go version

查看go版本

go version

go 环境变量

`GOROOT`和`GOPATH`都是环境变量，其中`GOROOT`是我们安装go开发包的路径，而从Go 1.8版本开始，Go开发包在安装完成后会为`GOPATH`设置一个默认目录，参见下表。

GOPATH在不同操作系统平台上的默认值

平台	GOPATH默认值	举例
Windows	%USERPROFILE%/go	C:\Users\用户名\go
Unix	$HOME/go	/home/用户名/go

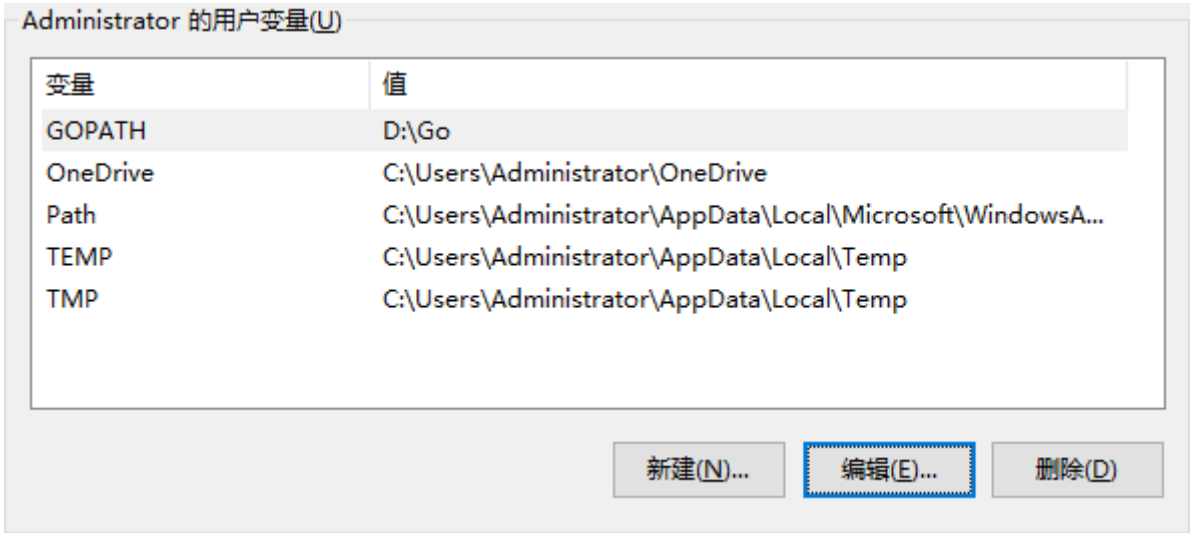
环境变量添加

`GOPATH`

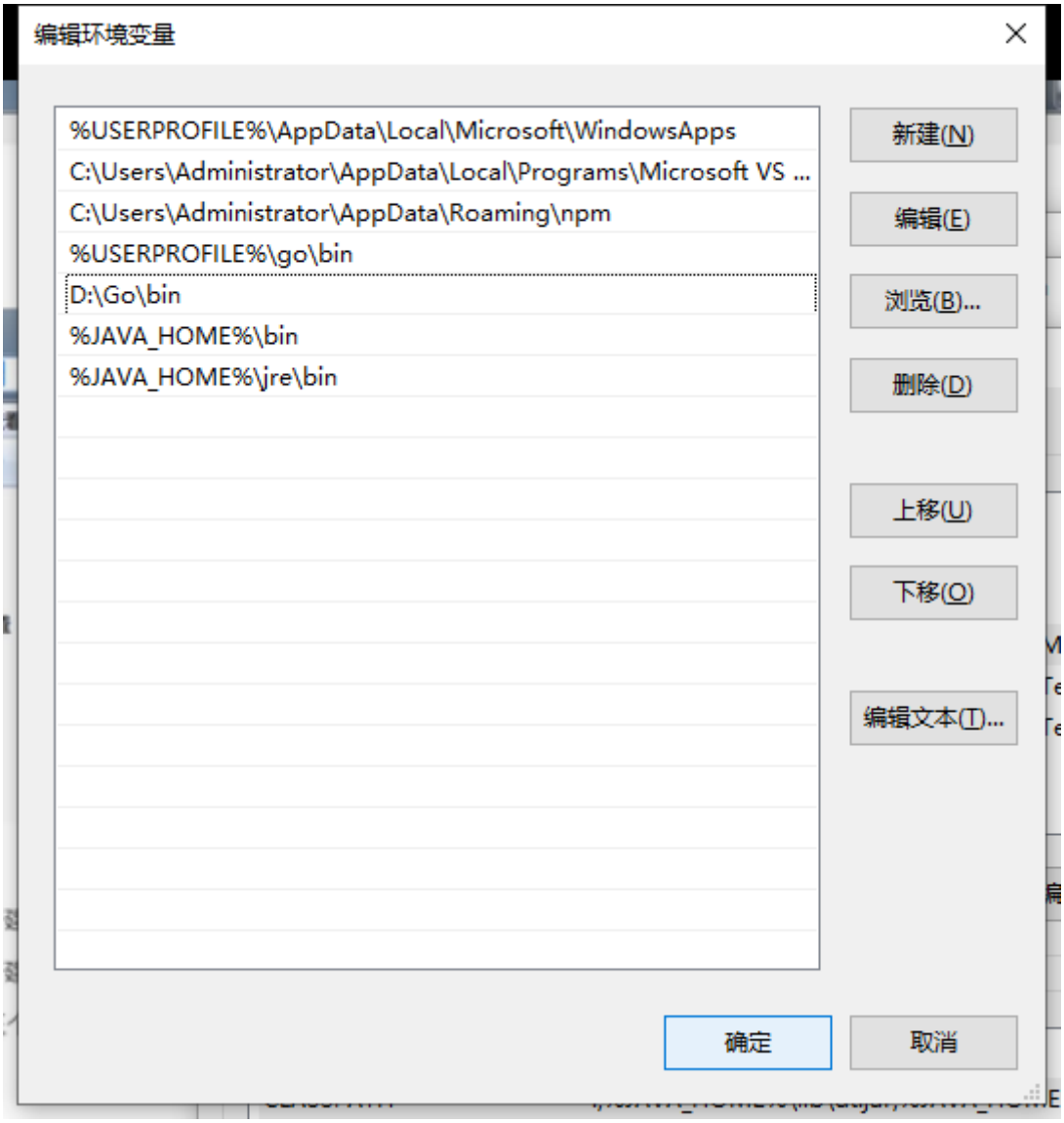
默认

`%USERPROFILE%\go`

可调整自定义目录



添加bin 环境变量 `path`



go 编译

go build

1.当前文件夹

/test/main.go

go build => test.exe

2.如果在任意文件夹下 构建

go build src之后 对应文件目录

go build /xxxx/xxxxx/xxxxxx

3 别名 -o

go build -o sss.exe => sss.exe

mac 不用添加exe

go run

像执行脚本执行文件

go install

1.先编译可执行文件 go build

2.拷贝至gobin 目录 cp => gopath/bin

跨平台编译

不同平台交叉编译

linux mac window

默认我们`go build`的可执行文件都是当前操作系统可执行的文件，如果我想在windows下编译一个linux下可执行文件，那需要怎么做呢?

