# 计算概论 C（2024 春）期末复习

## 考试要求

| | | | |
|---|---|---|---|
| 计算机组成原理：5% | | 组合类型 25%； | 函数、算法 20% |
| 基础语法 10%； | 选择循环 25% | 文件：5%； | 正则：5%　Excel 5% |

## 计算机组成原理

**发展**　图灵（可计算模型、图灵机）—冯诺依曼（二进制 结构）—ENIAC

**硬件**　**冯诺依曼结构**　**影响性能**

输入：键盘、鼠标

主机：CPU（控制器+运算器）、存储器（内存 RAM +外存 硬盘）

**CPU:** 字长（同时处理的二进制位数）、主频（秒内工作周期数）

运算器的核心是 算术逻辑单元 ALU (Arithmetic Logic Unit)

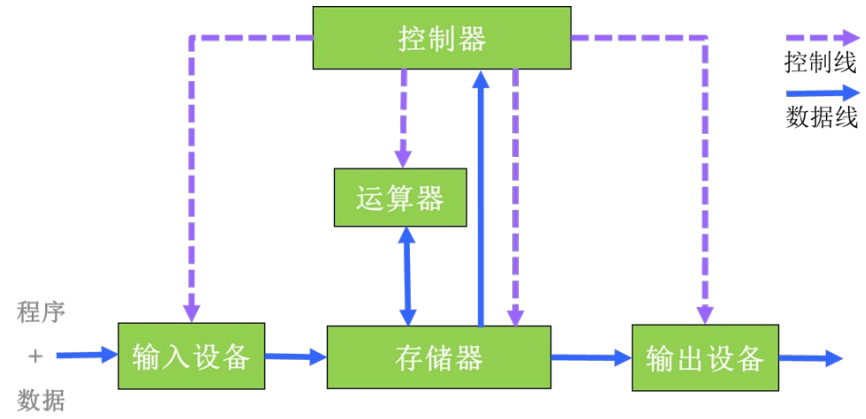
运算速度:　每秒执行的指令条数

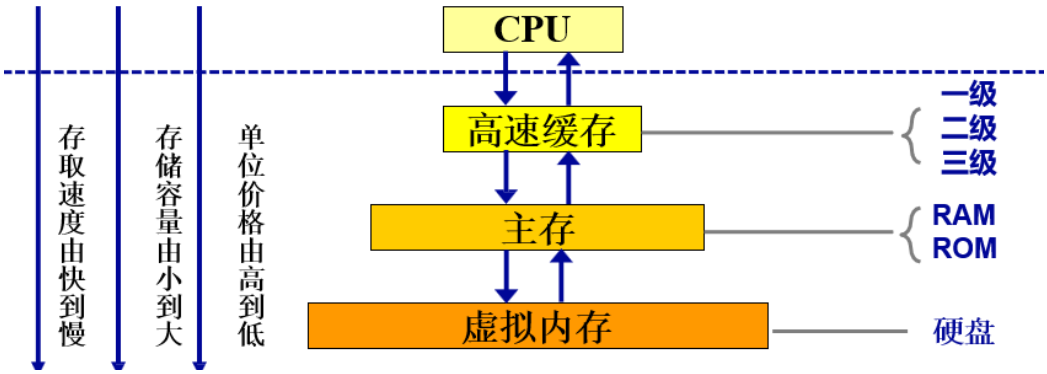
**内存/外存:** 速度、容量

存储单位:　bit, byte（字节=8bit）, KB, MB, GB, TB, PB, ...(1024)

虚拟内存技术　利用计算机的硬盘来模拟内存

输出：显示器（分辨率 刷新频率 响应时间）、打印机




**摩尔定律：** 器件发展遵循 半导体芯片密度每两年增加 1 倍 的规律

**网络**

发展　ARPAnet, WWW, CERnet；依赖于统一的 TCP/IP 协议
IP 地址　IPv4 采用 32 位二进制，IPv6 采用 128 位，DHCP 服务器动态匹配
WLAN　无线网卡+接入设备（无线接入控制器 AP）
　　无线网络标识 SSID 可对应多个 IP
DNS　DNS 解析即将域名(如 its.pku.edu.cn)转化为 IP，请求指定资源
WWW　支持超文本传输协议 HTTP，超链接 HyperLink，URL(协议://主机域名路径文件名)，内容以 HTML 形式显示

