上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Introduction to Computation (CS1602) · 第 1 讲

计算、机器与语言

Instructor: Tao Huang

Part I: 计算基础与 Python 入门 · 2026 秋季学期

什么是计算机

1943 年，美国陆军在报纸上登招聘广告。

职位名称：**computer**。

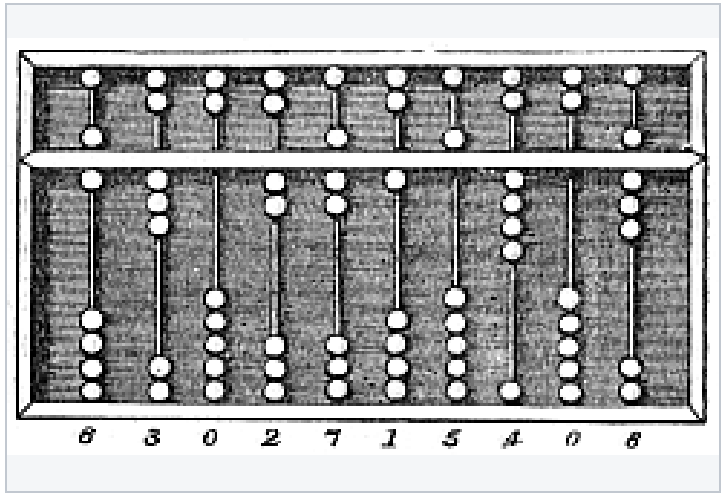
要求：数学好，有耐心，能长时间专注。

在那个年代，**computer 指的是从事计算工作的人。**

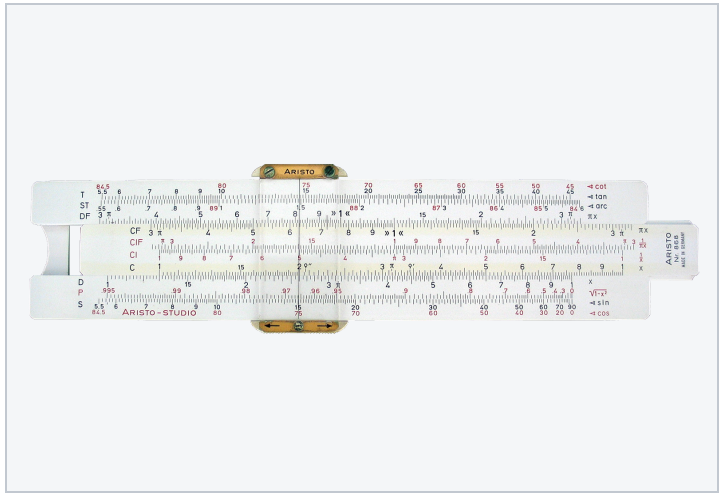
人工计算的时代

天文台和航海需要成千上万页的三角函数表、对数表，这些表由一批被称为 computer 的人手工算出来。

十六世纪的木刻：一边在纸上写数字算，一边拨算板。那时候这是两种**技术路线**之争。



算盘。中国人用了上千年，它把「记住中间结果」这件事交给了珠子。



计算尺。靠对数刻度把乘法变成加法，工程师一直用到 1970 年代——阿波罗登月就是它算的。

人工计算难免出错。**而一页表出错，所有依赖它的航海图都随之出错。**

让机器来做计算

让机器代替人做计算，这个念头出现得比多数人以为的早得多。

1900 年，一队采海绵的潜水员在希腊**安提基特拉岛**外遇上风暴，躲进背风处下潜，撞见一艘沉了两千年的古罗马货船。

捞上来的文物里有一块锈死的青铜，被当作杂物堆在博物馆。两年后才有人注意到：**那上面有齿。**

此后几十年靠 X 光和 CT 一层层看进去，才认出里面是 **三十多个精密咬合的青铜齿轮**，最小的一个只有十几毫米。

它是一台**天文计算器**：转动手柄输入一个日期，刻度盘就指出那天日月的位置、当晚的月相，还能推算日月食。

制造年代约在**公元前 100 年。用机器代替人计算，这个念头至少有两千年了。**

本讲主题

计算机凭什么能算？

本讲不涉及语法。四个问题：计算这件事的由来、机器的结构、三位学者各自的回答，以及本课程选择 Python 的理由。

今日内容

- **课程说明** —— 规则、成绩、AI 使用政策
- **计算的历史** —— 从算盘到摩尔定律
- **计算机的组成** —— 冯·诺依曼体系，以及「一切都是 0 和 1」的含义
- **计算理论的三个来源** —— 图灵、香农、哥德尔各自回答了什么
- **编程语言** —— 本课程选择 Python 的理由
- **程序的执行过程** —— 以及你的第一个程序

第 1 段 / 共 6 段

课程说明

刚才那个问题先挂着 —— 「计算机凭什么能算」。

这一段是规矩：**成绩怎么算、AI 能用到什么程度、卡住了找谁。**

课程组

主讲 黄涛（Dr. Tao Huang）

助教 李晓欧（lixiaoou@sjtu.edu.cn）、苏奕涵（suyihan1@sjtu.edu.cn）

课程网站 taohuang.info/cs1602 讲义、实验、课件、提交、答疑全在上面

课程微信群 通知和临时答疑走群里，扫左边的码

课程相关的问题，**请优先在群内提出。**

课程目标

- 拿到一个用中文描述的问题，把它拆成步骤，用 Python 写出来、跑通
  - 比如「统计这份名单里每个姓出现了多少次」
- 程序出错时，知道该看哪一行
- 读懂别人写的代码，判断它写得好不好、在什么输入下会崩
- 与 AI 协作编程：明确它该做什么，并能验证它的输出

本课程**不预设任何编程基础**。

COURSE MAP

课程章节

- **Part I** 计算基础与 Python 入门 —— 写出含输入、判断、循环、函数的完整程序
- **Part II** 数据的组织 —— 为问题选对容器，处理批量数据
- **Part III** 问题求解与 AI 协作 —— 拆解问题；读懂、审查、验证别人写的代码
- **Part IV** 抽象与组织 —— 用类和模块组织中等规模程序
- **Part V** 健壮性与真实世界 —— 评估效率、处理异常、读写真实数据
- **Part VI** Pythonic 与现代实践 —— 写地道的 Python，用 AI 完成一个真实项目

课时安排

- 理论课讲清楚原理
- 实验课形成动手能力
  - 自己编写、自己调试、自己定位错误，这三步无法由他人代替
- 上课**请携带电脑**

写代码的能力是在实验课上形成的。

GRADING

成绩构成

- **小作业 ×5 —— 10%**　上机课的内容，每次覆盖两到四周，合并交一次
- **个人作业 ×3 —— 15%**　汉诺塔 · 素数计数 · 大整数运算，各给两周
- **小组作业 ×1 —— 25%**　两人一组，做一个简化版 NumPy。**只提交，不答辩**
- **期末考试（闭卷） —— 50%**

