	定义	英文名	英文名渊源
二进制 (Binary)	以 2 为基数，仅使用 0 和 1	Binary	源自拉丁语 <i>bini</i> ，意为“两者一组”
八进制 (Octal)	以 8 为基数，使用 0-7	Octal	源自拉丁语 <i>octo</i> ，意为“八”
十进制 (Decimal)	以 10 为基数，使用 0-9	Decimal	源自拉丁语 <i>decem</i> ，意为“十”
十六进制 (Hexadecimal)	以 16 为基数，使用 0-9 和 A-F	Hexadecimal	Hex- 源自希腊语 <i>hex</i> （六），-decimal 源自拉丁语 <i>decem</i> （十）

3. 进制转换

- a. **十进制转其他进制**：首先将待转换的十进制数除以目标进制的数，记录余数，然后用所得的商继续除以目标进制，直到商为 0。把所有余数倒序排列，得到的数就是对应进制的表示。例如，将十进制数 156 转换为 8 进制：156 除以 8 商 19 余 4，19 除以 8 商 2 余 3，2 除以 8 商 0 余 2，倒序排列余数就是 2 3 4，所以 156 的 8 进制表示为 234。
- b. **其他进制转十进制**：从右往左给每一位乘上目标进制的对应幂次（最右边的数位为该进制的 0 次幂，依次向左依次增大），然后把所有乘积相加得到的和就是十进制数。例如，将八进制数 234 转换为十进制：右边数字 4 乘 8^0 得 4，中间数字 3 乘 8^1 得 24，最左边数字 2 乘 8^2 得 128，将这些值相加 $128+24+4$ 得 156，所以八进制数 234 转换为十进制数为 156。
- c. **使用系统自带的计算器**：Windows 徽标键 + R 键，输入 “calc”，回车，点击 “查看”，选择 “程序员” 模式，选择待转换的进制，输入待转换的值，再选择目标进制，结果即为所求。

4. 进制转换技巧

- a. 待转换值的最右边的数位的转换结果与最终结果的最右边的数位一致；
- b. 二进制与八进制或十六进制之间转换时，可先将二进制分成 3 位（ 2^3 ，对于二进制/八进制）或 4 位一组（ 2^4 ，对于二进制/十六进制），再分别对应转换后合并。

三、汇编语言基础

1. 汇编语言基本概念

- a. 寄存器 (Registers): CPU 内部的小型存储单元, 如 AX、BX (8086);
- b. 指令 (Instructions): CPU 可执行的基本操作, 如 MOV (数据传输)、ADD (加法);
- c. 内存寻址: 汇编通过地址操作数据, 如 MOV AX, [1234H] 读取内存地址 0x1234 的数据;
- d. 中断: 计算机系统中用于处理异步事件的一种机制。当 CPU 在执行程序时, 某些事件 (如 I/O 设备请求、异常情况或外部信号) 可能需要 CPU 立即响应, 这时 CPU 会暂停当前任务, 转去执行相应的中断处理程序, 然后再恢复原任务。

2. 基本寄存器

寄 存 器	名称	作用	示例用途
AX	累加寄存器 (Accumulator)	主要用于算术运算、I/O 操作	MOV AX, BX (数据传输)
BX	基址寄存器 (Base)	存放内存地址或数据	MOV [BX], AL (访问内存)
CX	计数寄存器 (Counter)	主要用于循环计数	LOOP start (循环控制)
DX	数据寄存器 (Data)	I/O 端口访问、扩展乘法	OUT DX, AL (I/O 操作)
SP	堆栈指针 (Stack Pointer)	指向栈顶 (相对于 SS 段)	PUSH AX (压栈)
BP	基址指针 (Base Pointer)	访问栈中的局部变量	MOV AX, [BP-2]

3. 汇编语言基本指令

指令	语法	作用	示例
MOV	MOV 目标, 源	数据传输	MOV AX, BX (将 BX 的值复制到 AX)

ADD	ADD 目标, 源	加法	ADD AX, 5 (AX += 5)
SUB	SUB 目标, 源	减法	SUB BX, AX (BX -= AX)
MUL	MUL 源	无符号乘法	MUL CX (AX = AX * CX)
IMUL	IMUL 源	有符号乘法	IMUL DX (AX = AX * DX)
DIV	DIV 源	无符号除法	DIV CX (AX / CX, 余数存 DX)
IDIV	IDIV 源	有符号除法	IDIV DX (AX / DX, 余数存 DX)
INC	INC 目标	加 1	INC AX (AX += 1)
DEC	DEC 目标	减 1	DEC BX (BX -= 1)
CMP	CMP 目标, 源	比较	CMP AX, BX (AX - BX, 不改变寄存器值)
JMP	JMP 目标地址	无条件跳转	JMP start
JE	JE 目标地址	等于时跳转	JE loop (ZF=1 时跳转)
JNE	JNE 目标地址	不等时跳转	JNE retry (ZF=0 时跳转)
JG	JG 目标地址	大于时跳转 (有符号)	JG end
JL	JL 目标地址	小于时跳转 (有符号)	JL retry
AND	AND 目标, 源	按位与	AND AL, 0x0F (AL &= 0x0F)
OR	OR 目标, 源	按位或	OR BX, AX (BX
XOR	XOR 目标, 源	按位异或	XOR CX, CX (清零 CX)
NOT	NOT 目标	按位取反	NOT DX
SHL	SHL 目标, 位数	左移	SHL AX, 1 (AX <<= 1)
SHR	SHR 目标, 位数	右移	SHR BX, 2 (BX >>= 2)
PUSH	PUSH 目标	压栈	PUSH AX
POP	POP 目标	出栈	POP BX
CALL	CALL 地址	调用子程序	CALL my_func
RET	RET	返回	RET

