

目录

第 4 章：MATLAB 程序流程控制	2
4.1 条件语句	2
4.1.1 if-elseif-else-end 语句.....	2
4.1.2 switch-case-otherwise-end 语句	7
4.2 循环语句	10
4.2.1 for-end 语句	10
4.2.2 while-end 语句.....	13
4.2.3 break 和 continue	15
4.3 处理异常的 try-catch 语句	19
4.4 控制程序流程的其他常用指令	20
4.4.1 程序运行计时：tic/toc	21
4.4.2 暂停程序运行：pause	22
4.4.3 用户输入数据：input	22
4.4.4 显示警告信息：warning	24
4.4.5 显示报错信息：error.....	24
4.5 本章小节	25
4.6 课后习题	25

讲解视频：可以在 bilibili 搜索 “MATLAB 教程新手入门篇——数学建模清风主讲”。

<https://www.bilibili.com/video/BV1dN4y1Q7Kt/>

配套的讲义和代码下载方式：

微信公众号《数学建模学习交流》后台发送 701365 六个数字

第 4 章：MATLAB 程序流程控制

程序流程控制是计算机程序中用于管理和控制程序执行顺序的一种技术。前面章节我们编写的程序都是按照从上至下的顺序依次执行的，这种结构叫做顺序结构。本章开头将介绍程序设计中另外两种非常重要的结构：分支结构和循环结构，它们是通过条件语句和循环语句实现的；此外，本章还会介绍 `try-catch` 语句，它能对代码中的错误进行异常捕获；最后，本章也会介绍一些控制程序流程的其他常用指令，例如程序计时、暂停程序、输入数据等。

4.1 条件语句

在编写程序时，有时我们需要根据不同的条件执行不同的命令。举个例子，b 站上有许多电影需要开通会员后才能观看完整的内容。在这种情况下，b 站后台会自动检查观众是否是会员用户，如果是会员，则允许他们观看整个电影，否则只允许试看几分钟。这个过程可以用右侧的流程图来表示，图上有左右两个分支。



在 MATLAB 中，我们将介绍两种条件语句：**if-elseif-else-end 语句** 和 **switch-case-otherwise-end 语句**，它们可实现分支结构。其中前者更为通用，能够满足多种条件判断的需求，因此常用于实际编程中。无论使用哪种条件语句，其核心思想都是一样的：首先判断给定的条件是否满足，然后根据判断结果（true 或 false）执行相应的命令。

因此，条件语句也可以被称为条件判断语句或判断语句，它们在程序中扮演着至关重要的角色，帮助我们实现不同情况下程序的灵活控制和处理。

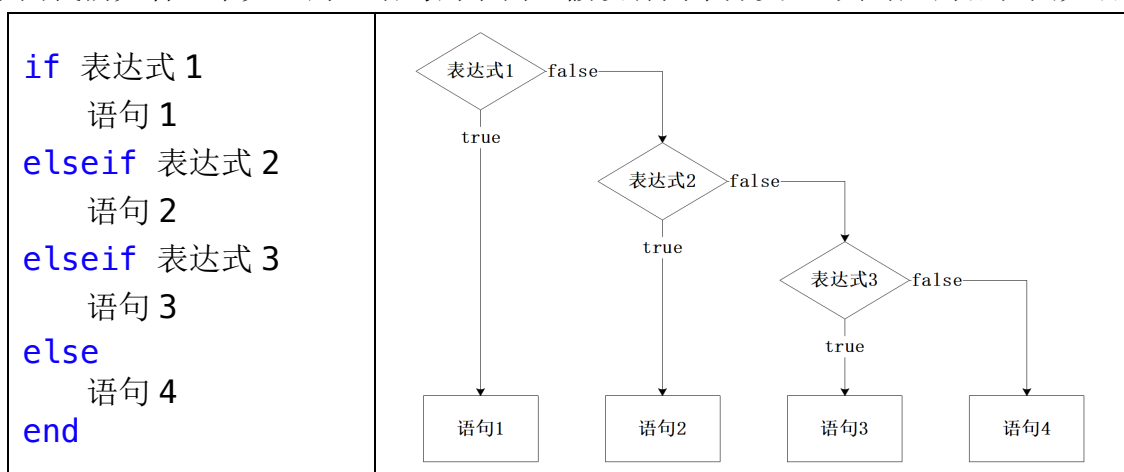
4.1.1 if-elseif-else-end 语句

if、elseif、else 和 end 是 MATLAB 中的四个关键字，在第二章中我们强调过：不能定义与 MATLAB 关键字同名的变量。这四个关键字就构成了我们要学的第一种条件语句，后续我们将其简称为 **if 语句**。

易错点：elseif 关键字中间不能加空格，不能写成 else if，这和 C、Java 等语言不同。

注意：在使用 if 语句时，**if 和 end 这两个关键字是无论如何都不能省略的**。有很多新手在使用 MATLAB 时都会漏掉 end 这个关键字，导致后续无论输入什么命令 MATLAB 都没有反应，而 elseif 和 else 可以根据自己的需要来决定是否添加。

下面我们先看一个完整的 if 语句的示例（假设有四个分支），其对应的流程图见右侧：

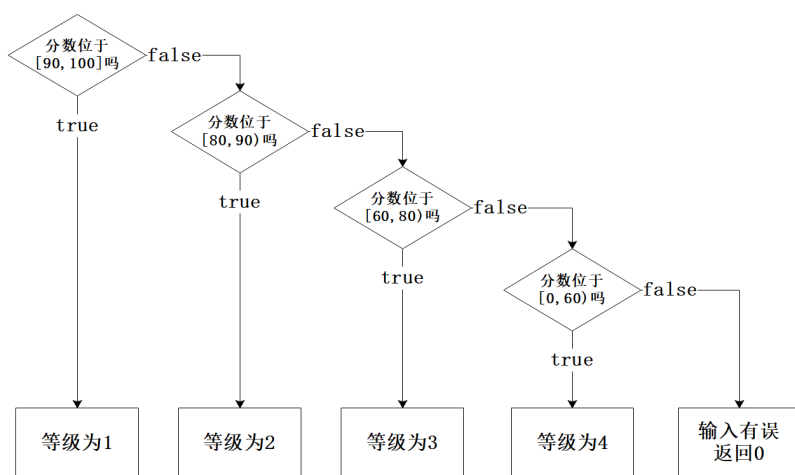


注意，这里的语句 1、语句 2、语句 3 和语句 4 最终只能有一个被执行。如果表达式 1 成立（通常表达式返回一个逻辑值，值为逻辑值 1 则表示成立），则执行语句 1，后面的表达式都不用判断，它们对应的语句也都不会被执行；如果表达式 1 不成立，则会接着判断表达式 2 是否成立，如果表达式 2 成立则会执行语句 2，否则会接着判断表达式 3 是否成立，如果表达式 3 成立则会执行语句 3，如果表达式 3 也不成立则会执行语句 4。

我们举一个具体的例子：计算分段函数 $y = \begin{cases} 1, & 90 \leq x \leq 100 \\ 2, & 80 \leq x < 90 \\ 3, & 60 \leq x < 80 \\ 4, & 0 \leq x < 60 \\ 0, & x \text{ 取其他值} \end{cases}$ ，你可以将这个分段函数

想象成如下场景：给定一个同学的成绩（假设为整数），输出这个同学的等级。等级规则如下：90 至 100 分为 1 级、80 至 89 分为 2 级、60 至 79 分为 3 级、低于 60 分为 4 级；如果成绩小于 0 分或者大于 100 分，则代表成绩输入有误，此时等级为 0。

那么，应该怎样用 MATLAB 表示这个分段函数呢？不熟练的同学可以先画一个流程图，其对应的 MATLAB 代码如右侧所示：



```

x = 88; % x表示成绩
if x >= 90 && x <= 100
    dj = 1; % 等级为1级
elseif x >= 80 && x < 90
    dj = 2; % 等级为2级
elseif x >= 60 && x < 80
    dj = 3; % 等级为3级
elseif x >= 0 && x < 60
    dj = 4; % 等级为4级
else
    dj = 0; % 输入有误
end
dj

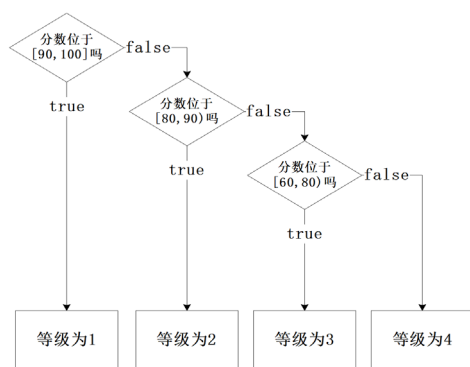
```

dj =
2

上面代码中有一些要说明的点：

- (1) 判断 x 是否位于 $[90, 100]$ 这个区间时，不能写成 $90 \leq x \leq 100$ ，MATLAB 不支持这种连续的判断。因此我们将其拆分成两个条件： $x \geq 90 \ \&\& \ x \leq 100$ 。这里用的是具有短路功能的逻辑与运算符 $\&\&$ ，这样判断效率会更高。
- (2) `if` 只能有一个，但 `elseif` 可以有多个，它们后面都需要跟上相应的判断条件。只有在 `if` 后面的条件不满足时，才会判断 `elseif` 后面的条件是否成立。
- (3) `else` 后面不能加上条件，当 `if` 和 `elseif` 后面的条件全部都不满足时，才会执行 `else` 对应的语句。
- (4) 不需要在 `if`、`elseif`、`else` 和 `end` 所在的行的最后面添加冒号或者分号。

另外，假设我们不考虑输入有误的情况，即确保 x 的取值范围在 0 至 100 之间时，我们可以对 `if` 和 `elseif` 后面的判断条件进行简化。显然，可以去掉 `if` 后面的 $x \leq 100$ ，因为 x 的取值范围已经固定了；`elseif` 后面的条件也能被简化，以第一个为例，原来代码中的判断条件写的是 $x \geq 80 \ \&\& \ x < 90$ ，实际上我们可以直接写成 $x \geq 80$ ，这是因为只有当前面的判断条件都不成立时才会到达当前的判断条件，而前面的判断条件不成立时意味着 $x < 90$ 。类似地，第二个 `elseif` 后面的条件都能简化成 $x \geq 60$ ，下面给出简化后的代码：



```

x = 68; % x表示成绩
if x >= 90 && x <= 100
    dj = 1; % 等级为1级
elseif x >= 80 && x < 90
    dj = 2; % 等级为2级
elseif x >= 60 && x < 80
    dj = 3; % 等级为3级
else
    dj = 4; % 等级为4级
end
dj

```

dj = 3

```

x = 68; % x表示成绩
if x >= 90
    dj = 1; % 等级为1级
elseif x >= 80
    dj = 2; % 等级为2级
elseif x >= 60
    dj = 3; % 等级为3级
else
    dj = 4; % 等级为4级
end
dj

```

dj = 3

前文提到过：使用 if 语句时，if 和 end 这两个关键字是无论如何都不能省略的，而 elseif 和 else 可以根据自己的编程需要来决定是否添加。下面我们来举两个例子：

(1) 给定包含 2 个元素的行向量 a，请使用 if 语句对向量 a 中的元素按照从小到大的顺序排序，并将排序后的向量保存为变量 sort_a。例如 a = [5,3]，那么 sort_a = [3,5]。

我们只需要判断一次：如果 a 中第一个位置的元素比第二个位置的元素大，那么就调换它们的位置并将结果赋值给 sort_a；如果 a 中第一个位置的元素小于等于第二个位置的元素则不用调换，直接将 a 赋值给 sort_a 即可，这时候就不需要用到 elseif 关键字了；当然，如果你非要用 elseif，一定要在后面跟上条件，这时候可以省去 else 关键字，这样做的优点是让判断条件更加清楚，缺点是要多敲一点代码。

```

a = [5,3];
if a(1) > a(2)
    sort_a = [a(2),a(1)];
else
    sort_a = a;
end
sort_a

```

```

sort_a =
     3     5

```

```

a = [5,3];
if a(1) > a(2)
    sort_a = [a(2),a(1)];
elseif a(1) <= a(2)
    sort_a = a;
end
sort_a