**软件**

系统软件：以操作软件、程序设计语言处理程序为代表
　操作系统　DOS; UNIX, LINUX；Windows, Macintosh; Android、iOS
　　控制面板、任务管理器、资源管理器
　文件管理　资源管理器；命名=文件名+扩展名；目录；绝对/相对路径
应用软件：通用软件、专用软件

**CPU 指令系统**

CPU 指令（操作码+地址码[数/位置]）— 程序 stored in 存储器 — 读取

**二进制**

正整数：用原码表示　$(100)_{10} = (00000000\ 01100100)_2$
负整数：补码表示法 = 取反码+1
小数：　存在精度误差
文字：　ASCII 码　ASCII—控制字符—显示字符[西文]，0-9, a-z 连续编码
　　GB　汉语，多字节编码 区位码
　　Unicode　当前计算机通用，适用于大多数文字
　　UTF-8　与 ASCII 兼容，与 GB 不兼容需转换

**高级程序设计语言**

C/C++, Java, Python, C#, VB, JavaScript, PHP, R, SQL
汇编：汇编语言源程序—汇编程序—可执行程序
编译：编译型~，高级语言—编译器—机器语言，如 C, 比↓快
解释：解释型~，源程序—解释器，如 Python

## Excel

文本类型：使用'指示之后的输入为文本
序列填充：右下角+自动填充，行/列 等差/等比/日期 步长可调
冻结窗格：e.g., 冻结标题行，使其在数据滚动过程中始终可见
公式：以=开始，　*区域引用、*函数、*文本连接符(&)
　引用 连续引用用:；分离引用用,
　　相对引用　A1:B5
　　绝对引用　$A$1:$B$5
　　混合引用　A$1, $A2:$A10
　函数 SUM, MAX, MIN, ...
　　IF(A, x0, x1)满足 A 返回 x0，不然返回 x1
　　SUMIF( )求和, COUNTIF( )求个数

LEN($), LEFT($, 5), MID($, start, numbers), RIGHT($, 3)
LOOKUP(key, A2:A14, B2:B14)在 A2-A14 找 key 返回对应 B 值
INDEX(B12:B26, RANDBETWEEN(1,16) )从 B12-B16 中随机选值
RANK(J2, J2:J21, 0)按照降序排列查 J2 在 J2:J21 中的位序
RAND()产生一个 0-1 间的随机数，0-10 采用 RAND()*10
RANDBETWEEN(a, b)产生 a-b 间的随机整数

常见报错类型

| 错误值 | 错误值出现的原因 | 举例说明 |
|---|---|---|
| #DIV/0! | 被零除；除数为0 | =3/0 |
| #N/A | 引用了无法使用的数值 | |
| #NAME? | 无效名称；不能识别的名字 | =SUN(A1:A4) |
| #NULL! | 交集为空 | =SUM(A1:A3 B1:B3) |
| #NUM! | 数据类型不正确 | =SQRT(-4) |
| #REF! | 引用无效单元格 | 引用单元格被删除 |
| #VALUE! | 值错误；不正确的参数或运算符 | =1+"a" |
| ######## | 宽度不够，加宽即可 | |

数据透视表
排序：右键>单列升/降序排列，多列多重条件排序
筛选：右键>数字、文本颜色、颜色、图标
条件格式：开始>样式>条件格式
数据验证：数据>验证数值类型及范围、数据长度、数据唯一性…

# Python

## 基本运算

//：　整除除法（向下取整），如 3//2 = 1, -7 // 2 = -4
%：　取余运算，如 3 % 2 = 1
**：　幂运算，如 2 ** 3 = 8
内置：　max, min, sum, round, abs
调用：　import math; ceil(2.8)=3, floor, factorial(阶乘)

## 数据类型

type()：　获取数据类型，如 a = 1, type(a) = <class: "int">
int()：　向下取整

### 1. 输入

```
a, b, c = input().split(" ")    # a b c
a, b, c = map(int, input().split(" "))
input = list(map(int, input().split()))   # 存入列表
```