INT	INT 中断号	调用中断	INT 0x21 (调用 DOS 21h 号中断)
SYSCALL	SYSCALL	调用 Linux 系统调用	SYSCALL

四、拓展

拓展 1. 逻辑门

逻辑门是计算机电路里最基本的“开关”，它们根据输入信号决定输出结果，就像做数学题一样。下面是 12 种常见的逻辑门：

逻辑门	符号	功能	学生能理解的例子
与门 (AND)	$\wedge$	只有所有输入都是 1，输出才是 1	只有 你完成作业 并且 老师批准，才能出去玩
或门 (OR)	$\vee$	只要有一个输入是 1，输出就是 1	你可以去春游，只要 交了钱 或者 被选中
非门 (NOT)	$\neg$	取反，1 变 0，0 变 1	老师说“没交作业不能下课”，NOT 让“没交”变成“交了”
与非门 (NAND)	$\uparrow$	与门的反向，只有全 1 时输出 0，否则 1	你和朋友都作弊，才会被扣分（但 NAND 反转成“不给分”）
或非门 (NOR)	$\downarrow$	或门的反向，只有全 0 才输出 1	老师规定“只要一个人说话，全班不能下课”
异或门 (XOR)	$\oplus$	输入不同时输出 1，相同时输出 0	你和朋友不能点一样的菜，否则就没得吃
同或门 (XNOR)	$\odot$	输入相同时输出 1，不同时输出 0	你和朋友答案一样，才能得分
缓冲器 (BUFFER)	=	输出和输入相同，但增强信号	老师用麦克风讲课，声音变大但内容不变
三态门 (Tri-state)	Z	允许或阻断信号（高阻态）	老师决定谁可以发言，谁要保持安静

施密特触发器（Schmitt Trigger）	-	过滤噪音，保持信号稳定	班上噪声很小，老师不会管，但超过一定程度就会喊“安静！”
传输门（Transmission Gate）	开关	选择信号是否通过	食堂只有你饭卡有钱，才会允许你刷卡吃饭
门控反相器（Controlled Inverter）	受控 NOT	只有控制信号允许，才会取反	作业只有老师批准后，才能被改成对的答案

拓展 2. RAM、ROM、硬盘的区分

属性	RAM（随机存取存储器）	ROM（只读存储器）	硬盘（HDD/SSD）
定义	计算机的 运行内存 ，用于 临时存储数据	预先写入数据的存储器，主要用于 存储固件	计算机的 长期存储设备 ，用于 存储操作系统、应用程序和用户数据
读写方式	可读可写，速度快	只能读取 ，某些类型支持写入	可读可写 ，速度比 RAM 慢但比 ROM 快（SSD > HDD）
数据持久性	断电后数据会丢失	断电后数据仍然保留	断电后数据仍然保留
用途	运行程序、缓存数据，提高计算机响应速度	存储 BIOS/UEFI 固件、嵌入式系统程序	存储操作系统、软件、文件等长期数据
典型容量	4GB - 128GB	几 MB - 几 GB EEPROM 可能几 KB	256GB - 数 TB
示例	DDR4、DDR5 内存	BIOS 芯片、微控制器固件	HDD（机械硬盘）、SSD（固态硬盘）

第二章 数字化

一、音频数字化

1. PCM 编码（最常用）

- a. 定义：PCM 编码就是把听到的连续的声音（比如说话、音乐），按照固定的时间间隔进行采样，也就是一小段一小段地“截取”，然后把每一小段声音的音量（振幅）用数字的方式表示出来。这样电脑就能用一串数字来记录原来的声音，之后再把这些数字还原回声音，让我们重新听到几乎一样的效果。

b. 分步：

步骤	名称	做了什么？	举个例子
一	采样	每隔一点时间“听”一次声音，就像不断拍声音的快照	每秒采样 44,100 次（CD 音质）
二	量化	把每次采到的声音大小（振幅）用数字表示出来	声音大小变成 0~65535 之间的数字（16 位）
三	编码	把所有的数字一个个排好，组成一串数据存进电脑或音频文件	得到一段数字音频，能还原成声音

c. 文件大小计算（未压缩）：

参数名称	含义说明	示例值
采样率	每秒采集多少次声音数据	44,100 Hz（CD 音质）
位深度	每次采样使用多少位（二进制位）表示	16 位
声道数	声音通道数量（单声道或立体声）	2（立体声）
时长	音频的总时长（单位：秒）	10 秒
公式	文件大小 = 采样率 × 位深度 × 声道数 × 时间 ÷ 8	—
计算过程	$44100 \times 16 \times 2 \times 10 \div 8$	1,764,000 字节
转换为 MB	字节数 ÷ 1024 ÷ 1024	≈ 1.68 MB

d. **压缩:**

项目	PCM 编码 (原始)	无损压缩音频	有损压缩音频
是否压缩	不压缩	压缩 (不丢失信息)	压缩 (丢失部分信息)
音质	非常高 (原始音质)	与原始一样	可察觉变化 (压缩后音质)
文件大小	最大	中等	最小
是否支持还原	是, 直接播放或编辑	是, 可 100%还原原始数据	否, 信息永久丢失
常见格式	WAV、AIFF	FLAC、ALAC	MP3、AAC、OGG
应用场景	音乐制作、广播、存档	高音质音乐存储、备份	流媒体、网络传输、手机音乐
举个例子	专业录音室录下的声音	无损音乐播放器里的歌曲	音乐软件较低音质的音频

- **PCM 编码**是不压缩的原始声音, 音质最好但文件大;

- **无损压缩**压缩后还原不丢音质;