```

```

sort_a =
     3     5

```

如果你的编程思维很清晰的话，使用左侧的代码会更加简洁，更推荐大家用左侧的代码。

(2) 给定一个包含 2 个元素的行向量 a，请对向量 a 中的元素按照从小到大的顺序排序，将排序后的结果直接赋值到 a 上，不需要使用新的向量保存排序结果。

此时我们可以省去 else：如果 a 中第一个位置的元素比第二个位置的元素大，那么就调换它们的位置并将结果重新赋值给 a；否则说明不用调换，a 还是原来的，即 if 后面的条件不成立时，我们什么事情都不用做，else 可以不写。

```

a = [5,3];
if a(1) > a(2)
    a = [a(2),a(1)];
end
a

```

```

a =
     3     5

```

下面我们专门介绍一下 `if` 和 `elseif` 后面的表达式，在前面的例题中，表达式实际上就是判断条件，该判断条件通常是第三章学过的关系运算(例如`==`、`~=`、`>=`)和逻辑运算(例如`&`、`&&`、`||`、`|`、`~`)，因此运算结果要么是逻辑值 1(true)，要么是逻辑值 0(false)。例如 `a(1)>a(2)`、`x>=80 && x<90`。事实上，`if` 和 `elseif` 后面的表达式也支持其他运算，例如算术运算，其计算结果可以是一个数值常数，不一定非要是逻辑值 1 或者 0。为什么可以这样？第三章介绍逻辑运算时讲过：**MATLAB 会将非零数值视为逻辑 1，将数值零视为逻辑 0 进行运算。**例如：`3&5` 返回逻辑 1，`-4&0` 返回逻辑 0。这里也是类似的：**如果 `if` 和 `elseif` 后面表达式的计算结果为非零数值，就会被当成逻辑值 1(true)；如果计算结果为数值零，则会被当成逻辑值 0(false)。**当然，我们在实际编程时很少用到这种情况，大家只需要了解背后的原理即可。

```
if 3+5
    a = 10;
else
    a = 20;
end
a
```

a = 10

```
if 6-6
    a = 10;
else
    a = 20;
end
a
```

a = 20

另外有同学可能在 `if` 或 `elseif` 后面放上了一个数值向量或者矩阵，这时候只有当这个向量或者矩阵中的所有元素都是非零元素时，才会被当成逻辑值 1(true)。

```
x = [1 2;0 3];
if x
    a = 10;
else
    a = 20;
end
a
```

a = 20

```
x = [1 2;-1 3];
if x
    a = 10;
else
    a = 20;
end
a
```

a = 10

思考题：修改上面的代码使得：当矩阵 `x` 中至少有一个元素为非零元素时，就会被当成逻辑值 1(true)。（答案：将 `if` 后面的 `x` 改成 `any(x(:))` 或者 `any(any(x))` 即可）

小技巧：利用智能缩进整理代码。俗话说人靠衣装马靠鞍，代码也是这样。在编程时，优秀的程序员都会注重代码的布局，即代码的缩进、留白等。好的布局会让代码更容易被理解。细心的同学已经发现了，前面的代码例子中，我在 `if`、`elseif` 和 `else` 下方的语句开头增加了缩进，这些缩进能让代码整体结构更加清晰，如果不加缩进的话是下面左侧的样子。当你觉得你的代码看着很不整齐时，**你可以选中你的代码，然后按快捷键“`ctrl+i`”，这样 MATLAB 会自动帮你整理代码，看起来效果也会好很多。**

```
x = 68; % x表示成绩
if x>=90
dj = 1; % 等级为1级
elseif x>=80
dj = 2; % 等级为2级
elseif x>=60
dj = 3; % 等级为3级
else
dj = 4; % 等级为4级
end
dj
```

不加缩进：

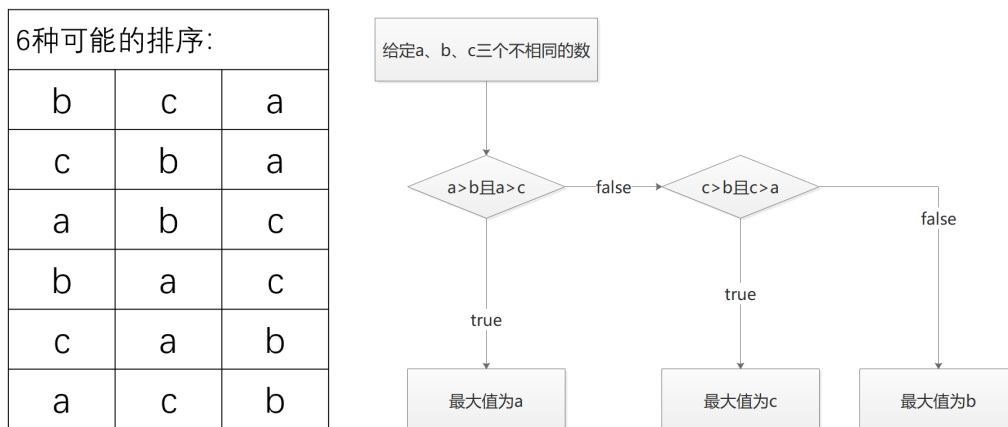
```
x = 68; % x表示成绩
if x>=90
    dj = 1; % 等级为1级
elseif x>=80
    dj = 2; % 等级为2级
elseif x>=60
    dj = 3; % 等级为3级
else
    dj = 4; % 等级为4级
end
dj
```

增加缩进：

本小节最后，我们通过一个经典例题来理解 if 的嵌套，所谓 if 的嵌套，是指在 if、elseif 或者 else 下方的语句中又出现了 if 语句（**一定要注意：每个 if 都要有配套的 end**）。

例题：已知 a 、 b 和 c 是三个互不相等的常数，请使用 if 语句找出 a 、 b 和 c 三个数的最大值。（注意，这里是练习条件语句，请不要使用 max 函数直接求最大值）

如果使用原来的思路，我们需要找到各种可能情况并进行多次判断，流程图和代码如下：

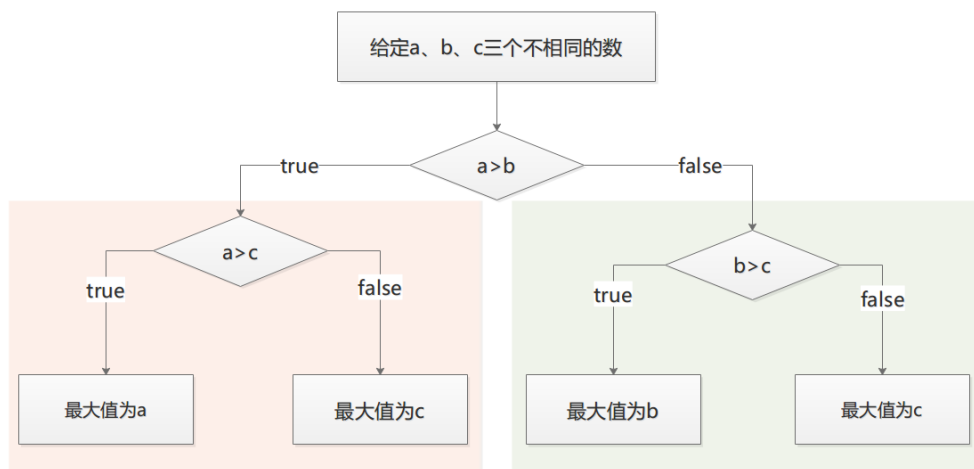


注意： $a>b$ 且 $a>c$ 涵盖了表中前两种排序的情况； $c>b$ 且 $c>a$ 则涵盖了表中第三种和第四种排序的情况；如果都不满足则只剩下表中最后两种排序的情况，此时最大值为 b 。

```

a = 5; b = 8; c = 3; % 随便编一组数据测试
if a > b && a > c
    Max = a; % 不要命名为小写的 max，否则和内置函数重名了
elseif c > b && c > a
    Max = c;
else
    Max = b;
end
Max
  
```

下面我们换一种思路，我们判断的流程图如下：



从上图可以看出：我们首先判断 a 和 b 的大小关系，然后根据第一次的判断结果分别进行条件判断，因此出现了嵌套的情况。根据上述流程图可编写下面的代码：


```

if a>b
    if a>c    % 此时 a>b 且 a>c
        Max=a;
    else    % 此时 a>b 且 c>a
        Max=c;
    end    % a>c 前面的那个 if 配套的 end，通过缩进可以看出
else    % 此时 a<b
    if b>c    % 此时 a<b 且 b>c
        Max=b;
    else    % 此时 a<b 且 b<c
        Max=c;
    end    % b>c 前面的那个 if 配套的 end，通过缩进可以看出
end    % 最上方的 if 配套的 end
Max

```

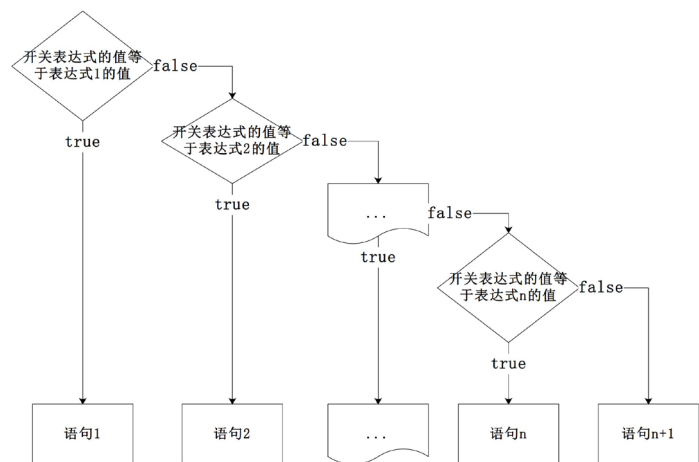
4.1.2 switch-case-otherwise-end 语句

switch 翻译成中文为开关，switch 语句是另一种实现条件语句的方法，其基本用法如下：

```

switch 开关表达式
case 表达式 1
    语句 1
case 表达式 2
    语句 2
...
case 表达式 n
    语句 n
otherwise
    语句 n+1
end

```



switch 语句在运行时，会将开关表达式的值依次和各个 case 后面的表达式的值判断是否相等，如果相等则为 true，此时会执行该 case 中相应的语句；如果不相等则为 false，此时会取下一个 case 后面表达式的值进行比较，直到出现 true 为止。如果所有 case 后面表达式的值均与开关表达式的值不相等，则执行 otherwise 中对应的语句。

注意：开关表达式的计算结果必须是一个数值标量或者是一个字符向量/字符串，不能是向量或者矩阵。若为数值标量，则当 case 后面表达式中的值等于开关表达式的数值标量时为 true；若为字符向量或字符串，则当 case 后面表达式中的字符向量/字符串和开关表达式的字符向量/字符串完全相同时为 true。下面举两个例子帮助大家理解：

```

season = randi([1,4])    % 季节
switch season
case 1
    disp("第一季度为春季");
case 2
    disp("第二季度为夏季");
case 3
    disp("第三季度为秋季");
otherwise
    disp("第四季度为冬季");
end

```

season 是在区间[1,4]上随机生成的一个整数，用来表示季节。switch 后面的开关表达式就是 season 这个数值标量，程序会按照从上到下的顺序依次判断 season 和 case 后面的数值是否相等，若相等则执行对应的语句。若 season 取值为 4，则和 case 后面的数值均不相等，此时程序会执行 otherwise 后面的语句。

注意：MATLAB 中的单引号和双引号表示的文本有一定的区别，使用单引号引起来被称为字符向量，例如'abc'，而双引号引起来的被称为字符串，例如"abc"。

上面代码中我们用到了 `disp` 函数（来自单词 `display`），它可以将文本、数值等显示到 MATLAB 的窗口，`disp` 语句是否以分号结尾不会影响结果的输出。

```
a = randi(10)
b = randi(10)
way = "乘法";
switch way
    case "加法"
        disp(a+b)
    case "减法"
        disp(a-b)
    case "乘法"
        disp(a*b)
    case "除法"
        disp(a/b)
    otherwise
        disp("你的输入有误")
end
```