### 2. 字符串索引/切片/求值

```
n = "python"
print(n[-1])        # n
print(n[0])         # p
```

```
print(n[::])        # python   [<start>:<end>:step]
print(n[::-2])      # nhy
print(n[1:3])       # yt
s_in = input()      # 2**10
print(eval(s_in))   # 1024
```

3. 输出格式化

```
print("hello" + "world")        # helloworld
print("hello" * 3)              # hellohellohello
print("hello" += "world")       # helloworld
print("x", "y", sep = ",")      # x, y, z
print(1, 2, 3, end = " ")       # 1 2 3(不换行)
print("{:.4f}".format(1.234))   # 1.2345
```

4. 输出转义符：\n (换行符)\t (Tab 制表符)\”（引号）\\(斜杠)

```
print("\"Hello\t world!\"")     # "Hello   world!"
print(3,4,sep="\n")             # 3（换行）4
```

## 条件分支语句

1. If – elif – else 结构

```
if x >= 1:
    print("x is greater than 1")
elif x < 0:
    print("x is negative")
else:
    print("x is between 0 and 1")
```

## 循环控制结构

1. For 循环

```
for 循环变量 in 循环序列:
    语句块（循环体）
```

continue: 跳过单项

```
for letter in "python":
    if letter=='h':
        continue
    print(letter+' ',end='')     # p y t o n
```

break: 自某项终止循环

```
for letter in "python":
    if letter=='h':
        break
    print(letter+' ',end='')     # p y t
```

range(): 注意取值区间左闭右开！

```
for i in range(1,10,2):          # 初值:终值:步长
    print(str(i)+' ',end='')     # 1 3 5 7 9
```

format()的另一种使用：

```
for i in range(1,4):
    for j in range(1,i+2):
        print("({},{}) ".format(i,j),end='')
    print()                          # 结果按 i 换行
```

for…else…语句

```
for i in range(5):          # for i in "python"也适用
    print(i)
else:
    print(str(i)+'end')    # 正常循环(break)后执行
```

2. **While 循环**

```
while 条件表达式:          # while True + break 实现循环
    语句块（循环体）
```

continue + +=: 跳过单项

```
num=1
while num<6:
    if num==4:
        num+=1
        continue
    print(str(num)+' ',end='')
    num+=1
print("end")       # 1 2 3 5 end
```

break: 自某项终止

```
num=1
while num<6:
    if num==4:
        break
    print(str(num)+' ',end='')
    num+=1
print("end")       # 1 2 3 end
```

while…else…语句

```
num=1
while num<5:
    print(num)
    num+=1
else:                           # // = 换行
    print(str(num)+' end')     # 1//2//3//4//5 end
```

## 组合数据类型

1. **列表 lst = [1, 2, 3, 4, 5]**

索引 注意 1) 第一项从 0 开始，2) 可以使用负数，-n 指倒数第 n 项

长度 采用 len(lst)

加法 result = lst + lst + lst  # [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

乘法 result = lst * 3 # [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
切片 注意第一项从 0 开始、左闭右开

```
lst = [0,1,2,3]      #[<start>:<end>:step]
slice = lst[::-1]    # [3, 2, 1, 0]= lst.reverse()
```

插入 lst.append 在列表末加入元素，lst.insert(position, value)在某位置加入元素
增加 lst_1 = [], lst.extend(lst_1) 向 lst 加入 lst_1 中元素
删除 lst.pop(position), lst.remove(value), del lst[position], lst.clear( )清空
计算 max(), min(), sum(), lst.count(value)=times
排序 **lst.sort()**

```
lst.sort(key = lambda x: (x[0], x[1])) 或
lst = sorted(lst, key = lambda x: (x[0],x[1]))
```

举例 输出正整数的所有因数

```
n = int(input())  # 374
lst = [x for x in range(1, n+1) if n % x == 0]
print(lst)     # [1, 2, 11, 17, 22, 34, 187, 374]
```

**2. 元组 tpl= (0, 1, 2, 3)**
一旦创建即不可修改版的列表

**3. 集合 st= {0, 1, 2, 3} = set([0, 1, 2, 3])**
并集 st_add = st | {4, 5} # {0, 1, 2, 3, 4, 5}
交集 st_mul = st & {2, 3, 4} # {2, 3}
遍历 for i in st 循环，无法使用索引
插入 st.add( )
增加 st_1 = {4, 5} st.update(st_1)
删除 st.remove(item), st.discard(item), st.pop(position)删除并返回，st.clear()清空
去重 由于集合无重复元素，可使用 lst = list(set())为列表去重

