0 第一讲：基本概念

笔记本：	浙江大学《数据结构》		
创建时间：	2025/3/6 18:09	**更新时间：**	2025/4/14 21:08
作者：	panhengye@163.com		
URL：	vscode-file://vscode-app/c:/Users/panhe/AppData/Local/Programs/cursor/reso...		

1. 什么是数据结构？
    1. 书架放书的例子：解决问题方法的效率，跟数据的组织方式有关
    2. 输出正整数的函数的例子：解决问题方法的效率，跟空间的利用效率有关
    3. 多项式求解（霍纳法则）：解决问题方法的效率，跟算法的巧妙程度有关

需求：传入一个正整数n，顺序打印从1到n的全部正整数

```
# 实现方式1：循环
def printN(n):

    # 传入一个正整数n，顺序打印从1到n的全部正整数

    i = 1
    while i <= n:
        print(i)
        i += 1

printN(10)
```

```
# 实现方式2：递归
def printN(n):
    # 传入一个正整数n，顺序打印从1到n的全部正整数

    def _print_helper(current):
        if current > n:
            return
        print(current)
        _print_helper(current + 1)

    _print_helper(1)

printN(10)
```

递归和迭代在时间复杂度上通常相似，但递归（什么是递归（recursion））会消耗更多的栈空间。对于较大的输入值，Python 的递归不是最佳选择，尤其是像打印数字这样的简单操作。

- 数据对象在计算机中的组织方式
  - 逻辑结构
  - 物理存储结构
- 数据对象必定与一系列加在其上的操作相关联
- 完成这些操作所用的方法就是算法

抽象数据类型（abstract data type）：

- 数据类型
  - 数据对象集
  - 数据集合相关联的操作集
- 抽象
  - 描述数据类型的方法不依赖于具体实现

只描述数据对象集和相关操作集“是什么”，并不涉及“如何做到”的问题

这个点让我想到了《用python学透线性代数和微积分》一书里提到的“基类”，只规定有什么特征，但具体的实现在类的继承中完成

2. 什么是算法
    1. 算法（Algorithm）
        1. 一个有限指令集
        2. 接受一些输入（有时候不需要）
        3. 产生输出
        4. 一定在有限步骤之后终止
        5. 每一条指令必须
            1. 有充分明确的目标，不可以有歧义
            2. 计算机能处理的范围之内
            3. 描述应该不依赖于任何一种计算机语言以及具体的实现手段
3. 什么是好的算法
    1. 空间复杂度S(n)：根据算法写成的程序在执行时占用存储单元的长度
    2. 时间复杂度T(n)：根据算法写成的程序在执行时占耗费时间的长度

- 在分析一般算法的效率时，我们经常关注下面两种复杂度
  - 最坏情况复杂度 $T_{worst}(n)$
  - 平均复杂度 $T_{avg}(n)$

- $T(n) = O(f(n))$ 表示存在常数$C>0, n_0>0$ 使得当$n \geq n_0$ 时有$T(n) \leq C \cdot f(n)$
- $T(n) = \Omega(g(n))$ 表示存在常数$C>0, n_0>0$ 使得当$n \geq n_0$ 时有$T(n) \geq C \cdot g(n)$

- O和Ω是函数的上界和下界
- 上下界都不唯一，所以一般O取最小

输入规模 n

函数	1	2	4	8	16	32
1	1	1	1	1	1	1
$\log n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40326	2092278988000	26313×10^{33}

复杂度分析小窍门

- 若两段算法分别有复杂度$T_1(n) = O(f_1(n))$ 和$T_2(n) = O(f_2(n))$，则
  - $T_1(n) + T_2(n) = \max(O(f_1(n)), O(f_2(n)))$
  - $T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$
- 若$T(n)$是关于n的k阶多项式，那么$T(n)=\Theta(n^k)$
- 一个`for`循环的时间复杂度等于循环次数乘以循环体代码的复杂度
- `if-else` 结构的复杂度取决于`if`的条件判断复杂度和两个分枝部分的复杂度，总体复杂度取三者中最大