只需要指定目标操作系统的平台和处理器架构即可:

```
SET CGO_ENABLED=0  // 禁用CGO
SET GOOS=linux  // 目标平台是linux
SET GOARCH=amd64  // 目标处理器架构是amd64
```

使用了cgo的代码是不支持跨平台编译的

然后再执行`go build`命令，得到的就是能够在Linux平台运行的可执行文件了。

Mac 下编译 Linux 和 Windows平台 64位 可执行程序:

```
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build
CGO_ENABLED=0 GOOS=windows GOARCH=amd64 go build
```

Linux 下编译 Mac 和 Windows 平台64位可执行程序:

```
CGO_ENABLED=0 GOOS=darwin GOARCH=amd64 go build
CGO_ENABLED=0 GOOS=windows GOARCH=amd64 go build
```

Windows下编译Mac平台64位可执行程序:

```
SET CGO_ENABLED=0
SET GOOS=darwin
SET GOARCH=amd64
go build
```

go 变量

声明变量

推荐使用驼峰命名法（小驼峰）

标识符

在编程语言中标识符就是程序员定义的具有特殊意义的词，比如变量名、常量名、函数名等等。 Go语言中标识符由字母数字和_(下划线）组成，并且只能以字母和_开头。 举几个例子：`abc`, `_`, `_123`, `a123`。