`a` 和 `b` 是在区间 $[1,10]$ 上随机生成的两个整数，我们希望根据变量 `way` 表示的字符串决定对 `a` 和 `b` 的计算方式。

`switch` 后面的开关表达式就是 `way` 表示的字符串，程序会按照从上到下的顺序依次判断 `way` 和 `case` 后面的字符串是否相同，若相同则执行对应的语句。例如 `way` 等于“乘法”，则程序会先判断第一个 `case`，第一个 `case` 为“加法”，字符串不相同，则会继续判断第二个 `case`，第二个 `case` 的字符串和 `way` 的字符串也不同，则会判断第三个 `case`，此时字符串一致，程序会执行 `disp(a*b)` 这一行语句，假设 `a` 为 5、`b` 为 3，则会在窗口输出 15。

易错点：和 `if` 语句一样，`switch` 语句必须以 `end` 结束，千万不能漏写！

在 `if` 语句中，有时候我们会不写 `elseif` 和 `else` 关键字，而 `switch` 语句也可以不写 `case` 和 `otherwise` 这两个关键字。另外我们也可以将上面两个例子转换成 `if` 语句，因此在实际使用中，`switch` 语句用的频率并不高。下面我们给出转换后的结果：

```
season = randi([1,4]) % 季节
if season == 1
    disp("第一季度为春季")
elseif season == 2
    disp("第二季度为夏季")
elseif season == 3
    disp("第三季度为秋季")
else
    disp("第四季度为冬季")
end
```

```
a = randi(10)
b = randi(10)
way = "乘法";
% 使用 strcmp 函数比较字符串是否相同
if strcmp(way, "加法")
    disp(a + b)
elseif strcmp(way, "减法")
    disp(a - b)
elseif strcmp(way, "乘法")
    disp(a * b)
elseif strcmp(way, "除法")
    disp(a / b)
else
    disp("你的输入有误")
end
```

注意：比较两个字符串是否相同可以使用关系运算符`==`，也可以使用 `strcmp` 函数；而比较两个字符向量是否相同请使用 `strcmp` 函数。关于文本处理的相关知识，我们会在下一章中进行系统的讲解。

易错点：下面代码中 `n` 的值为多少时，MATLAB 会输出成绩合格？

```
n = randi([0,100]) % n 是区间 [0,100] 上的随机整数，表示成绩
switch n
    case n >= 60
        disp("成绩合格")
    otherwise
        disp("成绩不合格")
end
```


注意，`case` 后面 `n>=60` 是一个判断语句，返回的逻辑值要么为逻辑 1，要么为逻辑 0。如果 `n` 等于 0，此时 `n>=60` 的计算结果为逻辑 0，此时 MATLAB 会认为开关表达式对应的 `n` 和 `case` 表达式计算的结果相等，此时会输出成绩合格；当 `n` 取值为 `[1,59]` 时，`n>=60` 的计算结果为逻辑 0，和 `n` 不相等，此时会运行 `otherwise` 中的语句，即输出成绩不合格；当 `n` 取值大于等于 60 时，`n>=60` 的计算结果为逻辑 1，和 `n` 也不相等，此时也会运行 `otherwise` 中的语句，输出成绩不合格。综上所述，当且仅当 `n` 等于 0 时才会输出成绩合格，这和我们想要看到的结果完全不同。那么如何修改代码能使得结果正确呢？答案：只需要将 `switch n` 改成 `switch true` 或者 `switch 1` 即可。

`switch` 语句还有另外一种用法，这种用法用到了我们还没正式学过的元胞数组（Cell Array），这里先简单了解下，后面章节会详细介绍元胞数组的知识点。元胞数组和普通的数值矩阵不同，元胞数组中的元素可以是不同类型、不同大小的数据。我们之前学过的数值矩阵中的元素是使用中括号 `[]` 括起来的，而元胞数组中的元素是使用大括号 `{}` 括起来的。例如 `{25, 15, [6, 3]}` 是有三个元素的元胞数组，`{"abc", [1,2; 3,4]}` 是有两个元素的元胞数组。

case 后面的表达式可以是一个元胞数组，如果 switch 后面开关表达式的值和 case 元胞数组中的至少一个元素相同，则为 true. 下面我们来看一个具体的例子：

<pre>month = randi([1,12]) % 农历对应的月份 switch month case {1,2,3} season="春天"; case {4,5,6} season="夏天"; case {7,8,9} season="秋天"; otherwise season="冬天"; end disp(season)</pre>	<p>month 是在区间 <code>[1,12]</code> 上随机生成的一个整数，用来表示农历月份。<code>case</code> 后面现在换成了使用大括号括起来的元胞数组，如果 <code>month</code> 取值为 1、2 或者 3，则会执行第一个 <code>case</code> 对应的语句，此时季节变量 <code>season</code> 为春天；若取值为 4、5 或者 6，则为夏天；以此类推。</p> <p>补充：<code>{1,2,3}</code> 可通过向量 <code>[1,2,3]</code> 转换而来，对应的函数是 <code>num2cell</code>，即 <code>{1,2,3}</code> 等价于 <code>num2cell([1,2,3])</code>。</p>
---	--

再来看一个输出成绩等级的例子。若 `n` 表示考试成绩（假设 `n` 为 0 至 100 的整数，不考虑输入错误的情况），其中 90~100 分等级为 A，80~89 分等级为 B，70~79 分等级为 C，60~69 分等级为 D，60 分以下不及格，请使用 `switch` 语句输出某一具体分数对应的等级。

<pre>n = 81; switch true % 也可以写成 switch 1 case n >= 90 disp("A") case n >= 80 disp("B") case n >= 70 disp("C") case n >= 60 disp("D") otherwise disp("不及格") end</pre>	<pre>n = 81; switch fix(n/10) % 丢掉小数部分取整 case {9, 10} disp("A") case 8 disp("B") case 7 disp("C") case 6 disp("D") otherwise disp("不及格") end</pre>
---	--

以上是两种不同的思路对应的代码，它们都能解决这个问题。事实上使用 `if` 语句解决这个问题更为方便，`switch` 语句可以转换成对应的 `if` 语句，大家课后可以自己尝试转换。

4.2 循环语句

循环语句的基本思想就是重复，通过反复执行某些语句，来满足重复计算的要求。在编写 MATLAB 程序时，循环是一项非常重要且常用的技术，它允许我们有效地执行相同或类似的操作多次，而不必每次都手动重复相同的代码。

在本节中，我们将深入探讨两种主要的循环语句：**for-end** 语句和 **while-end** 语句。这两种循环语句能够以不同的方式控制循环的次数，以及在何时终止循环。其中，**for-end** 语句用于事先已知循环次数的情形，如果在循环开始之前，我们已经知道了循环的次数，那么我们通常使用 **for-end** 语句；而 **while-end** 语句则更适合在循环的次数未知的情况下使用。

4.2.1 for-end 语句

for-end 语句（简称 for 循环）用于事先已知循环次数的情形，其语法如下：

```
for 循环变量 = 向量或者矩阵
    循环体
end % 循环体以 end 结束，千万不能漏写了！
```

在该语法中，循环变量是用于迭代的变量名，它会在每次循环迭代中从向量或矩阵中取出一列的值。数值向量或者矩阵则表示了循环变量可以取值的范围，通常根据实际需要事先给定。一旦循环变量遍历完数值向量或者矩阵中的所有值，循环就会结束。

下面我们来看几个简单的例子：

<pre>x = 1:5; for ii = x ii end % 第二行可以直接写成： % for ii = 1:5 % for 循环一定以 end 结束，别漏写了！</pre>	<pre>% x = [1, 2, 3, 4, 5]是一个行向量，可以看成一个 1 行 5 列 的矩阵，因此每次取出一列的元素，共循环五次。 ii = 1 ii = 2 ii = 3 ii = 4 ii = 5</pre>
<pre>x = [3;5;9;8]; for ii = x ii end % for 和 end 所在的行的最后面什么都不用 写，不需要添加冒号或者分号</pre>	<pre>% 注意：和上一个例子不同，此时的 x 是一个列向量， 只有一列，因此只会循环一次。这种情况几乎用不到。 ii = 4×1 3 5 9 8</pre>
<pre>% 循环遍历矩阵的每一列，这种情况在实际 编程中用的较少 x = [1 5 8; 3 6 9; 4 7 2; 6 5 3]; for ii = x ii end %通常我们将循环体的开头增加缩进，这样看 起来更美观。你可以选中代码，然后智能缩 进，快捷键“ctrl+i”。</pre>	<pre>ii = 4×1 1 3 4 6 ii = 4×1 5 6 7 5 ii = 4×1 8 9 2 3</pre>

通过上面三个例子可以看出，一旦给定的数值向量或者矩阵，**for** 循环的次数就固定了，取决于向量或者矩阵中有多少列。

下面我们来看 for 循环的例子（一定要看讲解视频，会介绍 MATLAB 的断点调试功能）。

(1) 不使用 sum 函数，计算行向量 x 中所有元素的和。

```
x = [5 8 9 1 4 3 7];
s = 0; % 初始化最终的求和结果为 0
for ii = x
    s = s + ii;
end
disp(s)
```

在这个示例中，for 循环遍历了向量 x 中的每个元素，将它们逐个加到变量 s 中，最终得到了所有元素的和。

思考：如果 x 是一个列向量，左侧的代码输出的 s 是什么，应该如何修改代码？

(2) 计算当 n 等于 100 时，下面式子的结果：

$$y(n) = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2}$$

```
n = 100; % 设置 n 的值
y = 0; % 初始化 y 的值为 0
for k = 1:n
    % 计算每一项并累加到 y 中
    y = y + 1 / (k^2);
end
disp(y)
```

在这个示例中，for 循环从 1 到 n 遍历每个整数 k，并计算每一项 $1 / (k^2)$ ，然后将它们累加到变量 y 中。最终，y 的值将是整个表达式的结果。

思考：使用上一章的知识点，如何通过一行代码直接计算 y 的值？

参考答案：y = sum(1./(1:100).^2)

(3) 计算当 n 从 1 一直取到 100 时，上一小问式子的计算结果，并将计算结果保存到一个长度为 100 的行向量 S 中（S 中第 i 个元素表示 y(i) 的结果）。

```
S = zeros(1, 100);
for n = 1:100
    y = 0; % 初始化 y 的值为 0
    for k = 1:n
        y = y + 1 / (k^2);
    end
    S(n) = y;
end
disp(S)
```

这里使用了循环的嵌套，上一问的代码可用来求出任意一个具体的 n 对应的 y。因此，这一问我们只需要使用循环让 n 从 1 遍历到 100，并将每次的计算结果保存到向量 S 中。请大家思考：(1) y = 0; 这行代码能否放在循环的外面？(2) 能否优化左侧的代码，使得通过一次循环就得到 S。

这两个问题留作本章最后的课后习题。



(4) 计算从公元 1 年到公元 9999 年间，有多少个闰年。闰年的判读条件是年份能够被 4 整除，但不能被 100 整除，或者年份能够被 400 整除。

```
% 初始化闰年计数器
leap_year_count = 0;
% 循环遍历从公元 1 年到公元 9999 年的每一年
for year = 1:9999
    % 检查是否为闰年的条件
    if (mod(year, 4) == 0 && mod(year, 100) ~= 0) || (mod(year, 400) == 0)
        % 如果是闰年，增加计数器
        leap_year_count = leap_year_count + 1;
    end
end
disp(leap_year_count)
```

(5) 一个三位正整数各位数字的立方和等于该数本身则称该数为水仙花数，例如： $1^3 + 5^3 + 3^3 = 153$ ，则 153 是水仙花数。请你找出所有的水仙花数并将其保存到向量 S 中。

```
% 初始化存储水仙花数的向量 S 为空
S = [];
% 循环遍历所有的三位整数
for num = 100:999
    % 拆解数字
    digit1 = floor(num / 100);    % 百位
    digit2 = floor(mod(num, 100) / 10); % 十位
    digit3 = mod(num, 10);        % 个位
    % 检查是否为水仙花数的条件
    if num == digit1^3 + digit2^3 + digit3^3
        S = [S, num]; % 若是水仙花数，则添加到向量 S 中
    end
end
% 显示所有的水仙花数
disp(S)
```

(6) 斐波那契数列的递推公式为
$$\begin{cases} F(1)=1, F(2)=1 \\ F(n)=F(n-1)+F(n-2), n \geq 3 \end{cases}$$
，求 $F(30)$ 。

```
n = 30;
F = ones(1,n); % 初始化用来保存中间计算结果的向量全为 1
for k = 3:n
    F(k) = F(k-1) + F(k-2);
end
Fn = F(n)
% 832040
```