**4. 字典 dit = dict([('jack', 4098), ('sape', 4139)])**
插入 dit['silver'] = 1000，dit.update(dit2)
删除 del dit['silver'], dit.pop('silver')
取键 dit.keys，也可直接 for keys in dit
取值 dit.values()
取对 dit.items()
删除 dit.pop()删除并返回对应值，dit.clear()清空
复制 dit_copy = dit.copy() = dict(dit)
统计 统计字符串/字典中每个字符出现的次数

```
s = 'abacdd'
dit = {}
for item in s:
    if dit.get(item) == None:
        dit[item] = 0
    dit[item] += 1
print(dit) # {'a': 2, 'b': 1, 'c': 1, 'd': 2}
```

join() 转化数据表现形式

```
lst = ["a","b","c","d"]
```

```
result = ','.join(lst)
print(result)   # a,b,c,d
```

排序 默认先按照第一个元素排，等同则第二个元素，以此类推

```
dit_sorted = sorted(dit.items(), key = lambda x: (-x[1], x[0]))
```

## 函数与算法

### 斐波那契数列

```
def fibonacci (n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci (n-1) + fibonacci(n-2)
n = int(input())
for i in range(n):
    print(i , ":", fibonacci(i), sep = "")
```

**变量** 局部变量（函数内），全局变量（函数外）

```
def func1(a, b):
    n = a+b      # 局部变量
    global n     # 声明 n 为全局变量
n = 0            # 全局变量
```

**递归算法** 定义函数时引用自身

```
def factorial(n):
    if n == 0:                  # 终止条件
        return 1
    else:                       # 递归条件
        return n * factorial(n-1)
```

## 其他

### 条件句

```
n.isalpha()    # 条件句：n 是字母
n.isdigit()    # 条件句：n 是数字
n.islower()    # 条件句：n 是小写字母
n.is_integer() # 条件句：n 是整数
```

**辗转相除法** 计算 m, n 的最大公约数

```
while m != 0:
    r = n % m
    n = m
    m = r
print(n)
```

**生成随机数** random

```
import random
random.randint(1, 10000) # 274
random.random(0, 1)    # 0.134
```

# 文件处理

**打开文件**　f = open("title\\pathway", mode = "rt", encoding = None)

**mode:**

| | |
|---|---|
| 'r' | 只读模式，如果文件不存在，返回异常FileNotFoundError，默认值 |
| 'w' | 覆盖写模式，文件不存在则创建，存在则完全覆盖原文件 |
| 'x' | 创建写模式，文件不存在则创建，存在则返回异常FileExistsError |
| 'a' | 追加写模式，文件不存在则创建，存在则在原文件最后追加内容 |
| 'b' | 二进制文件模式 |
| 't' | 文本文件模式，默认值 |
| '+' | 与r/w/x/a一同使用，在原功能基础上增加同时读写功能 |

**encoding:**

None 意味着与系统的字符编码格式相同，e.g., 使用中文

```
with open("title", mode="rt", encoding="GBK") as f:
    content = f.read()
    print(content)      # 输出文件内容
```

**读出文件**

<title>.read(-1)　　size=-1 读取整个文件，=k 指读取前 k 字节

<title>.readline(-1)　　size=-1 读取第一行的全部，=k 读取那一行前 k 字节

<title>.readlines(-1)　　size=-1 读取全部行，=k 读取前 k 行

**文件处理**

```
try:
    f1 = open("title", "rt")
    for line in f:
         line = line.replace("\n", "")
except (ValueError, NameError):
    print("Value/Name Error")
except FileNotFoundError:
    print("FileNotFound, Error")
except Exception as err:        # 其他异常
    print(err)
    if f2:  # 如果文件已打开，关闭文件
        f2.close()
        print("close file2 before exit!")
finally:
    print("Exit at last!")
```

**写入文件**

<title>.write(s)　　s 是写在原文件最后的字符串

<title>.writelines(lines)　添加 lines[列表]至原文件后，如需换行 lines 后加"\n"

*Recall* \n (换行符)\t (Tab 制表符)\"（引号）\\(斜杠)