- **有损压缩**更省空间, 但会牺牲一点音质。

2. DSD 编码

a. **定义:** DSD 编码是一种用极高频率的 **1 位数字信号**来记录声音变化的音频编码方式, 声音还原度很高, 是一种高保真音频格式。

b. **DSD 与 PCM 的区别:**

DSD 和 PCM 不一样, 它不是一次记录“声音有多大”, 而是用**非常快的速度** (比如每秒 2,822,400 次) 记录“声音是变大了还是变小了”。

- 每次采样只用 **1 位 (0 或 1)** 来表示声音的趋势。

- 类似“跟踪声音波动的方向”而不是“测量具体数值”。

- 虽然每次只有 1 位, 但因为采样非常密集, 最后还原的声音也非常细腻。

项目	PCM 编码	DSD 编码
位深度	多位（如 16 位、24 位）	1 位
采样率	一般为 44.1kHz~192kHz	超高，比如 2.8224MHz（DSD64）
表示方式	表示“当前声音有多大”	表示“声音是在变大还是变小”
应用	普通音频、CD、音乐制作	SACD（超级音频 CD）、高端音响
文件大小	相对较小（有压缩时）	较大（虽然是 1 位，但采样极密）
音质风格	清晰、细节多	柔和、自然、类模拟味道

c. 文件大小计算（未压缩）：

项目	说明	示例值（以 DSD64 为例）
采样率	每秒钟采样次数	2,822,400 Hz（DSD64）
位深度	每次采样使用的位数	1 位
声道数	声音通道数量（单声道=1，立体声=2）	2（立体声）
音频时长	声音的总时长	60 秒
公式	文件大小（字节） = 采样率 × 声道数 × 时间 ÷ 8	= 2,822,400 × 2 × 60 ÷ 8
计算结果	实际字节数	42,336,000 字节 ≈ 40.38 MB

常见 DSD 编码格式文件大小对比表（立体声、1 分钟音频）：

编码格式	采样率（Hz）	每分钟文件大小（字节）	约合文件大小（MB）
DSD64	2,822,400	42,336,000	≈ 40.38 MB
DSD128	5,644,800	84,672,000	≈ 80.77 MB
DSD256	11,289,600	169,344,000	≈ 161.53 MB
DSD512	22,579,200	338,688,000	≈ 323.06 MB

d. 压缩：

压缩类型	说明	示例格式	特点	文件大小
不压缩	完全没有任何压缩，保存原始的 DSD 数据	DSD Raw 或 DSF（无压缩）	音质保持原样，但文件非常大，不做任何数据丢失	文件非常大，接近原始大小
无损压缩	压缩后可以 100% 恢复原始音质	DST（Direct Stream Transfer）	压缩后音质不丢失，适用于高保真音频	文件大小减少约 30%-50%
有损压缩	压缩后丢失部分音质，牺牲一部分细节	DSF（降低了采样率）	压缩比高，适合流媒体和存储，但音质会有所牺牲	文件大小大幅减少（可高达 90%）

二、图像数字化

1. 位图

a. 定义: 位图(Bitmap)是一种用于存储图像数据的文件格式或数据结构。在位图中, 图像被表示为一个像素网格, 每个像素有特定的颜色值。通常, 位图是通过以二维矩阵的方式排列像素来表示图像的, 每个像素对应一个数据单元。

- 位图的分辨率由图像的像素数量决定, 分辨率越高, 图像越清晰。
- 由于每个像素都需要存储信息, 位图文件通常会比较大, 特别是在高分辨率或颜色丰富的情况下。
- 当你缩放位图图像时, 图像会失去清晰度, 可能会出现锯齿状的边缘或模糊现象。
- 常见的位图格式包括 BMP、PNG、JPEG、GIF 等。

b. 文件大小计算 (未压缩):

项目	含义	示例值
图像宽度	图像水平像素的数量	1000 像素
图像高度	图像垂直像素的数量	800 像素
像素总数	图像的总像素数量 (宽度 × 高度)	$1000 \times 800 = 800,000$ 个像素
颜色深度	每个像素所占用的位数 (bit), 表示颜色信息的丰富程度	24 位
每像素字节数	每个像素占用的字节数, 计算方式为颜色深度除以 8	$24 / 8 = 3$ 字节
文件大小 (字节)	图像的文件大小, 计算方式为像素总数乘以每像素的字节数	$800,000 \times 3 = 2,400,000$ 字节

色彩表达方式:

表达方式	定义描述	示例数值	说明
个 (种) 色	表示图像可显示的颜色数量, 即总的颜色个数,	256 个 (种) 色、65,536 个 (种) 色、16,777,216 个 (种)	数值通常为 2 的 n 次方, 例如 8 位色对应 $2^8=256$

	用“个”作为单位。	色	个色。
位色	指图像的色深,即每个像素用来表示颜色信息的位数,用“位色”描述。	8 位色、16 位色、24 位色、32 位色	例如 8 位色表示每个像素用 8 位存储颜色,通常对应 256 个色。
色位	同“个(种)色”,也用于描述像素颜色的种数。	8 色位、16 色位、256 色位	注意:这种表示方法没有被广泛使用,说法不一。

c. 压缩:

属性	不压缩 位图	无损压缩	有损压缩	说明
文件大小	最大	较小	最小	不压缩存储所有像素数据;无损压缩在不丢失信息的前提下减小体积;有损压缩则通过舍弃部分细节达到更高压缩率。
图像质量	保持原始质量,无任何损失	完全保留原始图像质量	存在不同程度的质量损失	无损压缩不影响图像质量,有损压缩在高压比下可能出现伪影和细节损失。
处理速度	数据量大,读写速度较慢	需要解压缩,但通常较快	解压和压缩处理较复杂,部分场景下较慢	无损和有损压缩都需要额外的压缩/解压过程,但有损压缩算法通常针对传输优化。
编辑灵活性	直接编辑,无重复编码问题	编辑时需解压为原始数据,后续可恢复	多次编辑和重压缩会累计损失图像质量	不压缩和无损压缩适合频繁编辑,有损压缩不适合反复处理。
存储与传输	占用大量空间,传输带宽要求高	存储和传输需求适中	占用最少存储空间,传输更高效	在网络应用和移动设备中,有损压缩常用以降低传输延时,无损压缩则用于对质量要求较高的场合。
应用场景	专业图像处理、医学影像、扫描件等	数据归档、无损要求的数字摄影和设计	数字摄影、网页图片、社交媒体、视频流媒体等	根据应用场景选择合适的存储方式:高质量需求时选用不压缩或无损压缩;对传输和存储要求较高时可采用有损压缩。

2. 矢量图

a. 定义：矢量图（Vector）是一种利用数学公式描述图形和图像的表现方式，其主要特点是用点、线、曲线和多边形等基本几何元素来构造图形，而不是像位图那样记录每一个像素的信息。这种表示方法的主要优点包括：