(7) 生成一个 5 行 8 列的矩阵，矩阵中每个元素都是在区间[1, 10]上取值的随机整数。接下来请循环每一列，若发现同一列的五个元素各不相同，则保留该列；若该列中有重复的元素则删除该列。

```
% 生成随机整数矩阵
matrix = randi([1, 10], 5, 8);
% 输出生成的随机整数矩阵
disp(matrix)
% 初始化新矩阵，用于存储五个元素各不相同的列
new_matrix = [];
% 循环遍历每一列
for column = matrix
    % 使用 unique 函数取当前列的唯一值
    unique_column = unique(column);
    % 若取完唯一值后还是 5 个元素，则添加到新矩阵中
    if length(unique_column) == 5
        new_matrix = [new_matrix, column];
    end
end
disp(new_matrix) % 显示新矩阵的结果
```

% 原来的矩阵:

7	7	2	6	3	1	7	9
4	4	8	2	4	8	1	4
10	2	4	10	1	4	10	7
3	1	9	3	7	7	9	6
7	5	8	10	2	4	8	6

% 新的矩阵:

7	3	7
4	4	1
2	1	10
1	7	9
5	2	8

思考题：你能不使用循环语句求解这个问题吗？

参考答案：

```
matrix(:,all(diff(sort(matrix))))
```

在使用 for 循环时，需要注意以下事项：

(1) 若 for 语句后面的向量或者矩阵为空，则循环一次也不会被执行。

<pre>for ii = 2:1 x = 10; disp(x) end % MATLAB 什么都不会输出</pre>	<pre>for ii = [] x = 10; disp(x) end % MATLAB 什么都不会输出</pre>
--	---

(2) for 语句后面的向量或者矩阵只会在循环开始时使用一次，向量或者矩阵元素一旦确定将不会再改变。即使你在循环体中改变向量或者矩阵的值，循环变量的值也不改变。

<pre>x = 1:4; for ii = x x = [0 0 0 0]; disp(ii) end disp(x)</pre>	<pre>1 2 3 4 0 0 0 0</pre>
--	----------------------------

(3) 可以在循环体中修改循环变量的值，但当程序执行流程再次回到循环开始时，循环变量会自动恢复成向量或者矩阵的下一列元素。

<pre>for ii = 1:3 disp(ii) ii = 10; disp(ii) end</pre>	<pre>1 10 2 10 3 10</pre>
--	---------------------------

4.2.2 while-end 语句

除了 for-end 语句之外，MATLAB 还提供了另一种强大的循环语句：while-end 语句（简称 while 循环）。与 for 循环不同，**while 循环的特点在于它允许我们在不知道具体循环次数的情况下执行循环体**，这种灵活性使得 while 循环在某些情境下非常有用，尤其是当我们需要满足某些某些条件时才执行循环操作。

while-end 循环语句的语法如下：

```
while 表达式
    循环体
end % 循环体以 end 结束，千万不能漏写了！
```

在这里，表达式通常是一个判断条件，当这个条件为 true 时，循环会不断的迭代执行下去；一旦表达式的值变为 false，循环就会被终止，程序将跳出循环体。

下面我们来看一个具体的例子：已知 $y(n) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$ ，当 n 最小取多少时， y 的值大于 10？

<pre>y = 1; n = 1; while y <= 10 n = n + 1; y = y + 1/n; end disp(n)</pre>	<p>y 和 n 的初始值都是 1，while 后面的表达式是 $y \leq 10$，即只要当前的 y 值小于等于 10，循环就会继续执行。在循环体内，会更新 n 和 y 的值，直到满足 $y > 10$ 才会退出循环，此时的 n 就是最小的满足 $y > 10$ 的 n。</p>
---	--

再来看个例子：斐波那契数列的递推公式为
$$\begin{cases} F(1)=1, F(2)=1 \\ F(n)=F(n-1)+F(n-2), n \geq 3 \end{cases}$$
，求

数列中第一个大于 99999 的元素。

```
a(1) = 1;
a(2) = 1;
n = 2;
while a(n) <= 99999
    n = n + 1;
    a(n) = a(n-1) + a(n-2);
end
disp(a(end))
```

这段代码的计算思路如下：

- ✧ 首先，我们初始化斐波那契数列的前两个元素 $a(1)$ 和 $a(2)$ ，它们都被初始化为 1。
- ✧ 我们引入了一个索引变量 n ，用于表示当前计算的斐波那契数列的第 n 项。初始时， n 被设置为 2，因为我们已经知道了前两项。
- ✧ 进入 while 循环，条件是 $a(n)$ 小于等于 99999。这表示代码将持续计算斐波那契数列，直到找到第一个大于 99999 的元素为止。
- ✧ 在循环中，首先将索引变量 n 变成 $n+1$ ，准备计算下一个斐波那契数。
- ✧ 接着，我们使用递推关系 $a(n) = a(n-1) + a(n-2)$ 计算下一个斐波那契数，并将其存储在向量 a 中的索引变量 n 的位置。
- ✧ 循环会继续，不断计算下一个斐波那契数，直到条件 $a(n) \leq 99999$ 不再满足。
- ✧ 一旦找到第一个大于 99999 的斐波那契数，循环结束。
- ✧ 最后，输出向量 a 中的最后一个元素，它就是第一个大于 99999 的元素。

注意：对于循环次数已知的题目，也可以使用 while 循环进行求解，但求解过程没有 for 循环那么直观。例如要计算 n 等于 100 时， $y(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ 的值，可以使用下面的代码进行求解：


```
y = 1; n = 1;
while n < 100
    n = n + 1;
    y = y + 1/n;
end
disp(y)
```

请大家思考：为什么 while 后面的条件表达式是 $n < 100$ ，而不是 $n \leq 100$ ？

如果将条件改为 $n \leq 100$ ，则循环到 n 等于 100 时还会继续执行，此时 n 等于 101，然后才会退出循环，这会导致 y 多计算一次。

在使用 while 循环时，需要注意以下事项：

- (1) 如果不小心执行了一个**无限循环**（即永远不会自行结束的循环，又称死循环），可以在脚本编辑器或命令行窗口中按下快捷键 **Ctrl+C** 来中断程序的运行。

MATLAB 的左下角会一直显示正忙： 正忙

```
x = 1;
while x < 10
    disp(x)
end
```


(2) `while` 后面表达式的计算结果不一定非得是逻辑值 1 或 0。如果表达式的计算结果是一个数值常数，则只有当这个常数为非零值时循环才会进行；若表达式的计算结果是一个数值向量或者矩阵，则仅当该向量或矩阵中的所有元素都是非零数时循环才会进行。

<pre>ii = 5; while ii disp(ii) ii = ii - 1; end</pre>	<pre>5 4 3 2 1</pre>
<pre>x = [1 2; 3 4]; while x x(1) = 0; disp(x) end</pre>	<pre>0 2 3 4</pre>

4.2.3 break 和 continue

`break` 和 `continue` 也是 MATLAB 中的关键字，它们可以更加灵活地控制循环过程的执行。在 MATLAB 中，**break 和 continue 只能与 for 循环或 while 循环一同使用**，不能用于其他场合。下面我们来简要介绍一下 `break` 和 `continue` 的用法：

- ✧ **break 关键字**用于终止执行 `for` 或 `while` 循环。实际使用中，当满足某个条件时，我们会使用 `break` 立即退出循环。这在找到所需结果后立即退出循环的场景非常有用。
- ✧ **continue 关键字**用于跳过循环的当前迭代，然后继续下一次迭代。实际使用中，当满足某个条件时，`continue` 将跳过当前循环迭代的剩余部分，然后继续进行下一次迭代。这对于在某些情况下跳过特定的迭代非常有用，而不必完全退出循环。

下面我们来看两个简单的案例：

(1) 已知 $y(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ ，当 n 最小取多少时， y 的计算结果大于 10？这个例子在 `while` 函数中出现过，下面我们尝试使用 `for` 循环求解。

<pre>y = 0; for n = 1:1e8 y = y + 1/n; if y > 10 disp(n) break % 退出 for 循环 end % if 配套的 end end % for 配套的 end</pre>	<p>使用 <code>for</code> 循环需要通过向量或者矩阵给出循环的次数，由于我们这个问题的循环次数是未知的，因此可以预先给一个很大的循环范围，我们这里给定 n 为 1 至 1×10^8 构成的向量。</p> <p>如果在循环体中找到了我们所需的结果（即 y 大于 10），就可以通过 <code>break</code> 关键字退出循环。通常情况下，判断条件是否成立需要用到 <code>if</code> 语句。</p>
--	--

(2) 使用循环输出 1 至 10 中所有的奇数。

<pre>for i = 1:10 if mod(i, 2) == 0 continue end disp(i) end</pre>	<p>在每次迭代中，使用 <code>mod(i, 2)</code> 来检查 i 是否为偶数。如果 i 是偶数，那么 <code>mod(i, 2)</code> 的结果将为 0，这时会执行 <code>continue</code>，直接跳到下一次迭代，因此当 i 为偶数时，代码后面的 <code>disp(i)</code> 不会被执行；只有 i 为奇数时才会被输出。</p> <p>思考：如果不使用 <code>continue</code> 关键字，代码应该如何修改？</p>
--	--

注意，如果存在循环的嵌套，**break** 和 **continue** 仅在调用它的循环的主体中起作用。即 **break** 仅从它所发生的循环中退出，**continue** 仅跳过它所发生的循环体内的剩余语句。

下面我们来看一个典型的例子，该例子中有两个 **for** 循环，因此存在循环的嵌套，我们称第一次出现的循环为外层循环（简称外循环），第二次出现的循环称为内层循环（简称内循环）。另外，在内循环中我们加上了 **if** 条件语句，并在 **if** 的条件不满足时使用了 **break** 关键字，此时的 **break** 由于出现在内循环中，因此在起作用时仅会跳过内循环，外循环会继续下去。

```
for ii = 1:2 % 外循环
    for jj = 1:3 % 内循环
        if jj <= ii
            disp(ii)
            disp(jj)
        else
            break
        end % if 配套的 end
    end % 内循环配套的 end
end % 外循环配套的 end
```

```
1
1
2
1
2
2
% 配套的代码文件中还有一个思考题
```

上面这段代码详细的计算思路如下：

1. 外层循环（由变量 **ii** 控制，**ii** = 1:2）首先从 **ii** 的值 1 开始。
2. 内层循环（由变量 **jj** 控制）在每个外层循环迭代内部执行。
3. 在外层循环的第一次迭代（**ii** 等于 1）内部，内层循环（**jj** = 1:3）执行。
 - ✧ 当 **jj** 等于 1 时，满足条件 **jj** <= **ii**，因此执行 **disp(ii)** 和 **disp(jj)**，输出 1 和 1。
 - ✧ 当 **jj** 等于 2 时，不再满足条件 **jj** <= **ii**，因此执行 **break** 语句退出内层循环。
4. 外层循环的第二次迭代（**ii** 等于 2）开始。
5. 再次进入内层循环（**jj** = 1:3），**jj** 从 1 重新开始。
 - ✧ 当 **jj** 等于 1 时，满足条件 **jj** <= **ii**，因此执行 **disp(ii)** 和 **disp(jj)**，输出 2 和 1。
 - ✧ 当 **jj** 等于 2 时，满足条件 **jj** <= **ii**，因此执行 **disp(ii)** 和 **disp(jj)**，输出 2 和 2。
 - ✧ 当 **jj** 等于 3 时，不再满足条件 **jj** <= **ii**，因此执行 **break** 语句退出内层循环。

在实际编程中，**break** 的使用频率远高于 **continue**，下面我们来看几道典型例题。

（1）质数（Prime number），又称素数，指在大于 1 的自然数中，除了 1 和该数自身外，无法被其他自然数整除的数（也可定义为只有 1 与该数本身两个正因数的数）。给定任意一个大于 100 的自然数 **n**（例如 **n**=135389），请判断 **n** 是否为质数。

思路：我们可以遍历从 2 到 **n**-1 的所有整数，检查它们是否能够整除 **n**。如果找到任何一个能够整除 **n** 的整数，那么 **n** 就不是质数；否则，**n** 就是质数。

```
n = 135389; %要判断的数 n
is_prime = true; % 初始化标志变量 is_prime 为 true, 此时代表 n 是质数
for ii = 2:n-1 % 思考：如何缩小循环遍历的范围来提高代码运行的效率，留作本章课后习题
    % 检查 ii 是否能够整除 n
    if mod(n, ii) == 0
        % 如果能整除，则 n 不是质数，将标志变量 is_prime 重新赋值为 false
        is_prime = false;
        break; % 跳出循环
    end
end
disp(is_prime)
```