关键字

关键字是指编程语言中预先定义好的具有特殊含义的标识符。 关键字和保留字都不建议用作变量名。

Go语言中有25个关键字：

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

此外，Go语言中还有37个保留字。

```
Constants:    true  false  iota  nil

    Types:    int  int8  int16  int32  int64
              uint  uint8  uint16  uint32  uint64  uintptr
              float32  float64  complex128  complex64
              bool  byte  rune  string  error

Functions:   make  len  cap  new  append  copy  close  delete
             complex  real  imag
             panic  recover
```

变量

变量的来历

程序运行过程中的数据都是保存在内存中，我们想要在代码中操作某个数据时就需要去内存上找到这个变量，但是如果我们直接在代码中通过内存地址去操作变量的话，代码的可读性会非常差而且还容易出错，所以我们就利用变量将这个数据的内存地址保存起来，以后直接通过这个变量就能找到内存上对应的数据了。

变量类型

变量（Variable）的功能是存储数据。不同的变量保存的数据类型可能会不一样。经过半个多世纪的发展，编程语言已经基本形成了一套固定的类型，常见变量的数据类型有：整型、浮点型、布尔型等。

Go语言中的每一个变量都有自己的类型，并且变量必须经过声明才能开始使用。

变量声明

Go语言中的变量需要声明后才能使用，同一作用域内不支持重复声明。 并且Go语言的变量声明后必须使用。

标准声明

Go语言的变量声明格式为：

```
var 变量名 变量类型
```

变量声明以关键字`var`开头，变量类型放在变量的后面，行尾无需分号。 举个例子：

```
var name string
var age int
var isOk bool
```

批量声明

每声明一个变量就需要写`var`关键字会比较繁琐，go语言中还支持批量变量声明：

```
var (
    a string
    b int
    c bool
    d float32
)
```

变量的初始化

Go语言在声明变量的时候，会自动对变量对应的内存区域进行初始化操作。每个变量会被初始化成其类型的默认值，例如： 整型和浮点型变量的默认值为`0`。 字符串变量的默认值为`空字符串`。 布尔型变量默认为`false`。 切片、函数、指针变量的默认为`nil`。

当然我们也可在声明变量的时候为其指定初始值。变量初始化的标准格式如下：

```
var 变量名 类型 = 表达式
```

举个例子：

```
var name string = "Q1mi"
var age int = 18
```

或者一次初始化多个变量

```
var name, age = "Q1mi", 20
```

类型推导

有时候我们会将变量的类型省略，这个时候编译器会根据等号右边的值来推导变量的类型完成初始化。

```
var name = "Q1mi"
var age = 18
```

短变量声明

在函数内部，可以使用更简略的`:=`方式声明并初始化变量。

```
package main

import (
    "fmt"
)
// 全局变量m
var m = 100

func main() {
    n := 10
    m := 200 // 此处声明局部变量m
    fmt.Println(m, n)
}
```

匿名变量

在使用多重赋值时，如果想要忽略某个值，可以使用`匿名变量（anonymous variable）`。 匿名变量用一个下划线`_`表示，例如：

```
func foo() (int, string) {
    return 10, "Q1mi"
}
func main() {
    x, _ := foo()
    _, y := foo()
    fmt.Println("x=", x)
    fmt.Println("y=", y)
}
```

匿名变量不占用命名空间，不会分配内存，所以匿名变量之间不存在重复声明。 (在`Lua`等编程语言里，匿名变量也被叫做哑元变量。)

注意事项：

1. 函数外的每个语句都必须以关键字开始（var、const、func等）
2. `:=`不能使用在函数外。
3. `_`多用于占位，表示忽略值。

常量

相对于变量，常量是恒定不变的值，多用于定义程序运行期间不会改变的那些值。 常量的声明和变量声明非常类似，只是把`var`换成了`const`，常量在定义的时候必须赋值。

```
const pi = 3.1415
const e = 2.7182
```

声明了`pi`和`e`这两个常量之后，在整个程序运行期间它们的值都不能再发生变化了。

多个常量也可以一起声明：

```
const (
    pi = 3.1415
    e = 2.7182
)
```

const同时声明多个常量时，如果省略了值则表示和上面一行的值相同。 例如：

```
const (
    n1 = 100
    n2
    n3
)
```

上面示例中，常量`n1`、`n2`、`n3`的值都是100。

iota

`iota`是go语言的常量计数器，只能在常量的表达式中使用。

`iota`在const关键字出现时将被重置为0。const中每新增一行常量声明将使`iota`计数一次(iota可理解为const语句块中的行索引)。 使用iota能简化定义，在定义枚举时很有用。