期末占 50%；**另一半分散在整个学期里**，每周都有机会拿到，也每周都可能漏掉。

AI 使用政策

这是这门课最重要的一条规则

AI 政策的三个阶段

- **第 1–7 周：不允许用 AI 生成或补全代码。**
  - 请关闭编辑器里的 AI 补全功能
  - 可以用 AI 问概念、查文档、解释报错 —— 但不能让它替你写代码
- **第 8 周起：允许使用，但每次提交要附一段声明。**
  - 说明你用了什么工具、它生成了哪部分、你如何验证它是对的
- **第 15–16 周（团队项目）：完全开放。**

完整规则在课程网站的「AI 政策」页。

前七周禁用 AI 的理由

- 前七周你要建立的是两样东西：**代码直觉**和**调试能力**
  - 看到一段代码，能预判它会输出什么
  - 看到一个报错，能猜到问题在哪一行
- 这两项能力没有捷径，只能通过自己编写、出错、排查获得
- 由 AI 代写，代码可以运行，**但你无法判断它在什么情况下会出错**

第八周之后能否用好 AI，**取决于这一阶段积累的判断力。**

这条政策的目的

前七周的限制，目的是让第八周之后的你具备判断能力

遇到问题的处理顺序

- **读报错信息。** 它通常直接告诉你哪一行、什么问题
  - 常见的问题是看到报错就跳过，并没有真正去读它
- **查官方文档。** docs.python.org 有完整中文版
- **搜索。** 把报错里的关键部分贴进搜索引擎，大概率有人遇到过
- **问同学、问助教。** 课程群里有问必答
- **问 AI —— 在政策允许的范围内**

有效提问的三个要素

- **你想做什么** —— 目标是什么
- **你写了什么** —— 贴出代码，而不是复述代码
- **发生了什么** —— 贴出完整的报错信息

「我的代码不对，怎么办」——**这样的提问无法得到有效回答。**

课程网站

- **讲义** —— 每一讲一篇，比幻灯片详细得多，是这门课的教材
- **课件** —— 本课使用的幻灯片，可逐页翻阅，也可下载 PDF
- **讲义版课件** —— 每页幻灯片配上对应的讲解，适合课后阅读
- **实验** —— 每周的题面和要求
- **实验提交** —— 交作业、看判题结果
- **随堂小测** —— 本讲稍后会用到

中英文两版，右上角切换。讲义和实验都是双语的。

这门课的两个入口

两个都要收藏。日常在课程网站，**Canvas 是最终交作业和看成绩的地方**。

课程网站 `taohuang.info/cs1602` —— 讲义、课件、实验、提交判题、随堂小测，**平时都在这里**

Canvas `oc.sjtu.edu.cn/courses/97484` —— 学期末把评测报告交到这里，成绩和教务通知也走这边

平时的题目和判分在课程网站；**学期末下载一份评测报告，交到 Canvas** —— 源码已经嵌在报告里，不用另外交 .py。

CHECKLIST

本节课后待办

- **加课程群**（二维码在前面那页）
- **课程网站和 Canvas 都加进书签**（上一页两个码）
- **本周实验课请携带电脑** —— 环境安装是实验课的第一项内容

环境安装安排在 **Lab 1**，届时会逐步演示。

随堂小测 · 第 1 轮

扫码作答 —— 这一轮两题

1 你之前用过哪些编程语言？

2 AI 补全替你写了一个循环，算不算违规？

第 1 题是问卷，没有对错；第 2 题必须答对，它决定你前七周怎么写作业。

https://taohuang.info/cs1602/zh/quiz/1

回到开头那个问题

1943 年招的那个 computer 是人。今天这台机器凭什么能算?