- 无损缩放：由于图形由数学表达式定义，放大或缩小时不会失去清晰度或出现锯齿现象。
- 较小文件体积：在图形内容较为简单的情况下，矢量图的文件通常比同尺寸的位图要小，因为它只存储形状和颜色的描述信息。
- 便于编辑：矢量图中的各个对象可以独立修改、调整形状、颜色、位置等，适合需要频繁修改和调整的设计工作。
- 矢量图常用于标志设计、图标、插图、技术绘图和排版等场合。常见的矢量图格式有 SVG、EPS、AI 和 PDF 等。

b. 文件大小计算（未压缩）：

矢量图没有文件大小的计算公式。

图形类型	描述	典型文件大小
简单 SVG（单矩形）	一个矩形，无复杂属性	约 100 字节
中等复杂 SVG（多形状）	包含多个矩形和圆，少量路径	约 1-10 KB
复杂 SVG（多路径）	包含详细路径和文本，如技术图	数十 KB 至数百 KB
高度详细矢量图	复杂插图或动画，包含大量元素	可能达 1 MB 以上

c. 压缩:

属性	不压缩	无损压缩矢量图	有损压缩矢量图	说明
文件大小	最大, 保存完整的矢量图形数据	较小, 去除冗余数据	最小, 通过丢弃部分信息进一步减小文件大小	不压缩保存完整图形, 无损压缩通过去冗余数据减小体积, 有损压缩通过丢弃细节减少体积。
图像质量	完全保留原始图像质量, 无任何损失	保持原始图像质量	存在质量损失, 特别是在高压缩比时	不压缩保持最佳质量, 无损压缩没有质量损失, 有损压缩会导致图形的细节丢失。
处理速度	读写速度较慢, 文件较大	读写速度快, 但需要解压	读写速度最快, 尤其是对于压缩较小的文件	压缩文件通常处理速度较快, 特别是有损压缩, 适合高效传输和加载。
存储与传输	存储空间大, 传输带宽需求高	节省存储空间, 传输效率更高	文件非常小, 传输和存储成本最低	无损压缩适合高质量但空间有限的需求, 有损压缩适合大规模传输, 节省带宽和存储。
编辑灵活性	便于直接编辑, 无需解压	需要解压或逐步解压编辑内容	多次编辑可能影响质量, 特别是有损压缩	不压缩和无损压缩都适合频繁编辑, 有损压缩不适合多次编辑, 会导致质量损失。
适用场景	专业设计、高质量存档、印刷等	网络应用、图形设计、文档归档、出版等	数字广告、网站图标、移动设备应用等	不压缩适用于高精度设计, 无损压缩适用于需要高质量同时节省空间的场景, 有损压缩适用于大规模传输。

3. 位图、矢量图的对比

属性	位图 (Bitmap)	矢量图 (Vector)	说明
基本定义	由像素组成, 每个像素存储颜色值	通过数学公式描述图形, 使用点、线、曲线、形状等元素	位图依赖于像素, 矢量图依赖于几何元素。
文件大小	文件较大, 特别是在高分辨率时	文件较小, 特别是在图形内容简单时	位图需要存储所有像素信息, 矢量图只存储几何描述。
图像质量	放大时会出现像素化、模糊或锯齿现象	放大或缩小时不会失真, 保持清晰度	位图的质量随分辨率变化, 矢量图无论如何缩放都保持高

			质量。
编辑灵活性	编辑时修改像素，细节丢失可能较大，特别是放大时	易于编辑，可以独立修改每个对象，如路径、形状、颜色等	位图编辑较为复杂，矢量图编辑灵活，且不会损失质量。
存储与传输	存储空间需求高，尤其是高分辨率图像，传输带宽较高	存储空间小，传输效率较高	位图文件较大，传输时需要更多带宽；矢量图文件较小，适合传输。
图像表现	适合表现复杂的细节，适合照片、艺术图像等	适合表现图标、标志、插图、简化图形等	位图表现更细腻，矢量图适合简洁、图形化的表现。
常见格式	JPEG、PNG、BMP、GIF、TIFF 等	SVG、EPS、PDF、AI 等	位图常用于照片和复杂图像，矢量图用于图标、插图和排版设计等。
适用场景	数码摄影、网页设计、打印输出等高精度图像要求的场景	标志设计、图标、插图、技术图纸、UI 设计等	位图适合照片和复杂的图像，而矢量图适合需要清晰和缩放的设计。

三、视频数字化

1. 文件大小计算（未压缩）

项目名称	含义说明	示例值（以 1920×1080, 24fps, 24 位色, 10 秒为例）
分辨率(Resolution)	每帧图像的像素宽度 × 高度	$1920 \times 1080 = 2,073,600$ 像素/帧
色深 (Color Depth)	每个像素占用的位数，常见为 24 位（8 位 RGB 各占 8 位）	24 位 = 3 字节（每像素）
每 帧 大 小 (Frame Size)	每帧所占的总字节数 = 分辨率 × 每像素字节数	$2,073,600 \times 3 = 6,220,800$ 字节 ≈ 5.93 MB
帧率 (Frame Rate)	每秒钟的帧数 (FPS)	24 帧/秒
每秒大小	每秒视频数据大小 = 每帧大小 × 帧率	$5.93 \text{ MB} \times 24 = 142.32 \text{ MB/秒}$

视频时长 (Duration)	视频总时长, 以秒计	10 秒
总文件大小	视频总大小 = 每秒大小 × 视频时长	$142.32 \text{ MB} \times 10 = 1,423.2 \text{ MB}$ $\approx 1.39 \text{ GB}$