举个例子：

```
const (
        n1 = iota //0
        n2        //1
        n3        //2
        n4        //3
    )
```

几个常见的`iota`示例:

使用`_`跳过某些值

```
const (
        n1 = iota //0
        n2        //1
        _
        n4        //3
    )
```

`iota`声明中间插队

```
const (
        n1 = iota //0
        n2 = 100  //100
        n3 = iota //2
        n4        //3
    )
    const n5 = iota //0
```

定义数量级 （这里的`<<`表示左移操作，`1<<10`表示将1的二进制表示向左移10位，也就是由`1`变成了`10000000000`，也就是十进制的1024。同理`2<<2`表示将2的二进制表示向左移2位，也就是由`10`变成了`1000`，也就是十进制的8。）

```
const (
        _  = iota
        KB = 1 << (10 * iota)
        MB = 1 << (10 * iota)
        GB = 1 << (10 * iota)
        TB = 1 << (10 * iota)
        PB = 1 << (10 * iota)
    )
```

多个`iota`定义在一行

```
const (
        a, b = iota + 1, iota + 2 //1,2
        c, d                      //2,3
        e, f                      //3,4
    )
```

标准声明

```
var name string
var age int
var isOk bool

var name string = "hoyin" // 声明同时赋值
```

批量声明

```
var (
    name    string
    age     int
    isOk    bool
)
```

fmt

```
fmt.Print(isOk)          在终端输出打印得内容
fmt.Printf("%v",isOk)    %s isok 变量占位符
fmt.Println(isOk)        打印指定内容追加换行符
fmt.Printf("%T\n", n)    类型
fmt.Printf("%v\n", n)    值
fmt.Printf("%b\n", n)    二进制
fmt.Printf("%d\n", n)    十进制
fmt.Printf("%o\n", n)    八进制
fmt.Printf("%x\n", n)    十六进制
fmt.Printf("%s\n", n)    字符串
fmt.Printf("%#s\n", n)   添加“字符串”
```

基本数据类型

Go语言中有丰富的数据类型，除了基本的整型、浮点型、布尔型、字符串外，还有数组、切片、结构体、函数、map、通道（channel）等。Go 语言的基本类型和其他语言大同小异。

整型

整型分为以下两个大类： 按长度分为：int8、int16、int32、int64 对应的无符号整型：uint8、uint16、uint32、uint64

其中，`uint8`就是我们熟知的`byte`型，`int16`对应C语言中的`short`型，`int64`对应C语言中的`long`型。

类型	描述
uint8	无符号 8位整型 (0 到 255)
uint16	无符号 16位整型 (0 到 65535)
uint32	无符号 32位整型 (0 到 4294967295)
uint64	无符号 64位整型 (0 到 18446744073709551615)
int8	有符号 8位整型 (-128 到 127)
int16	有符号 16位整型 (-32768 到 32767)
int32	有符号 32位整型 (-2147483648 到 2147483647)
int64	有符号 64位整型 (-9223372036854775808 到 9223372036854775807)

特殊整型

类型	描述
uint	32位操作系统上就是`uint32`，64位操作系统上就是`uint64`
int	32位操作系统上就是`int32`，64位操作系统上就是`int64`
uintptr	无符号整型，用于存放一个指针

注意： 在使用`int`和`uint`类型时，不能假定它是32位或64位的整型，而是考虑`int`和`uint`可能在不同平台上的差异。

注意事项 获取对象的长度的内建`len()`函数返回的长度可以根据不同平台的字节长度进行变化。实际使用中，切片或 map 的元素数量等都可以用`int`来表示。在涉及到二进制传输、读写文件的结构描述时，为了保持文件的结构不会受到不同编译目标平台字节长度的影响，不要使用`int`和`uint`。

数字字面量语法（Number literals syntax）

Go1.13版本之后引入了数字字面量语法，这样便于开发者以二进制、八进制或十六进制浮点数的格式定义数字，例如：

`v := 0b00101101`，代表二进制的 101101，相当于十进制的 45。`v := 0o377`，代表八进制的 377，相当于十进制的 255。`v := 0x1p-2`，代表十六进制的 1 除以 2^2，也就是 0.25。

而且还允许我们用`_`来分隔数字，比如说：`v := 123_456`表示 v 的值等于 123456。

我们可以借助fmt函数来将一个整数以不同进制形式展示。

```
package main

import "fmt"

func main(){
    // 十进制
    var a int = 10
    fmt.Printf("%d \n", a)  // 10
    fmt.Printf("%b \n", a)  // 1010   占位符%b表示二进制

    // 八进制  以0开头
    var b int = 077
    fmt.Printf("%o \n", b)  // 77

    // 十六进制  以0x开头
    var c int = 0xff
    fmt.Printf("%x \n", c)  // ff
    fmt.Printf("%X \n", c)  // FF
}
```