规矩讲完了。接下来四段，就是在回答这一个问题 —— 从人算不动开始，一直到你写出第一行代码。

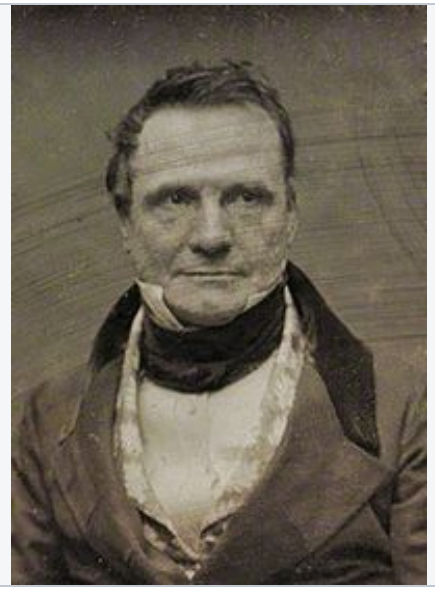
第 2 段 / 共 6 段

计算的历史

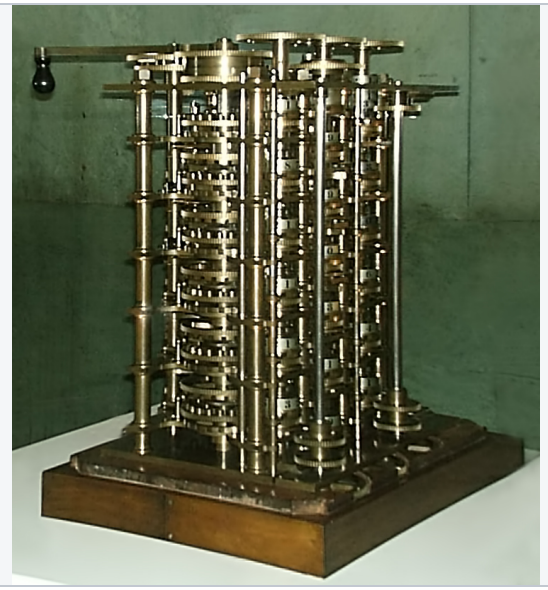
开头那三页说了：人自己算，**算不动，而且会算错。**

这一段问：想让机器接手，**卡在哪儿、后来怎么过去的。**

十九世纪的机械计算机



查尔斯·巴贝奇（Charles Babbage, 1791–1871）。他受不了对数表里的错误，决定造一台不会错的机器。



差分机。用齿轮做加法，摇手柄出结果。后来他又设计了更野心的**分析机**。

分析机已经有了处理器、存储器和输入输出的雏形，用**打孔卡**编程。

分析机的结局

当时的机械加工精度跟不上设计

但图纸留了下来。巴贝奇常被称为「计算机之父」，靠的就是这套图纸。

第一个计算机程序

她为一台还不存在的机器，写下了一组可以执行的指令。

阿达·洛芙莱斯（Ada Lovelace, 1815–1852）。诗人拜伦的女儿，数学家。

1843 年，她把一篇介绍分析机的法文文章译成英文，又在译文后面附了一组注释——**篇幅是原文的三倍**。

其中的「**注释 G**」给出了用分析机计算**伯努利数**的完整步骤：怎么循环、中间结果存在哪一格、什么时候再取出来。

她面对的是一台**从未存在过的机器**——没有硬件可以试，全部推演都在纸上完成。

这被认为是**世界上第一个计算机程序**——比第一台真正的计算机早了整整一百年。

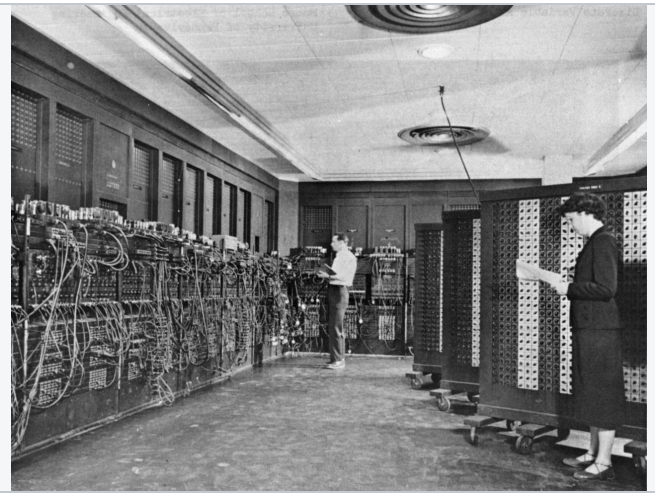
洛芙莱斯的洞见

- 巴贝奇想的是：这台机器能**算数**
- 洛芙莱斯想的是：只要能把别的东西编码成数字 ——
  - 比如音符 ——
- **它就能处理任何东西**

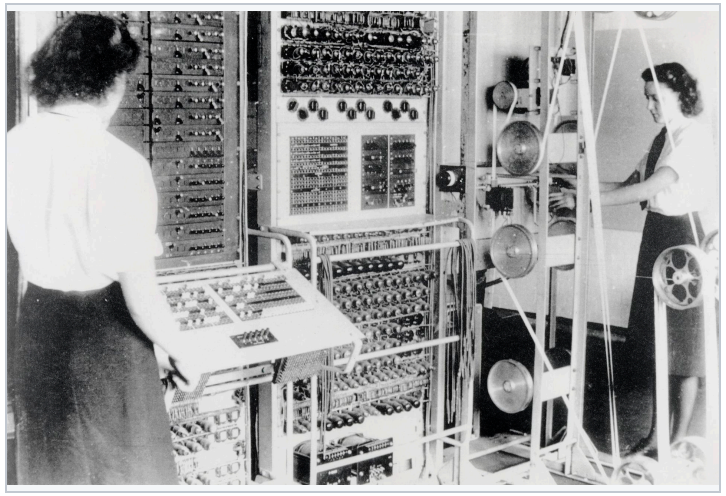
今天的照片、音乐与文本都是数字。**这一点她在 1843 年已经指出。**

开关器件的三次更替

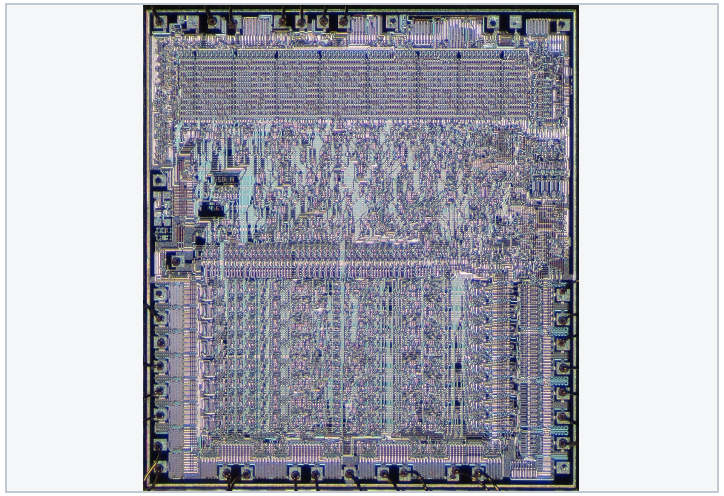
从继电器到电子管到晶体管，换的其实是同一样东西：**开关**。



电子管（1940s）。ENIAC 有 18000 支，占满一个大厅，开机时费城的灯会暗一下。



ENIAC 的操作员。**当年的程序员大多是女性**——那时候「编程」被当成文书工作。

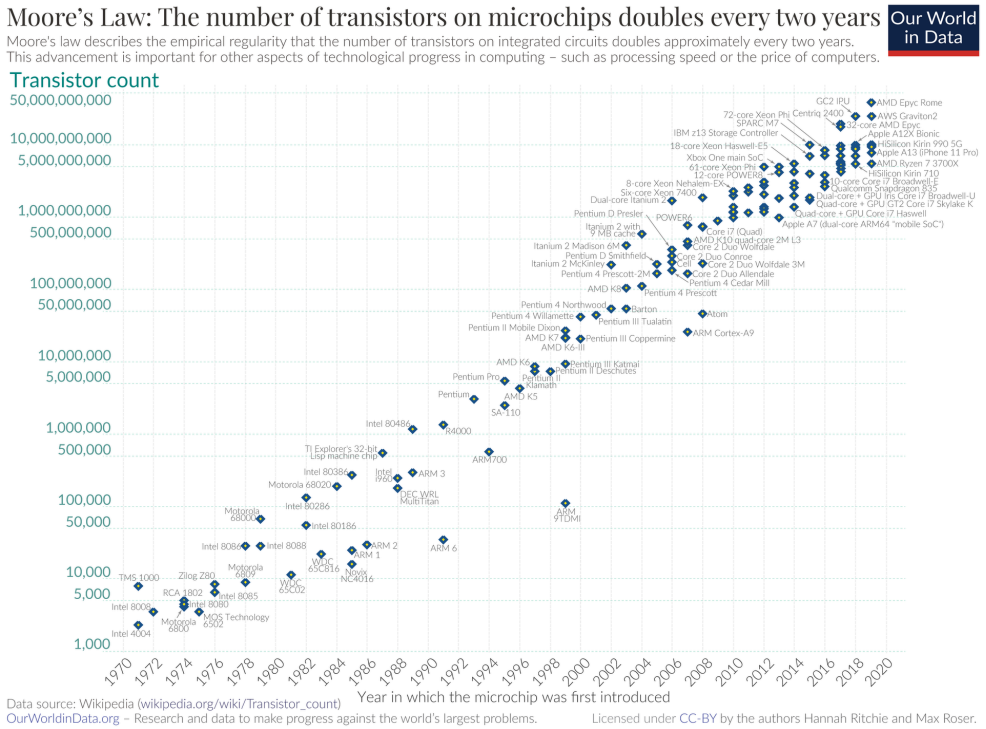


晶体管 → 集成电路（1947 至今）。今天这么大一块上有几百亿个开关。

三次更替的对象始终是开关。**每一次都更小、更快、更省电。**

摩尔定律

1965 年，英特尔创始人之一戈登·摩尔注意到一个规律。



集成电路上的晶体管数目随年份变化。纵轴是对数刻度 ——一条直线意味着指数增长。

集成电路上能容纳的晶体管数目，大约每 18 个月翻一番。

摩尔定律的三种读法

- **性能** —— 同样的价格，18 个月后能买到快一倍的机器
- **价格** —— 同样的性能，18 个月后价格减半
- **尺寸** —— 同样的功能，18 个月后体积缩小一半