2. 压缩

a. 未压缩 vs 压缩视频文件大小计算方法对比表

项目	未压缩视频计算方法	压缩视频估算方法	说明
原理	逐帧逐像素记录所有颜色信息	利用视频编码器压缩冗余数据(时间冗余、空间冗余、颜色冗余)	压缩算法通过预测和参考帧等技术极大减少数据量
基本公式	分辨率 × 色深 ÷ 8 × 帧率 × 时长 (秒)	码率 (bitrate) × 时长 ÷ 8 (换算成字节)	未压缩精确计算, 压缩视频以码率为估算依据
分辨率	必须考虑, 影响帧大小	影响编码器压缩效率, 影响最终码率	高分辨率 → 高数据量 → 高码率
色深	直接影响每像素占用的字节数	通常已包含在码率中间接体现	高色深时压缩效率略低, 文件略大
帧率 (FPS)	直接乘以每帧大小	影响压缩效率, 更多帧 = 更多数据	高帧率视频需要更高的码率保持同样清晰度
时长	线性增长	线性增长	视频越长, 文件越大
关键参数: 码率 (Bitrate)	N/A (未压缩)	是计算核心: 如 4 Mbps = 4,000,000 bps = 500,000 B/s	常见码率: 720p (1-5 Mbps)、1080p (3-8 Mbps)、4K (15-50 Mbps) 等
示例计算	$1920 \times 1080 \times 24 \text{ bit} \div 8 \times 24 \text{ fps} \times 10 \text{ s}$ $\approx 1.39 \text{ GB}$	$4 \text{ Mbps} \times 10 \text{ s} \div 8$ $= 5 \text{ MB}$	同样视频未压缩需 1.39GB, 压缩后只需 5MB (假设 4 Mbps)

压缩率	无压缩	可达 100:1 或更高	依编码器和视频内容而异, 运动越复杂压缩率越低
适用场景	专业制作、无损编辑、母版存档等	网络发布、移动播放、流媒体、存储优化	未压缩质量极高但文件大, 压缩适合大多数实际应用

b. 压缩视频大小估算公式:

$$\text{文件大小 (字节)} = \text{码率 (bps)} \times \text{视频时长 (秒)} \div 8$$

示例对比 (1080p, 10 秒):

类型	示例参数	估算文件大小
未压缩	1920×1080, 24 位色, 24fps, 10 秒	≈ 1.39 GB
压缩 (中等)	码率 4 Mbps	$4,000,000 \times 10 \div 8 = 5 \text{ MB}$
压缩 (高质)	码率 8 Mbps	$8,000,000 \times 10 \div 8 = 10 \text{ MB}$

c. 视频压缩编码格式对比表:

编码格式	全称	压缩效率	文件大小	解码兼容性	画质表现	开源	适用场景
H.264 (AVC)	Advanced Video Coding	中等	中等	极高 (几乎所有设备)	优秀	否	YouTube、直播、摄像头、智能手机等
H.265 (HEVC)	High Efficiency Video Coding	高	更小	较低 (新设备支持)	更好	否	4K/8K 视频、高质量流媒体、蓝光
VP8	-	中	中等偏小	中等	尚可	是	网络视频、WebRTC、旧版 WebM
VP9	-	高	更小	好	优秀	是	YouTube

				(Chrome/ Android)			高清/4K、 WebM 格式
AV1	AOMedia Video 1	极高	最小	中等（新硬 件支持）	优秀至极高	是	新一代网络 流媒体、 YouTube 8K、 Netflix、 未来网页视 频标准
MPEG-2	Moving Picture Experts Group 2	低	最大	高	普通	否	DVD、老式 电视广播
ProRes	Apple ProRes	几乎无压 缩	非常大 （母版 级）	低（苹果生 态为主）	极高	否	专业影视后 期、素材存 储、非线性 编辑
CineForm	-	轻压缩	较大	中等	极高	是	专业剪辑、 GoPro 制 作、后期处 理
Theora	-	中	中	较低	一般	是	自由软件项 目、老式开 源网页视频

四、拓展

拓展 1. 令人类感到舒适的媒体分辨率

类型	常见分辨率/参数	单位	感知体验	适用场景
音频	44.1 kHz / 16 bit	采样率 / 量 化位数	CD 音质，适合大多数人	音乐、普通语音、电影
	48 kHz / 24 bit		高清音频	电影音轨、专业录音

	≥96 kHz / 24 bit		极高保真，人耳难明显区分	专业音频制作、高保真爱好者
图像	72-100 PPI（显示）	像素每英寸	屏幕常用分辨率，正常浏览无压力	网页、显示器查看图像
	300 DPI（印刷）	点每英寸	印刷品质清晰，适合近距离阅读	照片、杂志、书籍
	1080×1080（社交媒体头像）	像素	头像清晰	社交媒体的头像
视频	720p（1280×720）	像素	基础高清，移动设备观看清晰	手机视频、老电视
	1080p（1920×1080）		全高清，大多数场合最舒适	家用电视、流媒体标准
	4K（3840×2160）		超高清，近距离或大屏幕体验更佳	影院、大尺寸电视、游戏
	8K（7680×4320）		超高分辨率，人眼难完全分辨	专业制作、未来应用

拓展 2．注意题目要求的最终单位

各种存储单位的英文全称及其与 **bit** 的转换倍率列表，分为**十进制（基于 1000）**和**二进制（基于 1024）**，此表格数据均以十进制表示：

缩写	英文全称	进制类型	换算为 bit 的倍率
bit	bit	-	1
Byte	Byte	-	8 bits
KB	Kilobyte	十进制	1 KB = 1,000 Bytes = 8,000 bits
KiB	Kibibyte	二进制	1 KiB = 1,024 Bytes = 8,192 bits
MB	Megabyte	十进制	1 MB = 1,000,000 Bytes = 8,000,000 bits
MiB	Mebibyte	二进制	1 MiB = 1,048,576 Bytes = 8,388,608 bits
GB	Gigabyte	十进制	1 GB = 1,000,000,000 Bytes = 8,000,000,000 bits
GiB	Gibibyte	二进制	1 GiB = 1,073,741,824 Bytes = 8,589,934,592 bits
TB	Terabyte	十进制	1 TB = 1,000,000,000,000 Bytes = 8,000,000,000,000 bits

TiB	Tebibyte	二进制	1 TiB = 1,099,511,627,776 Bytes = 8,796,093,022,208 bits
-----	----------	-----	--