(2) 一副扑克牌有 54 张牌（桃杏梅方四种花色的 A 2 3 4 5 6 7 8 9 10 J Q K 加双王），假设三名玩家玩斗地主，其中地主有 20 张牌，两个农民各 17 张牌。若你是其中一名玩家，且你每次都选择当农民。请编程模拟以下场景：先玩第一把，若这把手上有炸弹则这把玩完后下场换其他人玩；若手上没有炸弹则继续玩第二把，直到玩到第 k 把时手上有炸弹，此时玩完这一把后下场换其他人玩。请输出你模拟的 k 。注意：假设每把牌都洗的足够的混乱，确保为无序；有炸弹是指手上有双王或者有四张相同的牌例如 4 张 3。（这里的 k 表示你作为农民首次出现炸弹的轮数，由于发牌过程是随机的，那么 k 肯定也是一个随机的变量，即每次模拟的 k 可能都不相同。比如运气好可能第一把就出现了炸弹，此时 k 等于 1，运气不好可能需要好多把才会出现炸弹，此时 k 较大）。



思路：由于循环的次数不定，因此我们可以使用 `while` 循环来不断模拟游戏的进行，直到满足退出的条件。在每一轮中，我们作为农民会随机抽取 17 张牌，并检查是否有双王或者普通的炸弹。如果有任何一种炸弹，就会退出循环，否则会增加游戏的轮数，继续下一轮。最后，我们可以输出模拟的 k 值，表示玩到第 k 把时手上有炸弹。

```
% 用 1 至 13 分别代替 A 2 3 ... J Q K, 重复 4 次表示四种花色; 用 14 和 15 分别代表大小王
poke = [ repmat(1:13,1,4), 14, 15 ]; % 生成一副扑克牌
k = 1; % 玩了多少把游戏
while 1
    % 从 1:54 中随机抽取 17 个数, 表示 17 张牌对应的下标
    idx = randperm(54,17); % randperm 函数的用法: 《第 3 章: 课后习题讲解中拓展的函数》
    % 发牌并排序 (排序后牌面看起来更清楚一点, 事实上不排序也不影响下面的代码)
    p = sort(poke(idx));
    % 检查是否有大小王
    v1 = all(ismember([14,15],p)); % v1 为 true 表示有双王, 为 false 表示没有双王
    % 检查是否有普通的炸弹
    v2 = false; % v2 表示是否有普通的炸弹, 先假设没有, 因此初始化为 false
    for ii = 1:13
        if sum(p == ii) == 4 % 如果有四张一样的牌
            v2 = true; % 将 v2 赋值为 true, 表示有普通的炸弹
            break % 只要有一个普通的炸弹就可以退出 for 循环了
        end
    end
    if v1 || v2 % 如果有王炸或者普通炸弹就可以退出 while 循环
        break % 跳出 while 循环
    else
        k = k + 1; % 没有炸弹就再玩一把
    end
end
disp(k) % 输出首次出现炸弹时玩的轮数
```

在本题中，既用到了 `while` 循环又用到了 `for` 循环，且出现了两个不同用途的 `break`，大家课后一定要认真消化，并尝试自己求解这个例题（当然，判断是否存在普通的炸可以不用循环语句，我们在第三章的课后习题中有讲解，详情请看第三章课后习题挑战篇的 Q5）。

另外，本题还能继续扩展下去，例如重复上面的模拟过程 N 次（ N 可以设置得大一点，例如 N 等于 10 万），得到这 N 次模拟结果的 k ，并计算这 N 次 k 的平均值，这个平均值就能表示你作为农民首次出现炸弹所需的期望轮数。这个拓展的问题将留作本章最后的课后习题，我们下一道题也会介绍类似的思想。

(3) 一只失明的小猫掉进山洞里，山洞有三个门，其中进入第一个门后走 2h 后可以回到地面，进入第二个门后走 4h 会回到原始的出发点，进入第三个门后走 6h 还是回到原始的出发点。假设小猫每次都随机地选择这三个门中的一个进入，求小猫走出山洞的期望时间？

思路：在上一章的课后习题中，我们见到过类似的题目，当时我们介绍过蒙特卡罗模拟这种方法，蒙特卡罗模拟将所求解的问题同一定的概率模型相联系，用计算机实现统计模拟或抽样来获得问题的近似解。我们可以模拟这个过程 N 次（ N 一般要设置的大一点，例如让 N 等于 10 万），每次模拟中我们都让一只猫走出山洞，并记录下这只猫所需的时间。接下来我们只需要对这 N 次模拟结果得到的时间计算平均值，就能估计小猫走出山洞的期望时间。

```
N = 100000; % 设置模拟的次数
T = zeros(N,1); % T 用来存储每次模拟得到的时间
for ii = 1:N % 开始进行 N 次模拟
    t = 0; % 初始化时间
    while 1 % 开始模拟小猫走出山洞的过程
        choose = randi(3); % 随机选择第几个门
        if choose == 1 % 选择第 1 个门
            t = t + 2; % 走 2 小时回到地面
            break % 小猫成功走出山洞，结束模拟
        elseif choose == 2 % 选择第 2 个门
            t = t + 4; % 走 4 小时回到原始出发点
        else % 选择第 3 个门
            t = t + 6; % 走 6 小时回到原始出发点
        end
    end
    T(ii) = t; % 记录每次模拟得到的时间
end
mean(T) % 计算所有模拟结果的平均值，即小猫走出山洞的期望时间的估计
```

(4) 这个例题我们介绍二分搜索法求函数零点。若函数 $f(x)$ 在区间 $[a, b]$ 上连续严格单调，且满足 $f(a) \times f(b) < 0$ ，那么 $f(x)$ 在区间 $[a, b]$ 上有且仅有一个零点。二分搜索法的基本思想是不断将区间 $[a, b]$ 一分为二，然后判断零点位于哪一半区间内，接着继续将包含零点的那一半区间一分为二，如此循环，直到得到足够精确的零点的估计值。以下是二分搜索法的一般步骤：

步骤 1：选择函数零点所在的初始区间 $[a, b]$ ，确保 $f(a) \times f(b) < 0$ 。

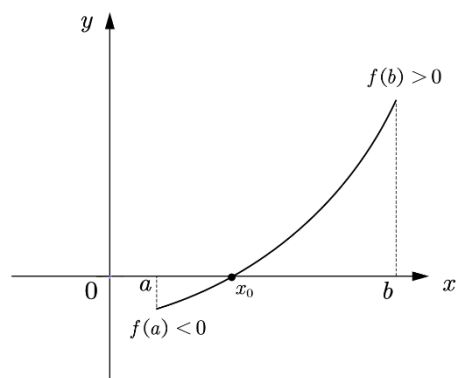
步骤 2：计算区间的中点 $c = (a + b) / 2$ ，并计算函数在 c 处的值 $f(c)$ 。

步骤 3：如果 $f(c)$ 的值恰好等于零，或者 $f(c)$ 的绝对值小于某个给定的误差阈值，那么 c 就可以当成零点，迭代结束。

步骤 4：如果 $f(c)$ 与零的差异较大，那么需要根据 $f(c)$ 的正负号，将原来包含零点的区间 $[a, b]$ 更换为 $[a, c]$ 或 $[c, b]$ ，确保零点仍然在新的区间内（例如： $f(a) \times f(c) < 0$ 则更换为 $[a, c]$ ）。

步骤 5：重复步骤 2 到 4，直到找到零点或者达到所需的精度停止迭代。

下面看一个具体的题目：函数 $f(x) = x^3 - 8x^2 + x - 5$ ， $f(x)$ 在区间 $[6, 10]$ 严格递增且 $f(6) < 0, f(10) > 0$ ，请用二分搜索法求零点 x_0 （ $f(x_0)$ 和 0 的误差控制在 $1e-8$ 内即可）。



```

% 设置初始搜索区间
a = 6; b = 10;
% 设置误差阈值
epsilon = 1e-8;
% 开始二分搜索
while 1
    % 计算区间中点
    c = (a + b) / 2;
    % 计算中点处的函数值
    fc = c^3 - 8*c^2 + c - 5;
    % 如果中点处的函数值已经足够接近零, 停止搜索
    if abs(fc) < epsilon
        break
    end
    % 否则根据函数值的正负来调整搜索区间
    fa = a^3 - 8*a^2 + a - 5;
    if fa * fc < 0 % f(a) * f(c) < 0
        b = c; % 区间更换为[a,c]
    else % f(c) * f(b) < 0
        a = c; % 区间更换为[c,b]
    end
end
% 找到的零点估计值
x0 = c;
disp(x0)

```

4.3 处理异常的 try-catch 语句

在编写 MATLAB 程序时, 我们需要考虑程序的健壮性和容错性。有时候, 我们编写的程序会出现意想不到的错误或异常情况, 如数据输入错误、文件读取问题、网络连接错误等。为了更好地处理这些异常情况, MATLAB 提供了一种强大的工具, 即 try-catch 语句, 它允许我们在代码块中尝试执行可能会引发异常的操作, 并在异常发生时捕获并处理它们, 而不会导致程序报错终止。

try-catch 语句的基本结构如下:

```

try
    可能引发异常的代码块
catch
    异常处理代码块
end % try-catch 语句以 end 结束, 别漏写了!

```

在上述结构中, 我们在 try 中编写可能引发异常的代码。如果异常发生, MATLAB 将跳转到 catch 块中, 并执行异常处理代码。

举个例子: A 矩阵是 3 阶的方阵, B 矩阵是一个 5 阶的方阵, 正常情况下我们计算 A+B 就会报错, 若将 A+B 放到 try 语句块内, MATLAB 就不会报错。

```

A = ones(3,3);
B = ones(5,5);
A + B
% 矩阵维度必须一致。(低版本 MATLAB 的报错)
% 对于此运算, 数组的大小不兼容。(较新的版本的报错)

```



```

try
    A + B
catch
    disp("大小不兼容哦，无法计算！")
end
% 大小不兼容哦，无法计算！

```

下面再来看一个例子，这个例子将展示如何处理数组索引越界的异常：

```

M = rand(4,4) % 生成一个 4x4 的随机矩阵
r = 6;
try
    x = M(r, :) % 取 M 的第 r 行元素
catch
    % 若 M 没有第 r 行元素,则返回空向量
    x = []
end

lasterr

```

```

M = 4x4
    0.5000    0.6177    0.1829    0.4899
    0.4799    0.8594    0.2399    0.1679
    0.9047    0.8055    0.8865    0.9787
    0.6099    0.5767    0.0287    0.7127

x =

[]

lasterr

ans = '位置 1 处的索引超出数组边界。索引不能超过 4。'

```

% 低版本 MATLAB: '索引超出矩阵维度。'

注: `lasterr` 函数能够返回 MATLAB 生成的最后一条错误消息，尽管这个错误已经被代码中的 `try-catch` 语句捕获了。

上面两个应用 `try-catch` 语句的例子比较简单，由于大家目前学到的知识有限，有些复杂的应用场景无法给大家介绍，下面给大家描述一些常见的应用 `try-catch` 语句捕获异常的场景。

(1) 文件操作异常：例如 MATLAB 在读取或写入本地文件时容易出现文件不存在、文件被占用、读取或写入权限不足等问题。

(2) 数据处理异常：例如题目给的数据中有 1000 条字符串，我们需要通过某个函数来提取这 1000 条字符串中的数据，但有少数几条字符串在提取数据时会报错，需要单独处理。

(3) 网络通信异常：在使用 MATLAB 执行爬虫操作时（爬取网站上的数据），可能出现的一些网络错误，如网络连接超时、ip 被封禁等。

(4) GUI 应用异常：在 GUI（即图形用户界面）应用程序中处理用户输入或操作可能引发的异常，例如某个输入框中需要用户输入一个数字，结果用户输入了一个字符串。