它不是物理定律，是**经验观察**，而且近年已经明显放缓。

算力增长的影响

很多二十年前「算不动」的问题，今天可以硬算

你手里这台手机，算力超过 1990 年代的超级计算机。很多学科现在的做法是：先建模型，让机器算，最后才回实验室验证。

人工智能的两个里程碑

1997，深蓝击败国际象棋世界冠军卡斯帕罗夫。路子是穷举——把所有可能都算一遍。

2016，AlphaGo 击败李世石。围棋的搜索空间太大，穷举走不通，它换了一条路：**学习**。

两者相隔 19 年。当前的模型已不限于博弈，**也能编写代码**。

两百年，换的其实是同一样东西

齿轮 → 继电器 → 电子管 → 晶体管

巴贝奇的齿轮卡在加工精度上，后面三种都是电控的开关，一次比一次小、快、省电。但它们干的是同一件事：**通，或者断**。

所以下一段只剩一个问题：**只有通和断两个状态，怎么装得下一整台计算机？**

第 3 段 / 共 6 段

计算机的组成

上一段的结论：机器算数靠的是**开关**，而开关只有通和断。

这一段问：**只有两个状态，怎么装得下一整台计算机。**

冯·诺依曼与 EDVAC 报告

约翰·冯·诺依曼（John von Neumann, 1903–1957）。匈牙利裔数学家，在数学、物理、经济学上都留下了奠基性的工作。

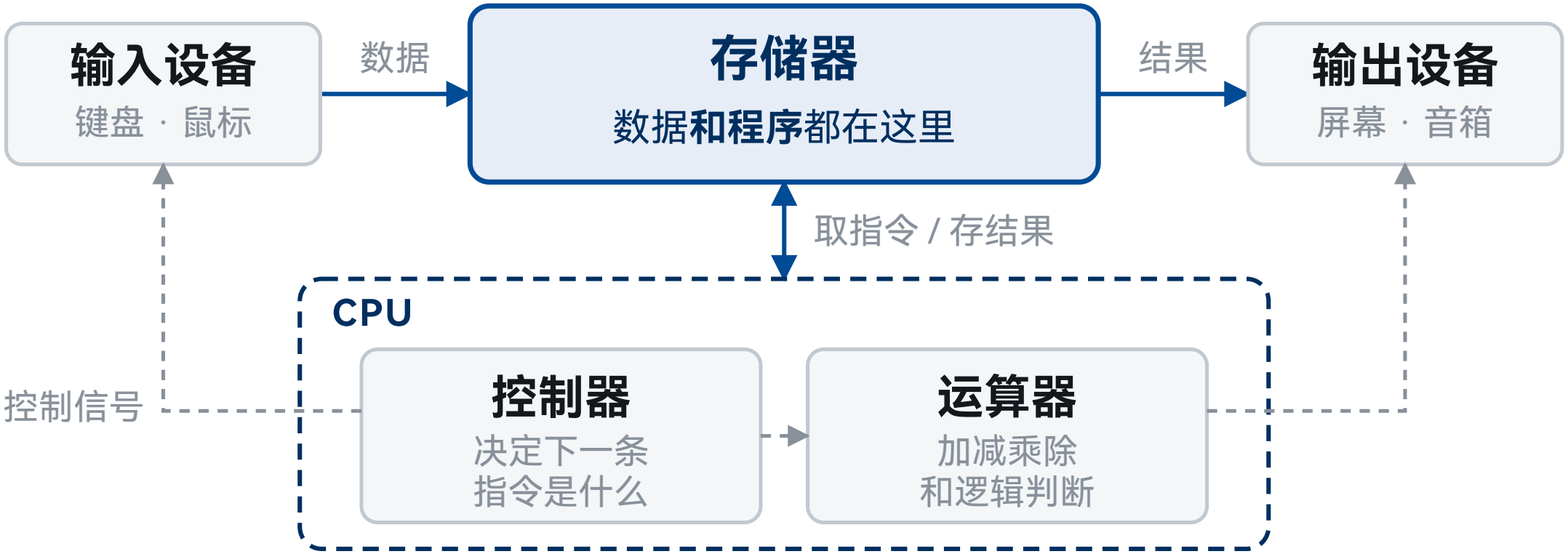
1945 年 6 月，他为 EDVAC 计算机项目写了一份报告草稿，在里面描述了一种计算机的组织方式：**五个部件，而且程序和数据放在同一个地方。**

这份草稿本来只在项目组内部传阅，却被油印散了出去。此后造机器的人，基本都照着它建。

需要说明：这个结构不是他一个人的功劳，ENIAC 团队的埃克特和莫奇利做了关键工作——只是那份报告上**署了他一个人的名字**。

今天你能碰到的几乎所有计算机 —— 手机、笔记本、服务器、路由器 ——**都是这个结构的变体**。

冯·诺依曼体系的五个部件



实线是数据，虚线是控制。 控制器指挥所有人，但它自己不搬数据 —— 这是「谁在干活」和「谁说了算」的分工。

对着自己的笔记本，这五样都能一一指出来。**注意存储器那一格：它存的不只是数据。**

存储程序思想

程序和数据存在同一个地方，用同样的方式表示

这叫「存储程序」思想。后果是：机器不用重新接线就能换任务，只要把另一段程序读进存储器。而且——程序本身也可以被当作数据来处理。

二进制与进制表示

我们把它们记作 1 和 0。一个 0 或 1 叫一个 **bit**（位），八个 bit 叫一个 **byte**（字节）。

进制

```
1  n = 2026
2
3  print("十进制:", n)
4  print("二进制:", bin(n))
5  print("八进制:", oct(n))
6  print("十六进制:", hex(n))
```

output

```
十进制: 2026
二进制: 0b11111101010
八进制: 0o3752
十六进制: 0x7ea
```

`0b` `0o` `0x` 是前缀，告诉你后面这串数字按几进制读。

二进制序列的解释

同一串 01000001

```
1  bits = 0b01000001
2
3  print("当作整数:", bits)
4  print("当作字符:", chr(bits))
5  print("当作红色深浅:", bits / 255)
```

output

```
当作整数: 65
当作字符: A
当作红色深浅: 0.2549019607843137
```

二进制序列本身没有意义，意义来自约定。记住这一句 —— 第 5 段讲的「语言」，就是人和机器之间的一层约定。

操作系统

- 操作系统管理硬件资源，并向其他程序提供**统一的服务**：
  - 你写 `open("data.txt")` 时，不需要知道硬盘的磁头在哪个磁道
  - 你同时开着浏览器和音乐播放器，是操作系统在决定 CPU 现在给谁用
  - 你按下键盘，是操作系统把这个事件送到当前窗口