注意：

- Byte = 8 bits 是基本换算。
- 带 i (如 KiB, MiB) 的单位是 二进制 (IEC 标准)。
- 不带 i 的 (如 KB, MB) 是 十进制 (SI 标准)。
- 考试时若无特殊说明，会混用*B和*iB，此时按 1024 (二进制标准) 计算。

拓展 3. 声道个数

声道类型	格式名称	组成说明	低 频 通 道 (LFE)	实际总声道数 (含 LFE)
Mono	1.0	单声道	0	1
Stereo	2.0	左、右声道	0	2
Stereo + LFE	2.1	左、右 + 低频	1	3
Surround	3.0	左、右 + 中央	0	3
	3.1	左、右 + 中央 + 低频	1	4
Quadraphonic	4.0	左、右 + 左后、右后	0	4
	4.1	同上 + 低频	1	5
5.0	5.0	左、右 + 中央 + 左环绕、右环绕	0	5
5.1	Dolby Digital, DTS 5.1	左、右 + 中央 + 左/右环绕 + LFE	1	6
6.1	DTS-ES, Dolby Digital EX	5.1 + 后中央声道	1	7

7.0	7.0	左、右 + 中央 + 左/ 右环绕 + 左/右后	0	7
7.1	Dolby TrueHD, DTS-HD MA	5.1 + 左/右后	1	8
9.1	扩展环绕音效	7.1 + 前高左/右或前 宽声道	1	10
11.1	Dolby Atmos, Auro 11.1	7.1 + 前宽 + 前高 + 顶部或额外中置	1	12
22.2	NHK Super Hi- Vision	超高清多声道系统(3 层 布局)	2	24

第三章 Python 基础

一、 名词概念

1. Python 语言

a. 定义：Python 是一种**解释型**语言，代码无需编译，直接由解释器逐行执行，便于调试和开发。它语法简洁、可读性强，广泛应用于数据分析、人工智能、Web 开发等领域，适合快速开发与原型设计。

b. 模块：

模块类型	描述	示例模块	特点说明
内置模块	Python 自带,无需安装或导入文件即可使用	math、sys、time	加载速度快,提供基础功能
标准库模块	Python 安装时附带的模块集合	datetime、json、os	安装 Python 时自动包含,覆盖面广
第三方模块	由社区开发,需要通过 pip 安装	numpy、requests、pandas	功能丰富,用于数据分析、网络请求、科学计算等
自定义模块	用户自行创建的 .py 文件	my_module.py (自定义)	可复用代码,适合项目模块化开发
内置扩展模块	使用 C/C++ 编写并编译的高性能模块	re (部 分) 、unicodedata	提供更高执行效率,通常作为内置模块存在
包 (Package)	含 __init__.py 文件的模块集合	email、urllib	支持模块分层组织,便于管理大型项目

c. 作用域（范围）：

作用域类型	描述	示例	生效范围
局部作用域 (Local)	函数内部定义的变量	<code>def func(): x = 5</code>	仅在函数或代码块内部可访问
嵌套作用域 (Enclosing)	嵌套函数中, 内函数可访问外函数的变量	<code>def outer(): x = 5; def inner(): print(x)</code>	外层函数中定义, 供内层函数访问
全局作用域 (Global)	模块层级定义的变量, 供整个模块使用	<code>x = 10</code>	整个模块 (文件) 内有效
内置作用域 (Built-in)	Python 解释器预定义的变量和函数	<code>len()</code> 、 <code>range()</code>	所有 Python 脚本全局可用

d. 数据类型 (变量类型):

数据类型	类型名称 (类名)	示例值	可变性	描述与用途
整数 (整数型)	<code>int</code>	<code>42, -7, 0</code>	不可变	表示整数值, 用于计数、索引等
浮点数	<code>float</code>	<code>3.14, -0.001</code>	不可变	表示小数或实数, 支持数学计算
字符串	<code>str</code>	<code>'hello', "world"</code>	不可变	表示文本, 支持多种操作如拼接、查找
布尔值	<code>bool</code>	<code>True, False</code>	不可变	表示逻辑值, 常用于条件判断
列表	<code>list</code>	<code>[1, 2, 3], ['a', 'b']</code>	可变	有序集合, 支持修改、添加、删除元素
元组	<code>tuple</code>	<code>(1, 2), ('a', 'b')</code>	不可变	有序集合, 元素不可修改, 适合不可变数据结构
集合	<code>set</code>	<code>{1, 2, 3}</code>	可变	无序唯一元素集合, 适合去重和集合运算
字典	<code>dict</code>	<code>{'a': 1, 'b': 2}</code>	可变	键值对集合, 常用于存储映射关系
空值	<code>NoneType</code>	<code>None</code>	不适用	表示空或无值

- 数据类型的规范严格，不同的数据类型不能混用参与运算；
- 全局作用域最简单，考试中最常见；
- Python 模块相当于其他编程语言的（动态）库。

2. 程序结构类型

结构类型	描述	常见关键词或语法	示例用途
顺序结构	按照代码编写顺序逐行执行	无特殊关键词	普通语句执行
分支结构	根据条件判断执行不同路径	if / elif / else	条件判断、逻辑分流
循环结构	重复执行某段代码	for 、 while 、 break 、 continue	遍历列表、重复计算等
函数结构	将代码封装为可复用模块	def、return	封装逻辑，复用功能
异常处理结构	捕获并处理运行时错误	try、except、finally	防止程序因错误中断
类与对象结构	面向对象编程的基本结构	class、self、__init__	模型构建、封装数据和方法
模块结构	将功能拆分为独立文件	import、from	提高代码组织性与复用性
包结构	管理多个模块的目录结构	__init__.py	管理大型项目模块
生成器结构	延迟生成数据，提高性能	yield	流式数据处理、大数据遍历
装饰器结构	修改函数/类行为的函数结构	@decorator	日志、权限控制、缓存等
并发结构	同时运行多个任务	threading、asyncio	提高程序执行效率（如 I/O 密集）

a. 顺序结构:

```
print("开始程序")

x = 5

y = 10

z = x + y

print("结果是: ", z)

print("程序结束")
```