(5) 信号处理异常：在信号处理或实时控制系统中，可能需要处理硬件故障或信号丢失等异常情况。

(6) 数据库操作异常：MATLAB 与数据库交互时，可能会出现连接中断、SQL 查询错误或数据不一致等异常。

这些不同的应用场景都可以受益于 `try-catch` 语句，处理可能的异常情况能够提高程序的可靠性，也能给使用程序的其他用户提供更好的体验。

4.4 控制程序流程的其他常用指令

在 MATLAB 编程中，除了条件语句、循环语句和 `try-catch` 语句外，还有一些其他常用的程序流程控制指令，它们能用于实现特定的控制需求，例如：计算程序的运行时长、暂停程序的运行、让用户输入数据、显示警告和报错信息等。

在本节中，我们将介绍以下五个常用指令：

指令名称	作用
tic/toc	可以输出 tic 和 toc 中间的程序的运行时长，以秒为单位
pause	暂停程序的执行，可以指定暂停的时间长度，以秒为单位
input	暂停程序让用户输入数据
warning	在程序运行时显示警告信息
error	显示自定义的报错信息并终止程序运行

4.4.1 程序运行时计：tic/toc

tic 和 toc 是一对用于计时的 MATLAB 指令，你需要将 tic 和 toc 分别放在需要计时的代码段的前面和后面，运行代码后 MATLAB 就会输出这段代码的运行时长，单位为秒。

tic
需要计时的代码段
toc

事实上，许多题目都可能有多种求解方法，有的方法求解效率很高，代码运行的时长就会更短。例如我们看下面这个例子：

已知 A 和 B 都是 1000 阶的方阵，且 A 和 B 的每个元素都是在区间[0,1]上均匀分布的随机数。请计算 C 矩阵，其中 C 矩阵的第 i 行第 j 列元素 $c_{ij} = \begin{cases} a_{ij} + b_{ij}, & a_{ij} > b_{ij} \\ a_{ij} \times b_{ij}, & a_{ij} \leq b_{ij} \end{cases}$ 。

首先创建 A 和 B 这两个随机矩阵：

```
n = 1000;
A = rand(n, n);
B = rand(n, n);
```

方法 1：使用 for 循环计算

```
tic
C1 = zeros(n,n); % 初始化最后的计算结果，提高代码的运行的效率
for i = 1:n
    for j = 1:n
        if A(i, j) > B(i, j)
            C1(i, j) = A(i, j) + B(i, j);
        else
            C1(i, j) = A(i, j) * B(i, j);
        end
    end
end
toc
% 历时 0.026510 秒。
```

方法 2：直接使用矩阵运算

```
tic
C2 = (A > B).*(A+B) + (A <= B).*(A.*B);
toc
%历时 0.001956 秒。
```

从上面的运行时间可以看出，第二种方法的计算效率远高于第一种方法。这也提示我们，使用循环编程的效率通常较低，更推荐大家使用向量或者矩阵直接运算。

MATLAB 官网给出了提高代码性能的常用方法，感兴趣的同学可以点击下面链接了解：

https://ww2.mathworks.cn/help/matlab/matlab_prog/techniques-for-improving-performance.html

4.4.2 暂停程序运行：pause

pause 函数用于暂停程序的运行，当程序运行到该命令时，程序暂时中止。它的调用格式有下面两种：

- (1) **pause**：暂时停止代码的执行，等待用户按任意键继续运行；
- (2) **pause(n)**：暂停程序运行 n 秒。

举个例子，每隔 0.5 秒在屏幕上输出一个 $[1,10]$ 上的随机整数，若输出的整数恰好为 10，则停止输出。

```
while 1
    % 生成一个区间[1,10]上的随机整数
    x = randi([1,10],1);
    disp(x) % 输出 x
    pause(0.5) % 暂停 0.5 秒
    % 判断结果是否为 10，若为 10 则退出循环
    if x == 10
        break
    end
end
```

下面我们再使用 **pause** 函数来绘制一个动态的爱心图形，大家现在只需要学习 **pause** 函数的用法即可，后续的章节中我们会专门讲解 MATLAB 中的绘图命令。



```
% 创建一个图形窗口
figure('Color','w')
% 主循环，模拟动画
for t = 0:0.01:2*pi
    % 计算爱心形状的坐标点
    x = 16 * sin(t) .^ 3;
    y = 13 * cos(t) - 5 * cos(2*t) - 2 * cos(3*t) - cos(4*t);
    % 使用 plot 函数绘制爱心
    plot(x, y, '.r', 'LineWidth', 2);
    hold on % 确保在每次迭代时不清空图形窗口
    axis([-20 20 -20 20]); % 设置坐标轴范围
    axis off % 不显示坐标轴
    % 暂停一小段时间
    pause(0.0001);
end
hold off
```

4.4.3 用户输入数据：input

编写 MATLAB 程序时，有时需要从用户那里获取输入的数据，以便程序根据这些数据执行不同的操作或计算。MATLAB 提供了 **input 函数接收来自用户的输入数据**，这在创建交互式程序或需要用户提供输入的情况下非常有用。**input** 函数有下面两种调用方式：

- **x = input(提示用户输入的文本)**
- **txt = input(提示用户输入的文本, 's')**

第一种方式只需要给一个输入参数，即一个提示用户输入的文本，此时 MATLAB 的界面会显示这段文本，并等待用户输入值后按回车键确定。用户可以输入 10、 $\pi/4$ 、**rand(3)**、1:10、[1, 2; 3, 4]之类的数据或表达式，还可以输入工作区中已有的变量。

第二种方式有两个输入参数，这时候 `input` 会将用户的输入看成文本格式，返回的变量 `txt` 是字符向量格式，这对于获取纯文本的输入非常有用。

注意：提示用户输入的文本请使用字符向量（单引号引起来）的形式，如果使用字符串（双引号引起来）则在较低的 MATLAB 版本中运行会报错。


下面我们举两个例子帮助大家理解。

(1) 用户输入圆的半径，然后计算圆的面积。

```
r = input('请输入圆的半径: ');
% r = input("请输入圆的半径: "); % 较低版本的 MATLAB 会报错，例如 MATLAB2017a 版本
area = pi*r*r
```

运行完上面这段代码后，MATLAB 的命令行窗口会出现相应地提示：

命令行窗口
fx 请输入圆的半径:

同时，MATLAB 的左下角会显示： 正在等待输入

如果我们输入 10，那么 MATLAB 会返回 `area` 的计算结果：

area =

314.1593

从这个例题我们可以看出，当 MATLAB 遇到 `input` 语句时，会暂停程序的运行并等待用户的输入，等用户输入完毕后才会计续执行后续的代码。

(2) a 和 b 是用户输入的两个常数，如果用户输入“乘法”这个文本，则计算 $a \times b$ ；如果用户输入“加法”这个文本，则计算 $a + b$ ；如果输入的是其他的文本，则返回空向量。

<pre>a = input('请输入常数 a: '); b = input('请输入常数 b: '); % 下面的 ss 是一个文本，因此这里加上了第二个输入参数 's' ss = input('请输入操作（乘法、加法）: ', 's'); % 根据用户输入的操作执行相应的计算 if strcmp(ss, '乘法') % strcmp 函数比较字符向量是否相同 result = a * b; elseif strcmp(ss, '加法') result = a + b; else result = []; end disp(result)</pre>	<p>请输入常数 a: 10 请输入常数 b: 20 请输入操作（乘法、加法）：乘法 200</p> <p>请输入常数 a: 5 请输入常数 b: 8 请输入操作（乘法、加法）：加法 13</p> <p>请输入常数 a: 10 请输入常数 b: 20 请输入操作（乘法、加法）：减法 % disp([]) 什么都不会出现</p>
--	---

使用 `input` 函数输入数据时，有以下两点需要大家注意：

(1) 第一种调用方法 “ $x = \text{input}(\text{提示用户输入的文本})$ ” 也能够用来保存用户输入的字符串或字符向量，但在输入时需要将字符向量用单引号引起来、字符串用双引号引起来。

以下两种输入字符向量的方式是等价的：

```
>> x1 = input('请输入一个字符向量: ')
请输入一个字符向量: 'abc'
x1 =
'abc' 不带's'时输入字符向量需要自己添加单引号
```

```
>> x2 = input('请输入一个字符向量: ', 's')
请输入一个字符向量: abc
x2 =
'abc' 带's'后输入字符向量不用添加单引号了
```

(2) 使用 `input` 函数后，有时候你会发现 MATLAB 无论输入什么语句都没有反应，这很有可能是因为 MATLAB 在等待我们的输入，你可以观察 MATLAB 左下角的状态提示。如果 MATLAB 卡住了也可以在脚本编辑器或者命令行窗口按 `Ctrl+C` 快捷键来中断程序的运行。

4.4.4 显示警告信息：warning

编写 MATLAB 程序时，警告信息是一种有用的反馈机制，可以用来提示用户在程序执行过程中可能存在的问题或不规范的操作。MATLAB 提供的 **warning 函数用于显示警告信息**。警告消息不会中断程序的执行，但它可以帮助用户和开发人员注意到潜在的问题，以便及时采取措施来解决。

首先介绍 `warning` 函数最常见的用法：**warning off** 和 **warning on**，它们分别用于关闭和打开 MATLAB 的警告信息。

例如运行下面这段代码时，MATLAB 会弹出橙色的警告信息：

<pre>A = [3 6; 4 8]; inv(A) % 计算 A 的逆矩阵</pre>	<p>警告：矩阵在工作精度内为奇异的。</p> <pre>C = 2x2 -Inf Inf Inf -Inf</pre>
---	--

如果我们不希望看到警告信息，那么可以在运行代码之前先运行 `warning off` 命令。

<pre>warning off A = [3 6; 4 8]; inv(A) % 计算 A 的逆矩阵</pre>	<pre>C = 2x2 -Inf Inf Inf -Inf</pre>
---	--

`warning` 函数还有另一种用法，可用来显示自定义的警告信息：**warning(警告信息)**，这里的警告信息请使用字符向量（单引号引起来）的形式，如果使用字符串（双引号引起来）则在较低的 MATLAB 版本中运行会报错。

下面我们举个例子：*a*和*b*是用户输入的两个常数，计算 *a* 除以 *b* 的值，如果 *b* 等于 0，则显示警告：分母不应为 0。

<pre>warning on % 重新开启警告功能 a = input('请输入常数 a: '); b = input('请输入常数 b: '); if b == 0 warning('分母不应为 0') end c = a/b</pre>	<table border="0"> <tr> <td>请输入常数 a: 1 请输入常数 b: 2</td><td>请输入常数 a: 1 请输入常数 b: 0 警告: 分母不应为0</td></tr> <tr> <td>c =</td><td>c =</td></tr> <tr> <td>0.5000</td><td>Inf</td></tr> </table>	请输入常数 a: 1 请输入常数 b: 2	请输入常数 a: 1 请输入常数 b: 0 警告: 分母不应为0	c =	c =	0.5000	Inf
请输入常数 a: 1 请输入常数 b: 2	请输入常数 a: 1 请输入常数 b: 0 警告: 分母不应为0						
c =	c =						
0.5000	Inf						

4.4.5 显示报错信息：error

在 MATLAB 编程中，当程序遇到严重错误或不符预期条件时，可以使用 `error` 函数显示自定义的报错消息，并终止程序的执行。

`error` 函数的用法很简单：**error(报错信息)**，这里的报错信息请使用字符向量（单引号引起来）的形式，如果使用字符串（双引号引起来）则在较低的 MATLAB 版本中运行会报错。

下面我们举个例子：*a*和*b*是用户输入的两个常数，计算 *a* 除以 *b* 的值，如果 *b* 等于 0，则显示报错：分母不应为 0。

<pre>a = input('请输入常数 a: '); b = input('请输入常数 b: '); if b == 0 error('分母不应为 0') end c = a/b</pre>	<table border="0"> <tr> <td>请输入常数 a: 1 请输入常数 b: 2</td><td></td></tr> <tr> <td>c =</td><td>请输入常数 a: 1 请输入常数 b: 0 分母不应为0</td></tr> <tr> <td>0.5000</td><td></td></tr> </table>	请输入常数 a: 1 请输入常数 b: 2		c =	请输入常数 a: 1 请输入常数 b: 0 分母不应为0	0.5000	
请输入常数 a: 1 请输入常数 b: 2							
c =	请输入常数 a: 1 请输入常数 b: 0 分母不应为0						
0.5000							