Windows、macOS、Linux、iOS、Android 都是操作系统。**第 8 次实验会带你接触 Linux 命令行**——服务器几乎都跑在上面。

计算理论的三个来源

前面讲的都是机器**怎么造出来**。现在换一个问题：它**能算什么**。

注意年份：这三项工作是 **1931**、**1936**、**1937**，都在第一台电子计算机之前。**理论先到，机器后到。**

图灵机

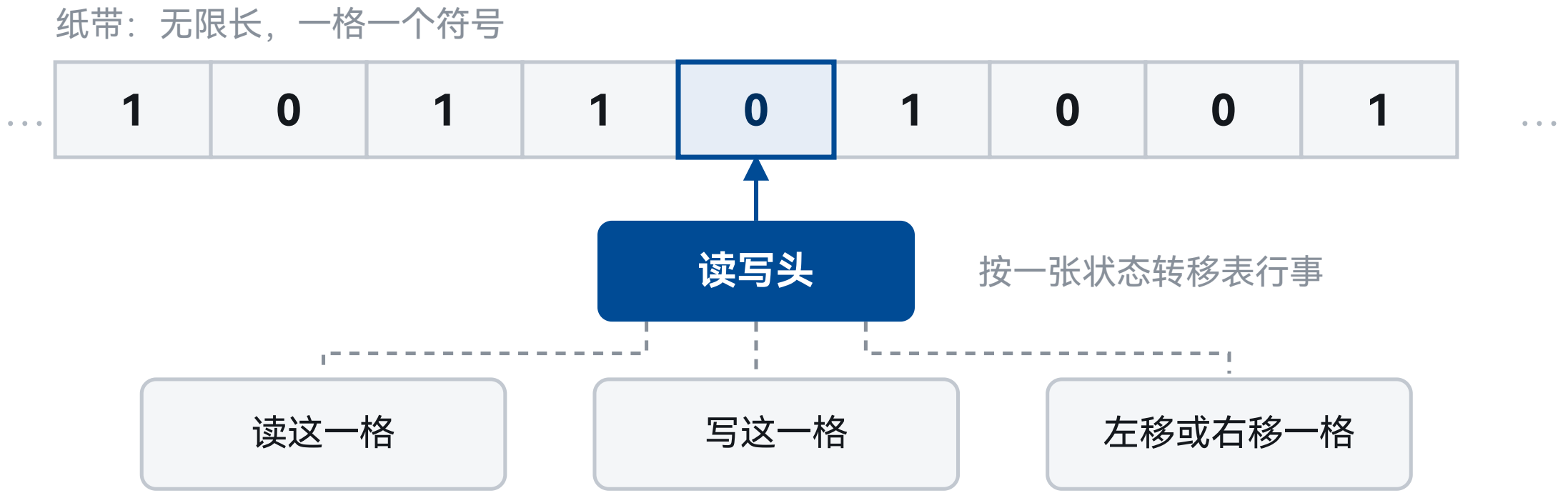
1936 年，他提出一个极简的假想机器：一条无限长的纸带，一个读写头，一张状态转移表。

阿兰·图灵（Alan Turing, 1912–1954）

它每次只能读一格、写一格，并左移或右移一格。**仅此而已。**

图灵机由三样东西组成

一条无限长的纸带、一个读写头、一张状态转移表。



它每一步只能读一格、写一格、左右挪一格。仅此而已。第一次看到的人通常觉得它太简陋，干不了什么事。

可计算性的定义

任何能被机械地计算出来的东西，都能用它算出来

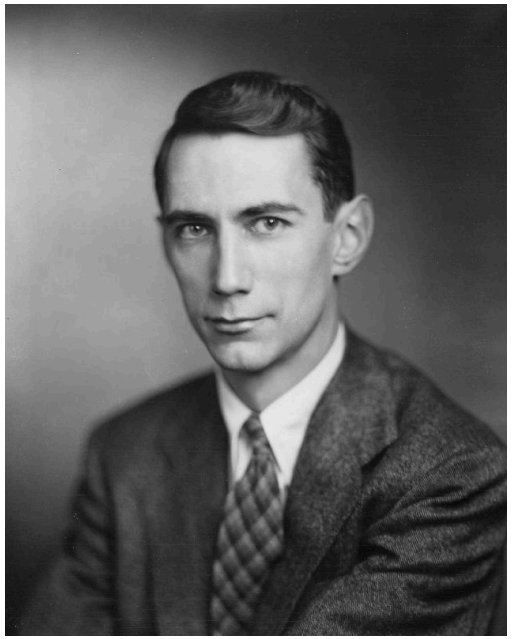
「可计算」自此有了精确的定义。

不可计算问题与停机问题

- **有些问题任何计算机都解不了** —— 不是算得慢，是根本不存在算法
- 最著名的是**停机问题**：
  - 不存在一个程序，能判断任意给定的程序是否会陷入死循环

推论：既然连「这个程序会不会死循环」都无法自动判断，那么**「这段代码有没有 bug」当然也无法自动判断**。

香农与信息的度量



克劳德·香农（Claude Shannon, 1916–2001）。美国数学家、电气工程师。

他在国内的名气不如图灵，但今天技术的形态更直接地由他决定：你天天说的「**比特**」，第一次出现在印刷品上就是他 1948 年那篇论文里。

他也是出了名的爱玩：骑独轮车穿过贝尔实验室的走廊，做过一台唯一功能是伸出手把自己关掉的机器，还做过世界上第一只会走迷宫、并且记得住路的电子老鼠。

两项奠基性的工作相隔十一年，都由他一个人完成：**一项让电路能做逻辑，一项让信息可以被计量。**

香农的两项工作

- **1937 年，硕士论文**：布尔代数（真/假、与/或/非）可以用继电器电路实现
  - 这篇论文让「用电路做逻辑运算」成为可能，是所有数字电路的理论起点
- **1948 年，《通信的数学理论》**：用 bit 度量信息量，并证明了信道容量的极限
  - 今天的手机通信、文件压缩、纠错编码，都建立在这套理论上

第 1 件补上了第 3 段留下的缺口：**开关凭什么配叫计算** —— 因为与、或、非都能用开关电路做出来。（发表时他 21 岁。）

哥德尔不完备定理

库尔特·哥德尔（Kurt Gödel, 1906–1978），1931年。

哥德尔（左）与爱因斯坦（右）在普林斯顿

不完备定理：任何足够强的、逻辑自洽的形式系统，都存在在系统内既无法证明也无法证伪的命题。

三项理论的关系

什么能算（图灵）· 算什么（香农）· 什么算不了（哥德尔）

哥德尔的不完备定理、图灵的停机问题，还有后来一堆「不可判定」的结论，说的其实是同一件事。

第 5 段 / 共 6 段

编程语言

已经确定的两件事：机器里只有 0 和 1；**意义来自约定**。

这一段问：**人怎么把自己的意图，翻译成那一串 0 和 1。**

机器语言与汇编语言

机器语言

```
01001000 01100101
01101100 01101100
01101111 00100000
01010111 01101111
01110010 01101100
01100100
```