**关闭文件**

f.close()

# 正则

字符串的查找、替换

```
"bc" in "abcd"                    # True
"abcdjsofub".find("djso")          # 3
"abcdjsofub".find("x")             # -1
"abc12bdf".index("12")             # 3
"abcd12bdf".index("jul")           # ValueError
"abcsabdf".count("ab")             # 2
"abcsabdf".replace("ab", "*")      # *cs*df
"abcsabdf".replace("ab", "*", 1)   # *csabdf
```

Raw 格式字符串

```
print(r"Hello\n world!")  # Hello\n world
```

字符编码 – 编码字符互转

```
ord("a")  # 97
ord("燕") # 29141
chr(122)  # 'z'
chr(38364)  # "雪"
```

大小写互转

```
"aBC".lower()      # abc
"foureh".upper()   # FOUREH
```

去掉字符串两端的指定字符

```
"   aB cdef   ".strip()      # aB cdef
"###aB cdef###".lstrip("#")  # aB cdef###
"###aB cdef###".rstrip("#")  # ###aB cdef
```

## 正则表达式

| | | |
|---|---|---|
| d | 数字 | D 为非数字 |
| w | 单词字符（大小写字母和数字） | W 为非单词字符 |
| . | 单个字符 | 代表除换行符以外的任意单个字符。例如'a.c'可代表'abc'、'acc'等，但不能代表'abbc' |
| ? | 多个字符，非贪婪模式符 | 匹配前一个字符 0 次或 1 次 |
| * | 多个字符 | 匹配前一个字符 0 次或无数次 |
| + | 多个字符 | 匹配前一个字符 1 次或无数次 |
| \| | 或 | 分隔字符之间“或”的逻辑关系，例如'[P\|p]ython'能匹配出'Python'或'python' |
| ^ | 开始 | 引导字符串开始的特征 **确定开头是否符合正则字符串** |
| $ | 结尾 | 引导字符串结尾的特征 **确定结尾是否符合正则字符串** |
| \ | 转义 | 为其后面的符号转义，但为避免与 Python 字符串本身的转义相混淆，建议正则表达式以 r 前缀统一转义。例如'\d'可表示为 r'd' |
| [] | 界定单个字符 | |
| () | 界定一个整体 | |
| {} | 重复次数 | |

确定输入是否符合某格式

```
import re
pattern = r'^\d{3}-\d{2}-\d{4}$'
string = '123-45-6789'
match = re.match(pattern, string)
if match:
    print("SUCCESS")
else:
    print("FAILURE")
```

查找是否存在某格式的字符

```
import re
pattern = r'\d{3}-\d{2}-\d{4}'
string = 'My number is 123-45-6789'
search = re.search(pattern, string)
if search:
    print("Found it")
else:
    print("None")
```

查找符合某格式的所有字符，返回的是**列表**！

```
import re
pattern = r'\d{3}-\d{2}-\d{4}'
string = 'My numbers are 123-45-6789 & 987-65-4321'
matches = re.findall(pattern, string)
print(matches)    # ['123-45-6789', '987-65-4321']
```

查找符合某格式的所有字符，替换为指定字符

```
import re
pattern = r'\d{3}-\d{2}-\d{4}'
replacement = 'XXX'
string = 'My numbers are 123-45-6789 & 987-65-4321'
result = re.sub(pattern, replacement, string)
print(result)  # My numbers are XXX & XXX.
```

通过正则匹配拆分字符串，返回**列表**！

```
import re
pattern = r'\s+'                        # s = space
string = 'This is    a    test'
result = re.split(pattern, string)
print(result)         # ['This', 'is', 'a', 'test']
```

匹配方式

```
import re
pattern = r'[0-9a-zA-Z\-_]'   # 匹配任何数字 字母 - _
string = 'hi-123'
result = re.findall(pattern, string)
print(result)       # ['h', 'i', '-', '1', '2', '3']
```

\d{3,8}即 3-8 位数字

```
import re
pattern = r'\d{3}\s+\d{3,8}'
string = 'My number is 173 21206602'
result = re.findall(pattern, string)
print(result)                    # ['173 21206602']
```

```
import re
pattern = r'([0-9]{1,3}\.[a-z])'
string = '21974462.abaxiudhf'
result = re.findall(pattern, string)
print(result)                    # ['462.a']
```