解释:

顺序结构是指程序代码按照从上到下、逐行执行的顺序运行;

每一条语句都会依次被执行, 不涉及条件判断或循环;

上面的例子中:

首先打印 "开始程序";

然后依次执行变量赋值 $x = 5$ 和 $y = 10$;

接着执行加法 $z = x + y$;

最后打印结果和结束信息。

b. 分支结构:

```
score = 85

if score >= 90:

    print("成绩优秀")

elif score >= 60:

    print("成绩及格")

else:

    print("成绩不及格")
```

解释:

分支结构用于根据不同条件执行不同的代码路径；

Python 使用 `if`、`elif`（可选）、`else`（可选）来表示条件判断；

在这个例子中：

变量 `score` 设为 85；

程序先判断 `score >= 90`，不成立；

再判断 `score >= 60`，成立，于是执行对应的 `print("成绩及格")`；

如果以上都不满足，才会进入 `else` 分支（在本例中不会）。

c. 循环结构：

示例代码 (for 循环)：

```
for i in range(5):  
    print("第", i + 1, "次问好")
```

示例代码 (while 循环)：

```
count = 0  
  
while count < 5:  
    print("计数: ", count)  
    count += 1
```

解释：

for 循环：

- `range(5)` 生成从 0 到 4 的整数；
- `for i in range(5)` 表示把每个数字依次赋值给 `i`，然后执行循环体；
- 适合用于**已知循环次数**的场景。

while 循环：

- 在条件 `count < 5` 为真时不断重复执行循环体；
- 每次循环中 `count` 自增；
- 适合用于**未知循环次数、依赖条件控制结束**的情况。

补充：

- break: 可用于提前跳出循环；
- continue: 跳过当前循环，进入下一次循环。

二、简单语句

1. 语法

语法元素	描述	示例	说明与用途
变量命名	变量名由字母、数字、下划线组成,且不能以数字开头	<code>x = 10</code>	变量名称应具有描述性,避免与 Python 保留字冲突。
注释	用于解释代码,不被执行	<code># 这是一个注释</code>	单行注释以 # 开头。多行注释使用三引号: <code>'''注释'''</code> 。
缩进	Python 使用缩进表示代码块,而不是大括号 {}	<code>if x > 0: print("正数")</code>	缩进严格控制代码块结构,通常为 4 个空格。
运算符	用于进行计算、比较、逻辑操作等	<code>+, -, *, /, ==, and, or</code>	常用运算符包括算数运算符、比较运算符、逻辑运算符等。
字符串	可以使用单引号、双引号或三引号表示字符串	<code>'hello', "world", '''多行字符串'''</code>	字符串支持拼接、切片、格式化等操作。
列表	用于存储多个值的有序集合	<code>[1, 2, 3], ['a', 'b', 'c']</code>	支持索引、切片、修改、添加和删除元素。
元组	用于存储多个值的不可变集合	<code>(1, 2, 3), ('a', 'b')</code>	一旦定义,元素不可修改。
字典	存储键值对的无序集合	<code>{'name': 'Alice', 'age': 25}</code>	支持通过键快速查找、修改、删除值。
条件表达式	根据条件选择值	<code>x if condition else y</code>	简写版的 if-else。

集合	存储唯一元素的无序集合	{1, 2, 3}, {}	支持集合运算如并集、交集、补集等。
函数定义	使用 def 定义函数	def add(x, y): return x + y	通过 def 关键词定义函数, 支持参数和返回值。
异常处理	捕获和处理程序中的错误	try: x = 1 / 0; except ZeroDivisionError:	通过 try、except 语句处理错误, finally 执行清理操作。
列表推导式	从现有列表创建新列表	[x * 2 for x in range(5)]	简洁的列表生成方式, 可以加入条件过滤。
lambda 表达式	定义匿名函数	f = lambda x: x + 2	使用 lambda 创建小型函数, 常用于简单的回调或传递。

2. 运算、字符串拼接

运算符	描述	示例	结果
加法	对两个数值进行相加	a + b	3 + 2 = 5
减法	对两个数值进行相减	a - b	5 - 2 = 3
乘法	对两个数值进行相乘	a * b	3 * 2 = 6
除法	对两个数值进行相除 (返回浮点数)	a / b	7 / 2 = 3.5
整数除法	对两个数值进行相除 (返回整数)	a // b	7 // 2 = 3
取余	计算两个数值相除后的余数	a % b	7 % 2 = 1
指数	对两个数值进行指数运算	a ** b	3 ** 2 = 9
比较运算	比较两个数值的大小	a == b, a != b, a < b	3 == 3(True), 3 != 2(True)

字符串拼接	描述	示例	结果
加号拼接	使用 <code>+</code> 操作符连接两个字符串	<code>'hello' + ' ' + 'world'</code>	<code>'hello world'</code>
乘法拼接	使用 <code>*</code> 操作符重复字符串指定次数	<code>'a' * 3</code>	<code>'aaa'</code>
<code>join()</code> 拼接	使用 <code>join()</code> 方法将多个字符串拼接为一个字符串	<code>' '.join(['hello', 'world'])</code>	<code>'hello world'</code>
格式化拼接	使用 <code>str.format()</code> 或 <code>f-string</code> 拼接字符串	<code>'Hello, {}'.format('Meow')</code> 或 <code>f"Hello, {name}"</code>	<code>'Hello, Meow'</code>

三、数据处理模块

1. Pandas 子库与常见语法

子库/模块	描述	常见语法/函数	示例代码
<code>pandas.DataFrame</code>	用于表示表格数据，行列索引结构	<code>pd.DataFrame()</code>	<code>df = pd.DataFrame(data)</code>
<code>pandas.Series</code>	用于表示一维数据，类似于数组	<code>pd.Series()</code>	<code>s = pd.Series([1, 2, 3])</code>
<code>pandas.read_csv()</code>	用于读取 CSV 文件并返回 <code>DataFrame</code>	<code>pd.read_csv('file.csv')</code>	<code>df = pd.read_csv('data.csv')</code>
<code>pandas.to_csv()</code>	将 <code>DataFrame</code> 写入	<code>df.to_csv('file.csv')</code>	<code>df.to_csv('output.csv', index=False)</code>