汇编语言

```
section .text
  global _start
_start:
  mov edx, len
  mov ecx, msg
  int 0x80
```

这两段都在做**同一件事**：让屏幕上出现 Hello World。

同一件事在 Python 里的写法

python

```python
print("Hello world")
```

output

```
Hello world
```

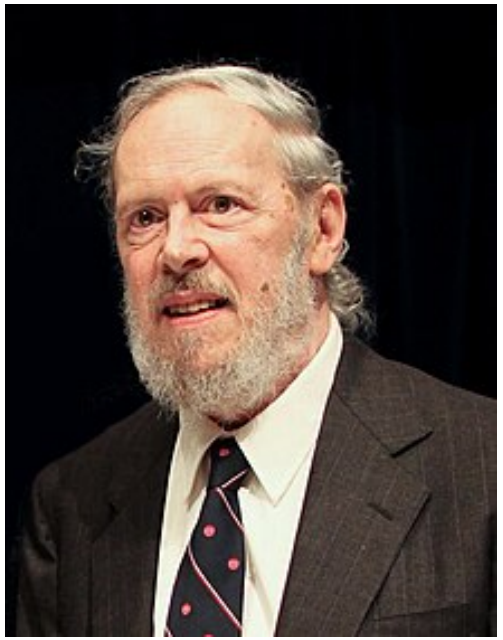
一行。「高级语言」的含义正在于此：编写者不必处理机器层面的细节。

编程语言的发展趋势

- 语言是人与机器之间的翻译层
- **总体方向：更易编写，也更难写错**
- 语言之间相互借鉴 —— 一门语言里好用的特性，几年后往往出现在其他语言中
- 你现在学到的概念，十年后大多仍然适用

因此，与其纠结先学哪门语言，**不如把解决问题的方法学扎实**。

C 语言与丹尼斯·里奇



今天几乎所有主流语言，往上追都能追到 C。

丹尼斯·里奇（Dennis Ritchie, 1941–2011）。1972–73 年在贝尔实验室做出 **C**，然后用它重写了 Unix 内核。后来斯特劳斯特卢普把类加进 C，成了 **C++**。

Python 解释器本身用 C 实现。**你调用的 `print`，其底层是 C 代码。**

本课程选择 Python 的理由

- **对初学者友好** —— 学习曲线不陡，细节少（对比 C、C++、Java）
- **语法简明，功能强大** —— 同一件事，代码量常常是别的语言的三分之一
- **实用** —— 数据处理、科学计算、网站、AI，一套语法全都能干
- **开放** —— 开源，不受任何一家商业公司控制（对比 Java、C#）

以及一条现实的理由：**未来十年你很可能仍会用到它。**

Python 的实际应用领域

- **人工智能几乎全在 Python 上** —— PyTorch、TensorFlow、JAX，主接口都是 Python
- GitHub 2025 年度报告：Python 有 **260 万贡献者**，同比增长 **48%**，在 AI 与数据科学领域领先
- **科学计算** —— NumPy 的论文 2020 年发在 **Nature**；2019 年第一张黑洞照片的成像代码（eht-imaging）就是 Python 写的
- 此外还有数据分析、网站后端、自动化运维、金融量化、生物信息

你这学期写的每一行，用的都是真实工程里在用的那门语言。

Python 的来历

吉多·范罗苏姆（Guido van Rossum）。1989 年圣诞节假期，他为了打发时间开始写 Python。

他的车牌。名字来自 **Monty Python** 剧团，**不是蟒蛇**——虽然 logo 后来还是画成了蛇。

起因是一个假期里的业余项目。三十多年后，它成为使用最广泛的语言之一。

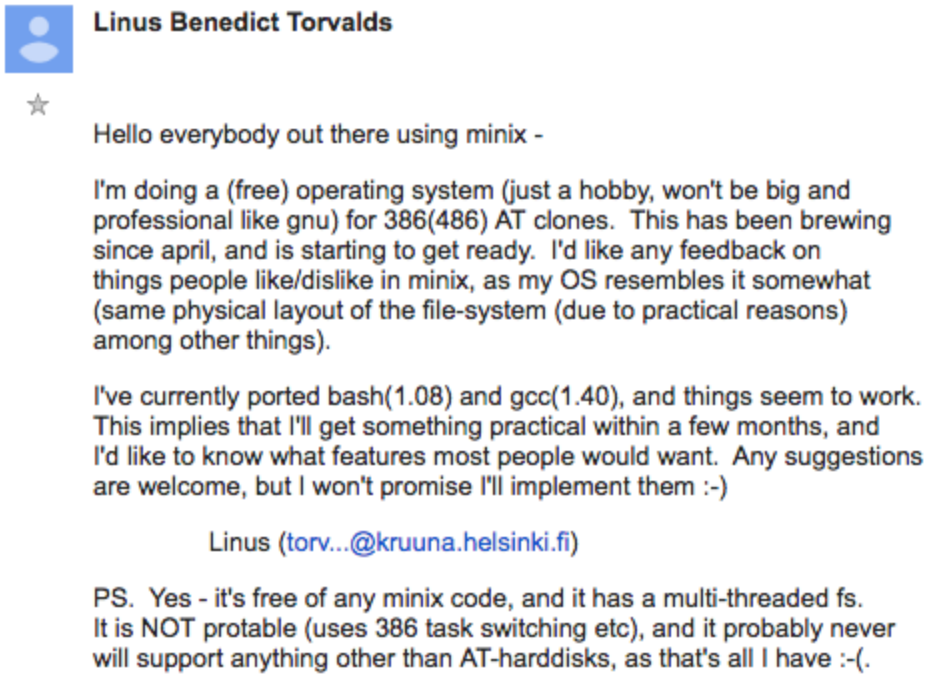
自由软件运动

1960 年代，硬件赚钱，软件是白送的。后来硬件利润变薄，厂商开始卖软件、并且不再给源代码。

理查德·斯托曼（Richard Stallman），自由软件运动的发起人

斯托曼的回应是：**写一套所有人都能自由使用和修改的软件**。GPL 许可证就是那时候的产物。

Linux 的诞生



Linus Benedict Torvalds

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Linus Torvalds 在新闻组里宣布他在写一个操作系统。他说这「只是个爱好，不会像 GNU 那样又大又专业」。