和 `warning` 函数的结果对比，`error` 函数生成的报错信息是红色的，且结果中没有出现 `c` 的值。这是因为一旦 `error` 函数被执行，程序会立即终止，后面的代码都不会被运行；而 `warning` 函数不会终止程序的执行，后面的代码仍会运行。

4.5 本章小节

(1) **条件语句** 可用来实现分支结构，它能根据不同的条件执行不同的命令。本章介绍了两种条件语句：

- **if-elseif-else-end** 语句：用于多个条件判断，根据条件的不同执行不同的代码块。
- **switch-case-otherwise-end** 语句：将开关表达式的值依次和各个 `case` 后面的表达式的值判断是否相等，如果相等则执行对应的代码块，都不相等则执行 `otherwise` 的代码。

(2) **循环语句** 是用于重复执行某些操作的重要工具，它允许我们有效地执行相同或类似的操作多次，而不必每次都手动重复相同的代码。本章介绍了两种循环语句：

- **for-end** 语句：用于循环次数已知的情形。
- **while-end** 语句：适用于循环次数未知的情形，根据条件判断是否继续执行循环。

此外，我们介绍了 **break** 和 **continue** 的用法，它们可以更加灵活地控制循环过程的执行。其中 `break` 语句用于终止执行 `for` 或 `while` 循环，这在找到所需结果后立即退出循环的场景非常有用；`continue` 语句用于跳过循环的当前迭代，然后继续下一次迭代。

在配套的讲解视频中，我们介绍了 **断点调试功能**，它允许程序在特定位置停止执行，以便观察程序的状态和变量的值。使用断点调试功能可以帮助我们更好地理解代码、发现错误。

(3) 我们编写的程序可能会出现意想不到的错误或异常情况，因此 **异常处理** 非常重要。**try-catch** 语句允许我们在程序中捕获和处理异常，而不会导致程序崩溃。

(4) 本章最后介绍了几个 **常用的程序流程控制指令**，它们能实现特定的控制需求：

- **tic** 和 **toc**：用于测量代码的执行时间，这可以帮助用户评估程序的性能。
- **pause**：用于暂停程序的运行，可以指定暂停的秒数。
- **input**：可以让用户输入数据，支持数据表达式输入和文本输入两种模式。
- **warning**：用于显示警告信息；`warning on/off` 也能打开/关闭 MATLAB 的警告信息。
- **error**：用于显示报错信息并终止程序。

4.6 课后习题

基础篇

Q1. 填空题

- (1) 在执行循环语句时，如果陷入了死循环，可以按快捷键_____中断程序的运行；如果要自动整理代码的缩进、对齐等可以选中代码按快捷键_____。
- (2) `if` 后面通常接上关系运算符或逻辑运算符，若为 `true` 则执行 `if` 后面的语句。如果你在 `if` 后面放上了一个矩阵或者向量，这时候只有当这个矩阵或者向量中的元素满足_____，才会被当成 `true`。
- (3) 程序可能会面临意想不到的错误或异常情况，如数据输入错误、文件读取问题，我们可以使用_____语句捕获和处理异常。
- (4) `input` 函数有两种调用方式，使用_____可获取纯文本输入。
- (5) 要获取代码的运行时间，我们可以_____。
- (6) 代码_____可以将程序暂停 1 分钟。

- (7) 和 if 条件语句配套的剩下三个关键字是_____。
- (8) 和 switch 条件语句配套的剩下三个关键字是_____。
- (9) 函数_____可以将文本、数值等输出到 MATLAB 的窗口。
- (10) 字符向量使用_____引起来，字符串使用_____引起来；比较两个字符向量或者字符串是否相等可以使用函数_____。
- (11) 如果不希望看到警告信息，那么可以在运行代码之前先运行_____命令。
- (12) 元胞数组的元素使用_____括起来；在 switch-case 语句中，如果 case 后面是元胞数组，那么当_____，则为逻辑值 1。
- (13) 如果变量 x 等于 eye(3)，那么循环语句 for ii = x 中，进行到第二次循环时变量 ii 的值是_____。
- (14) 使用循环得到 N 次模拟结果，并将最终的结果计算平均值来估计期望，这种方法称为_____。
- (15) _____函数可用来显示报错信息并终止程序。
- (16) 考虑存在循环嵌套的场景，如果你的代码中用到了两个 for 循环，且在内层循环中使用了一个 break，那么这个 break 会导致_____。
- (17) 使用 warning 函数显示警告信息时，后续的语句_____（会/不会）继续运行。
- (18) 如果 MATLAB 版本较低，使用 warning("abc")会报错，应改成_____。

Q2. 下面这段判断正负数的代码有问题吗？如果有问题应该如何修改？

```
% n 是区间[-10,10]上的随机整数
n = randi([-10,10])
switch n
    case n > 0
        disp("n 是正数")
    case n < 0
        disp("n 是负数")
    otherwise
        disp("n 为 0")
end
```

Q3. 本章 4.2.1 节中有一个思考题，计算当 n 从 1 一直取到 100 时，下面这个表达式的计算结果： $y(n) = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2}$ ，并将计算结果保存到一个长度为 100 的行向量 S 中（S 中第 i 个元素表示 y(i)的结果）。我们课堂上给出的代码是：

```
S = zeros(1, 100);
for n = 1:100
    y = 0; % 初始化 y 的值为 0
    for k = 1:n
        y = y + 1 / (k^2);
    end
    S(n) = y;
end
disp(S)
```

请回答：(1) `y = 0;` 这行代码能否放在循环的外面？ (2) 能否优化上面的代码，只循环一次就得到 S。(3) 能否使用第三章的知识，不使用循环直接计算 S。

Q4. 给定一个包含 3 个元素的行向量 a ，使用 if 语句对向量 a 中的元素按照从小到大的顺序排序，并将排序后的向量保存到 `sort_a` 中。例如原来 $a=[3,8,5]$ ，那么 $\text{sort_}a=[3,5,8]$ 。（你可以使用循环重复运行你的代码 100 次，每次生成的 a 都是随机的，测试你的代码是否有问题）

Q5. 本章 4.2.3 节中，有一道判断 n 是否是质数的例题。当时的思路是遍历从 2 到 $n-1$ 的所有整数，检查它们是否能够整除 n 。如果找到任何一个能够整除 n 的整数，那么 n 就不是质数；否则， n 就是质数。事实上我们可以缩小循环遍历的范围来提高效率。一般来说，只需要检查从 2 到 n 的平方根(如果为小数则需要向上取整)之间的整数，原因是如果 n 有一个大于 $\text{sqrt}(n)$ 的因子，那么它必定有一个小于 $\text{sqrt}(n)$ 的因子。请解决下面两个问题：（1）优化遍历的范围，对于 $n=100000037$ ，比较优化前后的代码的运行时间；（2）自然数 2 至 10000 中的质数有哪些？（注意 2 也是质数，需要单独判断）。

Q6. 下面这两段代码输出的结果分别是什么？

<pre>n = 2; for ii = 1:n+2 n = 5; disp(ii) end</pre>	<pre>for ii = [1;2;3] disp(ii+1) end</pre>
--	--

Q7. 一个五位正整数各位数字的五次方和等于该数本身则称该数为五角星数，请找出所有的五角星数并将其保存到向量 S 中。

Q8. 生成一个 8 行 5 列的矩阵，矩阵中每一个元素都是在区间 $[1,10]$ 上取值的随机整数。接下来请循环每一行，若发现同一行的五个元素中各不相同，则保留该行。

Q9. 在本章介绍 `break` 和 `continue` 的小节中，有一道和斗地主相关的例题。请使用蒙特卡罗模拟计算你作为农民首次出现炸弹所需的轮数的期望值。

Q10. a 和 b 是用户输入的两个常数，如果用户输入“乘法”这个文本，则输出 $a \times b$ ；若用户输入“除法”这个文本，则先判断 b 是否为 0，若为 0 则生成错误信息：分母不能为 0，否则输出 a/b ；如果输入的是其他的文本，则生成警告信息：只能输入乘法或除法。

提高篇

Q1. 一张 100 元的人民币要换成 10 元、5 元和 1 元面值的零钱，要求三种面值的零钱的总张数为 20 张，且三种面值至少都有一张，用一个矩阵表示所有可能的组合，矩阵中每一行的三个元素分别表示三种面值人民币的张数（例如 8 2 10 三个元素就是矩阵的某一行）。

Q2. 假设自然界中有一种动物，它每天被天敌捕食的概率均为 0.02，且每天是否被天敌捕食这个事件是独立的。请使用蒙特卡罗模拟得到这种动物能存活的天数的期望值。

Q3. 一个班上有 30 名同学，请使用蒙特卡罗模拟计算至少有两人是同一天生日的概率。

Q4. 每隔 0.2 秒在屏幕上随机输出 0 或者 1，当连续出现三次 1 时，停止输出。

Q5. 扔一枚正常的硬币，若要扔出连续的 3 个正面，所需扔硬币的期望次数是多少？请使用蒙特卡罗模拟进行计算，精确的数学答案是 14。

Q6. 有三个长度均为 10 的向量，分别是 $r = [5\ 3\ 2\ 2\ 4\ 1\ 3\ 5\ 1\ 4]$ 、 $c = [3\ 4\ 1\ 2\ 5\ 5\ 2\ 4\ 4\ 2]$ 、 $x = [9\ 6\ 3\ 7\ 5\ 1\ 4\ 9\ 2\ 4]$ ，若 A 矩阵是一个元素全为 0 的 5 阶方阵。请根据 r、c 和 x 三个向量给 A 矩阵重新赋值：将 $x(i)$ 赋值给 A 中第 $r(i)$ 行、第 $c(i)$ 列的元素 ($i=1,2,\dots,10$)。另外，大家可以思考能否不使用循环语句得到 A 矩阵？最终得到的 A 矩阵供大家参考：

0	0	0	2	1
3	7	0	0	0
0	4	0	6	0
0	4	0	0	5
0	0	9	9	0

Q7. 随机生成一个各行各列的和均为 1 的 5 阶方阵，且该方阵的元素仅为 0 或 1。例如下面这个矩阵就符合要求。（大家也可以尝试不使用循环语句得到这个随机的方阵）

0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
0	1	0	0	0
1	0	0	0	0

Q8. 将一根长度为 1 米的木棍随机的折成三段，请用蒙特卡洛模拟分别计算以下两个概率：（1）每段长度都不大于 0.45 的概率；（2）这三段能构成一个三角形的概率。

Q9. 一只蜗牛从 10 米深的井底往上爬，它白天爬 1 米，晚上下落 x 米，其中 x 为 $[0,2]$ 米的均匀分布的随机数，求它爬出这口井的期望天数？本题选自知乎，精确答案约为 277 天。

Q10. 给定一个 1 到 1 亿之间的整数，请判断这个整数是否为回文数（反向排列与原来一样的数就叫做回文数。如：1、353、4774 都是回文数）。

Q11. 二分搜索法不仅可以用于求函数的零点，还可以用于寻找向量中的特定值。给定一个长度为 100 且递增排列的向量 x 和一个要查找的目标值 t ，使用二分搜索法确定向量 x 中是否存在目标值 t 。请回答下面两个问题：（1）为了方便，你可以使用下面两句代码随机生成 x 和 t ： $x = \text{sort}(\text{randi}(200,1,100))$ ； $t = 88$ ；若 x 中存在多个 t ，你只需要返回任意一个 t 的下标，如果不存在 t 请返回 0。（2）假设我们令 $x = \text{sort}(\text{randi}(30,1,100))$ ； $t = 8$ ，那么 x 中出现多个 t 的次数将大大增加。请在上一问代码的基础上进行调整，使得代码能够输出 t 所在的所有下标，如果找不到 t 请返回空向量 $[]$ 。（你可以和 `find` 函数找到的结果进行比较：`find(x == t)`，看看结果是否一致）

Q12. 有一个人从原点（第 0 格）开始扔一个六面骰子（骰子数值为 1-6），扔到几就向前走几格，假设他可以无限次扔骰子，问他恰好走过第 100 格的概率是多少？这个题目选自知乎，精确答案约为 $2/7$ 。

Q13. 编写一个猜数游戏的代码，规则如下：在区间 $[1,100]$ 上随机生成一个整数 p ，用户需要去猜这个 p 是多少。用户每猜一次程序都会做出相应的提示“请输入你猜的数字：”。若用户输入所猜的数字小于 p ，则提示“你猜小了!”；若大于 p ，则提示“你猜大了!”；若相等，则提示“恭喜你赢了!”，游戏结束。用户猜的次数不能超过六次，否则提示“最多猜六次，你失败了。”，此时游戏结束。