\b 指在单词边界处匹配

```
import re
pattern = r'\b[_]\w'        # 匹配单词开头的“_字母”
string = 'test123, _hi, hello_bye'
result = re.findall(pattern, string)
print(result)                                # ['_h']
```

()子表达式分组匹配

```
import re
pattern = r'(\d+)/(\d+)/(\d+)'
string = 'Today is 12//07/2021. Next 1/1/2022.'
result = re.findall(pattern, string)
print(result)                 # [('1', '1', '2022')]
```

|表示或者，即匹配其中之一

```
import re
pattern = r'h[1-6]|[a-z]'
string = 'h1'
result = re.findall(pattern, string)
print(result)            # ["h1"]
```

groups()/group()返回正则匹配的字符串，仅匹配 r“()”内，(0)返回所有字符串

```
import re
text = "Today's date is 2024-06-03."
pattern = r"(\d{4})-(\d{2})-(\d{2})"
match = re.search(pattern, text)
if match:
    year = match.group(1)
    month = match.group(2)
    print(f"{year},{month}")  # 2024,06
```

```
import re
text = "Today's date is 2024-06-03."
pattern = r"(\d{4})-(\d{2})-(\d{2})"
match = re.search(pattern, text)
print(match.groups())     # ('2024', '06', '03')
```

start()/end() 返回正则匹配的字符串的起始/终止位置，(0)返回所有字符串

```
import re
text = "Today's date is 2024-06-03."
pattern = r"(\d{4})-(\d{2})-(\d{2})"
match = re.search(pattern, text)
if match:
    start_position = match.start()        # 匹配的起始位置，16
    start_year = match.start(1)    # 第 1 个捕获组的起始位置，16
    start_month = match.start(2)   # 第 2 个捕获组的起始位置，21
    start_day = match.start(3)     # 第 3 个捕获组的起始位置，24
```

span() 返回正则匹配字符串的长度，(0)返回所有字符串

```
import re
text = "Today's date is 2024-06-03."
pattern = r"(\d{4})-(\d{2})-(\d{2})"
match = re.search(pattern, text)
if match:
    start_position = match.span()   # (16, 26)
    start_year = match.span(1)      # (16, 20)
    start_month = match.span(2)     # (21, 23)
    start_day = match.span(3)       # (24, 26)
```

re.findall(pattern, text, flags=)

flags=re.I　　要求匹配时忽略大小写

```
import re
text = "Apple Banana apple banana"
pattern = r"apple"
matches = re.findall(pattern, text, flags=re.I)
print(matches)              # ['Apple', 'apple']
```

flags=re.M　　使^和$作用于每行的开始和结尾

```
import re
text = """1. First line
2. Second line
3. Third line
"""
pattern = r"^\d+"
matches = re.findall(pattern, text, flags=re.M)
print(matches)                # ['1', '2', '3']
```

flag=re.S　　使.匹配包含换行符在内的所有字符

```
import re
text = """Line 1: Hello
Line 2: World"""
pattern = r"Hello.+World"
matches = re.findall(pattern, text, flags=re.S)
print(matches)      # ['Hello\nLine 2: World']
```

flag=re.X　　忽略空格和#后的注释

```
import re
pattern = r"""
    ^               # 匹配字符串的开头
    [\w\.-]+        # 匹配用户名部分
    @               # 匹配@
    [\w\.-]+        # 匹配域名部分
    \.              # 匹配.
    [a-zA-Z]{2,}    # 匹配两位及以上字母
    $               # 匹配字符串的结尾
"""
text = "My email is test@example.com"
matches = re.findall(pattern, text, flags=re.X)
print(matches)          # ['test@example.com']
```

贪婪匹配与非贪婪匹配

贪婪匹配　　*, +, ?　　匹配尽可能多的字符

```
import re
pattern = r"[0-9]+"
text = "abc123456d1"
matches = re.findall(pattern, text)
print(matches)                  # ['123456', '1']
```

非贪婪匹配　*?, +?, ??　　匹配尽可能小的单位

```
import re
pattern = r"[0-9]+?"
text = "abc156d1"
matches = re.findall(pattern, text)
print(matches)           # ['1', '5', '6', '1']
```