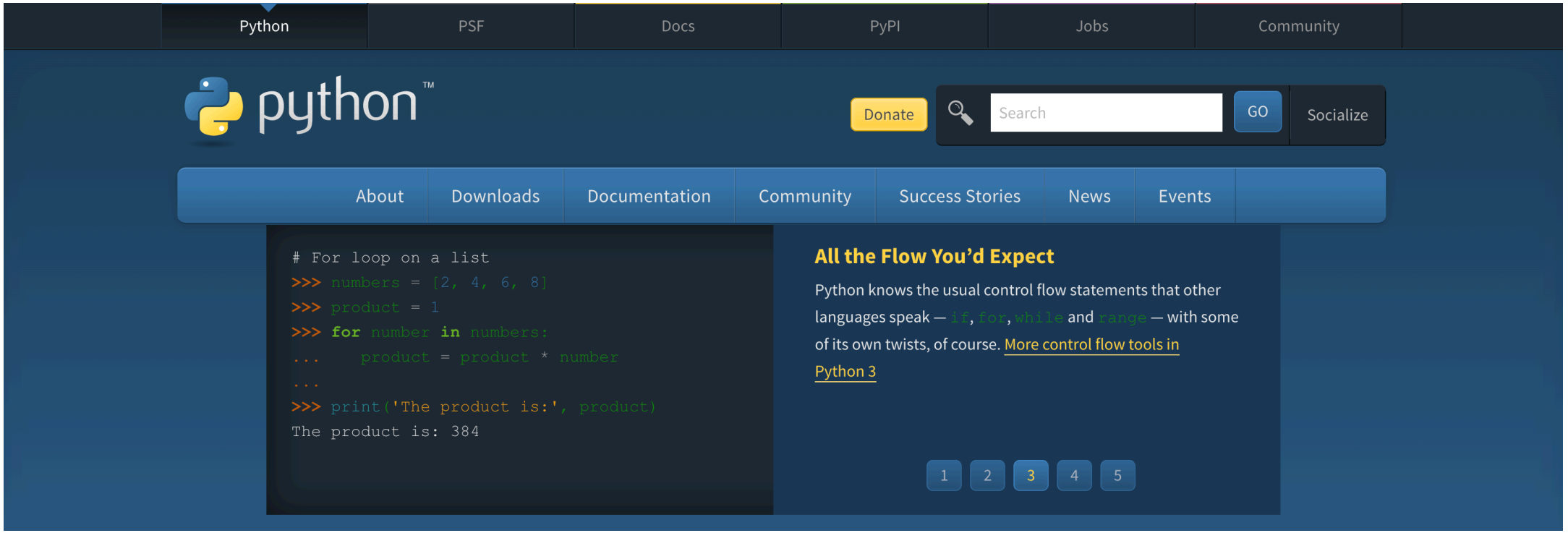
那个「爱好」就是 **Linux**。今天世界上绝大多数服务器跑的是它。

本课程使用的免费软件

- **Python 本身** —— 免费、开源
- **VS Code** —— 免费
- **Linux** —— 免费，服务器上到处都是
- **你会用到的几乎所有库** —— NumPy、pandas、matplotlib，全部免费