	CSV 文件		
pandas.merge()	合并两个 DataFrame（类似 SQL 的 join 操作）	pd.merge()	df_merged = pd.merge(df1, df2, on='key')
pandas.concat()	沿某一轴（行或列）拼接多个 DataFrame	pd.concat()	df_concat = pd.concat([df1, df2])
pandas.groupby()	根据某些列对数据进行分组	df.groupby('column')	df_grouped = df.groupby('column').sum()
pandas.fillna()	填充缺失值	df.fillna()	df.fillna(0)
pandas.dropna()	删除缺失值的行或列	df.dropna()	df.dropna(axis=0)

2. NumPy 子库与常见语法

子库/模块	描述	常见语法/函数	示例代码
numpy.array()	用于创建 NumPy 数组	np.array()	arr = np.array([1, 2, 3])
numpy.arange()	创建一个具有指定间隔的一维数组	np.arange(start, stop, step)	arr = np.arange(0, 10, 2)
numpy.zeros()	创建一个全为零的数组	np.zeros()	arr = np.zeros((3, 3))

<code>numpy.ones()</code>	创建一个全为一的数组	<code>np.ones()</code>	<code>arr = np.ones((2, 3))</code>
<code>numpy.shape()</code>	获取数组的形状	<code>arr.shape</code>	<code>arr.shape</code>
<code>numpy.reshape()</code>	改变数组的形状	<code>np.reshape()</code>	<code>arr.reshape(2, 2)</code>
<code>numpy.sum()</code>	对数组的元素进行求和	<code>np.sum()</code>	<code>arr_sum = np.sum(arr)</code>
<code>numpy.mean()</code>	计算数组的平均值	<code>np.mean()</code>	<code>arr_mean = np.mean(arr)</code>
<code>numpy.dot()</code>	计算两个数组的点积	<code>np.dot()</code>	<code>result = np.dot(arr1, arr2)</code>
<code>numpy.linalg.inv()</code>	计算矩阵的逆	<code>np.linalg.inv()</code>	<code>inv_matrix = np.linalg.inv(matrix)</code>

3. Matplotlib 子库与常见语法

子库/模块	描述	常见语法/函数	示例代码
<code>matplotlib.pyplot</code>	最常用的子库，包含各种绘图函数	<code>plt.plot()</code>	<code>plt.plot(x, y)</code>
<code>matplotlib.figure</code>	用于创建和管理图表容器	<code>plt.figure()</code>	<code>fig = plt.figure()</code>
<code>matplotlib.axes</code>	处理坐标轴相关的对象	<code>plt.subplot()</code>	<code>ax = plt.subplot(1, 1, 1)</code>
<code>matplotlib.bar()</code>	创建条形图	<code>plt.bar()</code>	<code>plt.bar(x, height)</code>
<code>matplotlib.scatter()</code>	创建散点图	<code>plt.scatter()</code>	<code>plt.scatter(x, y)</code>
<code>matplotlib.hist()</code>	绘制直方图	<code>plt.hist()</code>	<code>plt.hist(data, bins=10)</code>
<code>matplotlib.title()</code>	设置图表的标题	<code>plt.title()</code>	<code>plt.title('My Plot')</code>
<code>matplotlib.xlabel()</code>	设置 x 轴标签	<code>plt.xlabel()</code>	<code>plt.xlabel('X Axis')</code>

<code>matplotlib.ylabel()</code>	设置 y 轴标签	<code>plt.ylabel()</code>	<code>plt.ylabel('Y Axis')</code>
<code>matplotlib.legend()</code>	添加图例	<code>plt.legend()</code>	<code>plt.legend(['Line 1', 'Line 2'])</code>

用途、意义：

- **Pandas** 主要用于处理表格数据，支持读取、处理、合并、分组、清洗等功能。
- **NumPy** 是数值计算的核心库，提供高效的数组操作、线性代数运算、随机数生成等功能。
- **Matplotlib** 是绘图库，支持各种图表绘制，如折线图、条形图、散点图等，广泛应用于数据可视化。

四、拓展

拓展 1. 不同语言程序的运行效率

编程语言	运行速度	适用场景
Python	较慢，尤其是数值计算和循环密集型任务	脚本编写、快速原型开发、数据分析、Web 开发
C/C++	非常快，特别适合底层和系统编程	系统开发、嵌入式编程、游戏开发
Java	相对较快，通过字节码和 JIT 提高执行效率	企业应用、大型分布式系统、Web 应用
JavaScript	较快，特别是在现代引擎的优化下	前端开发、服务器端开发（Node.js）
Go	快，接近 C/C++，特别适合并发处理	网络编程、分布式系统、微服务架构
Ruby	比 Python 稍慢，但比 PHP 更快	Web 开发、快速原型开发

自由教辅是挣脱锁链的知识生命体，它们在数字原野上以二进制基因自我复制，用知识共享协议作为进化密码。这些文字拒绝成为图书馆里标价的标本，而是化作可以任意拆解重组的乐高积木——人们用它们搭建适合学情的脚手架，学生将其改造成应对考试的思维导图，编程爱好者则把例题转写成程序。当传统教材还在用版权页计算印数时，自由教辅早已通过网络在电子设备间自由分发，每一次转发都在为教育平权投下选票。这里没有防盗水印的牢笼，只有无数双手共同托举的、永远处于“合并中”版的知识星辰。

信息技术重构

键盘敲破信息的茧，
知识在屏幕上蔓延成河。
我们拒绝锁链的图书馆，
让每个字节都生出翅膀——

像蒲公英选择风的方向，
像野火车吻整片荒原。
你可以在任何教室的晨光里，
拆解这星辰组成的齿轮。

白纸黑字的法律文书，
终将被改写进历史课本。
而我们正在创造：
所有公式都开源的春天。

Distribution Info:

Distributor: B5-Software
Release Date: 7 May. 2025
Version: 0.0.1