Q14. 知乎上有这样一个问题：



请使用蒙特卡罗模拟验证这个答案是否可行（假设有 30 个人玩这个游戏）。

Q15. 下面这题来自 2022 年阿里巴巴全球数学竞赛，请大家求解该题。

春节期间，牛奶公司推出了新春集福活动：每盒牛奶都附赠一个红包，红包中藏有下列“虎”，“生”，“威”中的一款图案。（如下图）



集齐两个“虎”，一个“生”，一个“威”即可拼齐成为“虎虎生威”全家福。这项活动一经推出，就成为了网红爆款，很多人希望能够集齐一整套。假设红包中的图案是独立随机分布的（并且不能从红包外观上进行区别），“虎”，“生”，“威”三款红包按均匀概率 $\frac{1}{3}$ 分布。

(1) 收集齐一整套“虎虎生威”全家福所需要购买的牛奶盒数的数学期望是多少？

- (A) $6\frac{1}{3}$;
- (B) $7\frac{1}{3}$;
- (C) $8\frac{1}{3}$;
- (D) $9\frac{1}{3}$;
- (E) 以上都不对。

(2) 在市场部的周会讨论中，大家认为当前的图案投放比例，会导致在收集“虎虎生威”全家福时收集到太多的“生”和“威”，于是探讨可能的改进方案。记图案“虎”、“生”和“威”的投放比例为 (p, q, r) ，那么下面哪种方案下，收集齐一整套“虎虎生威”全家福所需要购买的牛奶盒数的数学期望是最小的？

- (A) $(p, q, r) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$;
- (B) $(p, q, r) = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$;
- (C) $(p, q, r) = (\frac{2}{5}, \frac{3}{10}, \frac{3}{10})$;
- (D) $(p, q, r) = (\frac{3}{4}, \frac{1}{8}, \frac{1}{8})$.

挑战篇

Q1. 排序算法是一类经典的算法，它们能将一个无序的向量变成有序的向量。请大家在网上搜索插入排序、选择排序和冒泡排序的讲解视频或者文字资料，并使用 MATLAB 复现这三种算法。为了测试方便，你可以令 $x = \text{randi}(100, 1, 20)$ ，将 x 中的元素按照从小到大进行排序。若你的代码返回的结果和 $\text{sort}(x)$ 的结果相同，则说明你做对了。（如果你看懂了算法原理也写不出来代码的话，可以参考本章最后一页的附录，里面提供了这三种算法的伪代码）

Q2. 在 MATLAB 中，'parfor'（Parallel for）是一种并行编程工具，它允许在多个处理核心上同时执行循环迭代。这种方法与常规的 for 循环类似，但能够在多个工作进程上并行执行循环迭代，从而加快代码运行速度。这对于需要进行重复计算或处理大型数据集的任务尤为有效。请在 MATLAB 官方网站上搜索关于 'parfor' 的相关信息，并尝试将之前代码中的 for 循环替换为 parfor 循环。测试代码是否能够正常运行，并比较 for 循环和 parfor 循环的执行时间。

Q3. 某游戏中有一把武器，该武器的初始等级为 1 级，在游戏开始时玩家可以免费领取。在游戏中，玩家可以花费金币对该武器进行升级，每次升级需要花费 10000 金币，且该武器最多能被升至 5 级。各等级升级的成功率如下表所示：

等级	1 级	2 级	3 级	4 级	5 级
1	65%	20%	10%	5%	0%
2	25%	40%	20%	10%	5%
3	10%	20%	40%	20%	10%
4	0%	10%	30%	40%	20%

以等级 1 和等级 3 所在的行为例，表格中各元素的解释如下：

- 1 级武器升级时，有 20% 的概率升到 2 级，10% 概率升到 3 级，5% 的概率升到 4 级，65% 的概率不变。
- 3 级武器升级时，10% 概率跌到 1 级，20% 概率跌到 2 级，20% 概率升到 4 级，10% 概率升到 5 级，40% 的概率不变。

请使用蒙特卡罗模拟，计算打造一把 5 级的武器平均需要花费多少金币（答案约 16 万）。

Q4. 在一台设备上，安装有四只型号和规格完全相同的电子管。假设这些电子管的寿命是整数小时，且在 1000 至 2000 小时之间均匀分布。设备运行中若出现电子管损坏，有两种维修方案可供选择：（1）逐个更换方案：每次只更换损坏的那一只电子管，更换单个电子管需要 1 小时的时间；（2）集体更换方案：当任意一只电子管损坏时，同时更换所有四只电子管，一次性更换四只电子管需要 2 小时的时间。已知每只电子管的价格为 100 元，且不论采用哪种维修方案，设备在更换电子管期间都需要暂停运转，导致的损失为每小时 200 元。请使用蒙特卡罗模拟判断选择哪一种维修方案更省钱（你可以设置总的模拟时长为 10 万小时，比较该设备运行 10 万小时后，哪种方案花费更小。参考答案：方案二比方案一节省约 1.3 万）

Q5. 三门问题（Monty Hall problem）又称蒙提霍尔问题或蒙提霍尔悖论，它是一道非常有趣的概率问题，该问题的答案违反大家的直觉。请大家搜索三门问题的相关资料，并使用 MATLAB 验证三门问题的答案（更换门的概率更高为 $2/3$ ）。

Q6. 莱斯利矩阵是英国生态学家 Leslie 于 1945 年提出的一种数学方法，该方法能利用某一初始时刻种群的年龄结构现状，动态地预测种群年龄结构及数量随时间的演变过程。请大家查阅相关资料学习该模型的建模过程，并解决下面这个问题。

已知某动物最长寿命为 10 岁，且初始状态下该动物各年龄组的数据如下表所示：

年龄	0	1	2	3	4	5	6	7	8	9	10
数量(万)	67	61	60	58	54	52	37	33	28	16	6
雌性生育率(%)	0	0	0	23.4	40.9	61.8	39.8	19.7	10.2	0	0
死亡率(%)	9.5	3.2	3.7	3.2	3.9	4.8	6.3	15.6	26.2	31.5	100

表中雌性生育率解释为雌性个体在一个繁殖周期内平均产生的后代数量（包括雄性和雌性）。假设该动物繁衍过程满足以下条件：1. 该动物在各个年龄段的雌雄比例都是 2:1；2. 新出生注意的该动物的雌雄比例也是 2:1；3. 各年龄组内该动物的生育率和死亡率不随时间变化。

请回答以下问题：（1）预测该动物未来 30 年各年龄段的数量以及占比（参考答案：30 年后 0 岁数量为 687846）；（2）若该动物的天敌在第 30 年后开始出现，这将导致从下一年开始各年龄段的死亡率均增加到原来的三倍，雌性生育率也降低到原来的一半。请问该动物从遭遇天敌开始，需要多少年该动物的数量会下降到 1000 只内。（参考答案：42 年）

Q7. 本题节选自 2023 年阿里巴巴全球数学竞赛，题目如下：A 与 B 二人进行“抽鬼牌”游戏。游戏开始时，A 手上有 n 张两两不同的牌。B 手上有 $n+1$ 张牌，其中 n 张牌与 A 手中的牌相同，另一张为“鬼牌”，鬼牌与其他所有的牌都不同。游戏规则为：

- （1）双方交替从对方手中抽取一张牌，A 先从 B 手中抽取。
- （2）若某位玩家抽到对方的牌与自己手中的某张牌一致，则将两张牌丢弃。
- （3）最后剩一张牌（鬼牌）时，持有鬼牌的玩家为输家。

假设每一次抽牌从对方手上抽到任一张牌的概率都相同，请用蒙特卡罗模拟 n 分别为 31 和 32 时，A 获胜的概率。（参考答案： $n=31$ 时为 $17/33$ ， $n=32$ 时为 $9/17$ ）。

Q8. 这是一道排队论的题目。假设某银行工作时间内只有一个服务窗口，工作人员只能逐个接待客户。当来的客户较多时，一部分客户就需要排队等待。若假设以下四个条件成立：

- （1）从银行开始营业起，客户到达的间隔时长(单位为分钟)服从参数 λ 等于 0.1 的指数分布；
- （2）每位客户的服务时长服从均值为 10，方差为 4 的正态分布(单位为分钟，若服务时长小于 1 分钟，则按 1 分钟计算)；
- （3）排队按先到先服务的规则，且不限队伍的长度；
- （4）银行每天工作时长为 8 小时，若客户开始服务的时间比银行下班的时间晚，银行不提供服务。

模拟 100 个工作日，计算银行平均每天服务的客户人数以及客户的平均等待时长。

Q9. 在上一题的基础上，解决以下三个进阶的问题：

- （1）由于客户反馈排队等待时间过长，银行决定开设一个新的服务窗口。请计算在有两个服务窗口的情况下，这 100 个工作日内银行平均每天服务客户的人数以及客户的平均等待时长。为了简化模拟过程，假设银行采取先到先服务的策略，只要有窗口空闲，就给先来排队的客户提供服务（类似于先取号再叫号的策略）。
- （2）假设银行新设立的第二个窗口每天仅在前四个小时提供服务（如果客户在后四小时到达银行，则只能去第一个窗口），请重新计算上述问题。
- （3）请将第 1 小问的两个窗口推广至 p ($p \geq 3$) 个窗口，并求解相同的问题。

练习题的讲解视频：<https://www.bilibili.com/video/BV1Bw411s7Zb>

(b 站搜索：MATLAB 课程第 4 章课后习题讲解——数学建模清风老师)

附录：三种排序算法的伪代码

伪代码（Pseudocode）是一种描述算法的语言，它介于自然语言和编程语言之间。使用伪代码的目的是使被描述的算法可以容易地以任何一种编程语言（C、Java、Python 等）实现，因此伪代码必须结构清晰、可读性好。

下面给出了挑战篇 Q1 中三种排序算法的伪代码。为了帮助大家理解，我在伪代码中加上了注释。另外，下面给出的伪代码中的循环、判断语句没有以 `end` 结尾，大家需要自己根据代码的缩进来判断 `end` 的位置。

(1) 插入排序

```
INSERTIONSORT(A): // 要对 A 进行插入排序
B = A // 创建 A 的副本 B
// 外层循环，从 B 的第二个元素开始，因为单个元素默认已排序
for i = 2 to B.length
    key = B[i] // key 保存当前要插入的元素
    j = i - 1 // j 表示当前考虑插入 key 的位置，开始于 key 的直接左侧
    // 当 j 未到达数组开头且 B[j] 大于 key 时，继续寻找插入位置
    while j > 0 and B[j] > key
        B[j + 1] = B[j] // 将 B[j] 向右移动，为 key 腾出空间
        j = j - 1 // j 向左移动，继续比较
    B[j + 1] = key // 在找到的位置插入 key
return B // 返回排序后的数组 B
```

(2) 选择排序

```
SELECTIONSORT(A): // 要对 A 进行选择排序
B = A // 创建 A 的副本 B
// 外层循环，遍历数组 B
for i = 1 to B.length - 1
    minIdx = i // 假设当前位置 i 是最小元素的位置
    // 内层循环，寻找真正最小元素的位置
    for j = i + 1 to B.length
        if B[j] < B[minIdx]
            minIdx = j // 更新最小元素的位置
    // 如果找到了新的最小值，则进行交换
    if minIdx != i // != 表示不等于
        tmp = B[i] // 保存当前位置的元素
        B[i] = B[minIdx] // 将找到的最小元素放到当前位置
        B[minIdx] = tmp // 将原本在当前位置的元素移动到最小元素之前的位置
return B // 返回排序后的数组 B
```

(3) 冒泡排序

```
BUBBLESORT(A): // 要对 A 进行冒泡排序
B = A // 创建 A 的副本 B
// 外层循环，遍历数组 B
for i = 1 to B.length - 1
    // 内层循环，从数组开始到未排序部分的末尾
    for j = 1 to B.length - i
        // 如果当前元素大于后面的元素，则交换它们
        if B[j] > B[j + 1]
            tmp = B[j] // 保存当前元素
            B[j] = B[j + 1] // 将后面的元素移到当前位置
            B[j + 1] = tmp // 将保存的当前元素移到后面的位置
return B // 返回排序后的数组 B
```