这些成果由几代人逐步争取而来，**今天可以直接使用**。

Python 官方网站



*下载走 **Downloads**。官网现在最新的是 3.14，但**这门课统一用 3.13** —— 安装包课程网站上有，Lab 1 直接下那一份。*

请认准这个域名。搜索结果靠前的下载站常附带捆绑软件，或者给你一个被改过的安装包。

第 6 段 / 共 6 段

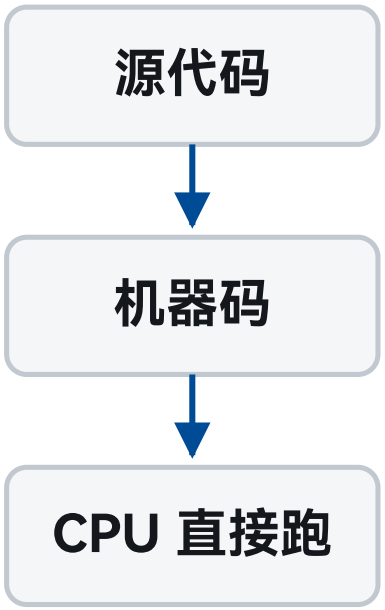
程序的执行过程

第 3 段说过：**程序本身也是数据**，所以程序能被另一个程序读。

这一段就是那句话的兑现：**你写的字符，怎么变成 CPU 上的动作。**

编译与解释

编译 COMPILE

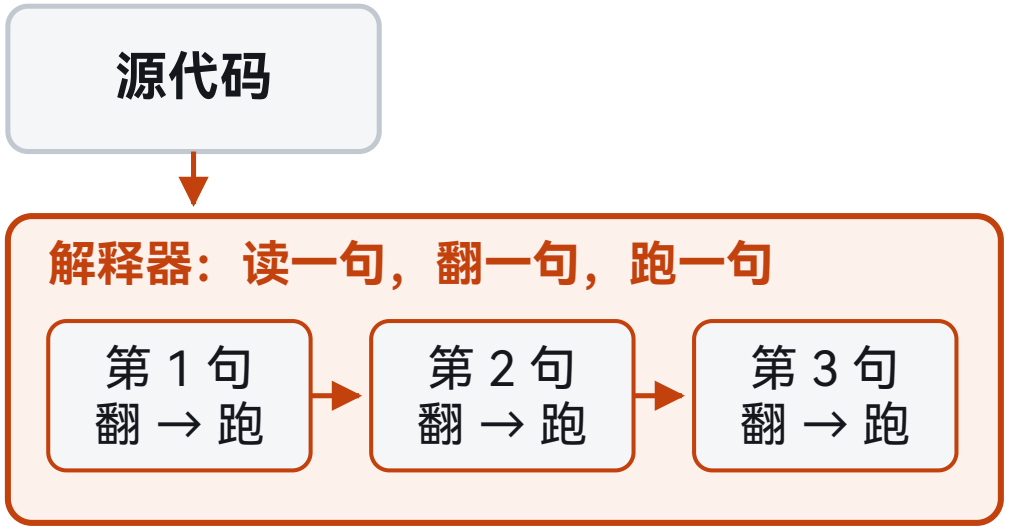


编译器
跑之前，一次翻完

生成一个
可执行文件

C · C++ · Rust ——　快，但换平台要重编

解释 INTERPRET



Python · JavaScript ——　慢，但到处都能跑

差别在于翻译发生在什么时候 —— 编译是跑之前一次翻完，解释是边跑边翻。

Python 是解释型的。 你运行 `python a.py` 时，启动的是一个叫解释器的程序，它读你的文件，一句一句执行。

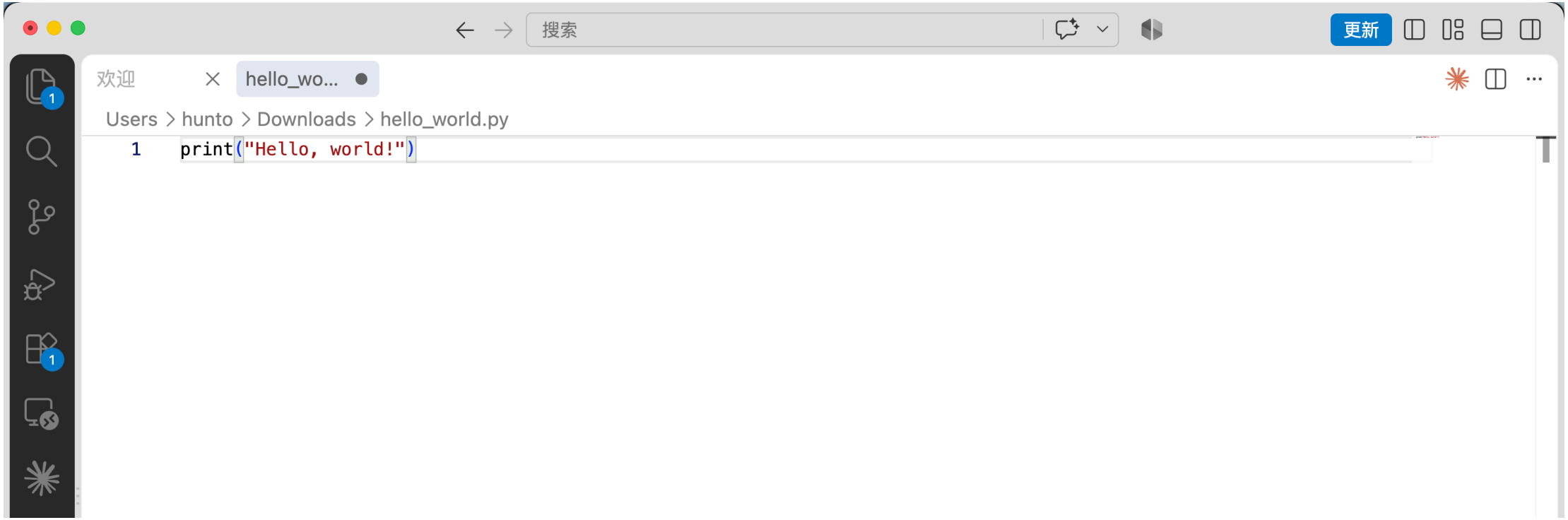
Python 环境的组成

- 装的是**解释器**（一个叫 python 的程序）
- 外加一套**标准库**（一堆已经写好的 .py 文件）
- 以及 **pip** —— 用来装别人写的库

python a.py 的意思是：**让解释器去读 a.py 并执行它。**

集成开发环境（IDE）

IDE（集成开发环境）= 编辑器 + 运行 + 调试 + 补全，装在一个窗口里。



*VS Code —— 这门课推荐的编辑器，免费。左边一竖条从上到下是：文件、搜索、版本控制、**运行和调试**、插件。中间是代码，一个标签页一个文件。*

第 1–7 周请关闭 AI 补全功能，安装时即可完成设置。

你的第一个程序

第一个程序

```
1  print("Hello world")
2  print("上海交通大学")
3  print("我叫 _____，学号 _____")
```

output

```
Hello world
上海交通大学
我叫 _____，学号 _____
```

print() 的意思是：把括号里的东西显示出来。

阅读报错信息

故意写错

```
1  pirnt("Hello world")
```

traceback

```
Traceback (most recent call last):
  File "example.py", line 1, in <module>
    pirnt("Hello world")
    ^^^^^
NameError: name 'pirnt' is not defined. Did you mean: 'print'?
```

报错已经指明：**第 1 行**，pirnt 未定义，并给出了拼写建议。

Python 之禅

试试这个

```
1  import this
```

output

```
点上面的「▶ 运行」看看
```

点一下运行 —— 它会打印出十九条 **Python 之禅**，这门语言的设计信条。

Python 之禅（节选四条）

- **Beautiful is better than ugly.** 好看胜过难看
- **Simple is better than complex.** 简单胜过复杂
- **Readability counts.** 可读性是要算数的
- **If the implementation is hard to explain, it's a bad idea.**
  - 如果一个实现很难解释清楚，那它多半就是个坏主意

选出你现在最认同的一条，**学期末再回来对照。**

随堂小测 · 第 2 轮

扫码作答 —— 这一轮四题

1. 哪个**不是**摩尔定律的读法？
2. `01000001` 到底是什么？
3. 停机问题能推出什么？
4. `python hello.py` 干了什么？

今天四个要点各一题。一题一题放，答完一题公布一题。

https://taohuang.info/cs1602/zh/quiz/1

所以，计算机凭什么能算

- **因为「算」可以拆成一串只有通断两种结果的判断** —— 这是香农 1937 年证明的：与、或、非都能用开关电路做出来
- **因为这样的机器不需要为每个任务重造** —— 程序和数据一样是 0 和 1，换任务只是换一段存进去的 0 和 1
- **因为「能算什么」是有定论的** —— 图灵证明了任何机械可算的东西，那条纸带都算得出来；同时也划出了算不了的那部分
- **而它之所以快到有用**，是因为开关越做越小 —— 摩尔定律那条线

这四条里没有一条讲到 Python。**语言是后面的事，今天这一讲讲的是它底下那一层。**

其他三条经验法则

- **梅特卡夫定律** —— 网络的价值与节点数的平方成正比
  - 这解释了为什么社交平台赢家通吃
- **墨菲定律** —— 如果一件事有可能出错，它就一定会出错
  - 写程序的人对这条会有越来越深的体会
- **Knuth 定律** —— 过早的优化是万恶之源
  - 这条我们在第 11 讲会认真讲

本讲小结：一条线

- 人自己算 —— **算不动，而且会算错**（所以才要造机器）
- 机器靠**开关**接手 —— 两百年只在把开关做小做快
- 开关只有通断 —— **所以机器里只有 0 和 1**
- 0 和 1 本身没有意义 —— **意义来自约定**
- **程序也是 0 和 1**，和数据存在一起 —— 所以程序能被程序读，才有了解释器
- 至于**能算什么**，图灵在机器造出来之前就划好了边界

本次实验：环境搭建与第一个程序

- **装 Python 3.13 和 VS Code**，跑通 `print("Hello world")`
  - 两种方式各跑一次：VS Code 的 ▶ 按钮，和终端里 `python hello.py`
- **A 段 · 必做** —— 自我介绍、学号转进制、猜四个算式、故意制造三种报错
- **B 段 · 深入** —— 十行代码看混沌：初值差百万分之一，二十五轮后面目全非
- **C 段 · 基本功** —— 命令行 `pwd` `ls` `cd`。每周一小段，**期末要考**
- **走通一次提交** —— 用学号注册评测平台（不要口令），交「小作业 1」

三段**同时开放**，不必按顺序做完；以后每周都是这个结构。**装环境会吃掉大半时间，所以本周三段都轻。**

下一讲预告

计算机只有 0 和 1，那它怎么表示 -3、3.14 和「你好」？

下一讲：字符与数。你会看到 0.1 + 0.2 != 0.3，并且明白这不是 Python 的 bug。