浮点型

Go语言支持两种浮点型数：`float32`和`float64`。这两种浮点型数据格式遵循`IEEE 754`标准：`float32`的浮点数的最大范围约为`3.4e38`，可以使用常量定义：`math.MaxFloat32`。`float64`的浮点数的最大范围约为`1.8e308`，可以使用一个常量定义：`math.MaxFloat64`。

打印浮点数时，可以使用`fmt`包配合动词`%f`，代码如下：

```
package main
import (
        "fmt"
        "math"
)
func main() {
        fmt.Printf("%f\n", math.Pi)
        fmt.Printf("%.2f\n", math.Pi)
}
```

复数

complex64和complex128

```
var c1 complex64
c1 = 1 + 2i
var c2 complex128
c2 = 2 + 3i
fmt.Println(c1)
fmt.Println(c2)
```

复数有实部和虚部，complex64的实部和虚部为32位，complex128的实部和虚部为64位。

布尔值

Go语言中以`bool`类型进行声明布尔型数据，布尔型数据只有`true`（真）和`false`（假）两个值。

注意：

1. 布尔类型变量的默认值为`false`。
2. Go 语言中不允许将整型强制转换为布尔型.
3. 布尔型无法参与数值运算，也无法与其他类型进行转换。

字符串

Go语言中的字符串以原生数据类型出现，使用字符串就像使用其他原生数据类型（int、bool、float32、float64 等）一样。 Go 语言里的字符串的内部实现使用`UTF-8`编码。 字符串的值为`双引号(")`中的内容，可以在Go语言的源码中直接添加非ASCII码字符，例如：

```
s1 := "hello"
s2 := "你好"
```

字符串转义符

Go 语言的字符串常见转义符包含回车、换行、单双引号、制表符等，如下表所示。

转义符	含义
`\r`	回车符（返回行首）
`\n`	换行符（直接跳到下一行的同列位置）
`\t`	制表符
`\'`	单引号
`\"`	双引号
`\\`	反斜杠

举个例子，我们要打印一个Windows平台下的一个文件路径：

```go
package main
import (
    "fmt"
)


func main() {
    fmt.Println("str := \"c:\\Code\\lesson1\\go.exe\"")
    //双引号里包含单引号不用换转译
    fmt.Println("str := 'c:\\Code\\lesson1\\go.exe\'")
}
```

多行字符串

Go语言中要定义一个多行字符串时，就必须使用`反引号`字符：

```go
s1 := `第一行
第二行
第三行
`
fmt.Println(s1)
```

反引号间换行将被作为字符串中的换行，但是所有的转义字符均无效，文本将会原样输出。

字符串的常用操作

方法	介绍
len(str)	求长度
+或fmt.Sprintf	拼接字符串
strings.Split	分割
strings.contains	判断是否包含
strings.HasPrefix,strings.HasSuffix	前缀/后缀判断
strings.Index(),strings.LastIndex()	子串出现的位置
strings.Join(a[]string, sep string)	join操作

byte和rune类型

组成每个字符串的元素叫做“字符”，可以通过遍历或者单个获取字符串元素获得字符。 字符用单引号（’）包裹起来，如：

```go
var a := '中'
var b := 'x'
```

Go 语言的字符有以下两种：

1. `uint8`类型，或者叫 byte 型，代表了`ASCII码`的一个字符。
2. `rune`类型，代表一个`UTF-8字符`。

当需要处理中文、日文或者其他复合字符时，则需要用到`rune`类型。`rune`类型实际是一个`int32`。

Go 使用了特殊的 rune 类型来处理 Unicode，让基于 Unicode 的文本处理更为方便，也可以使用 byte 型进行默认字符串处理，性能和扩展性都有照顾。

```
// 遍历字符串
func traversalString() {
    s := "hello沙河"
    for i := 0; i < len(s); i++ { //byte
        fmt.Printf("%v(%c) ", s[i], s[i])
    }
    fmt.Println()
    for _, r := range s { //rune
        fmt.Printf("%v(%c) ", r, r)
    }
    fmt.Println()
}
```

输出：

```
104(h) 101(e) 108(l) 108(l) 111(o) 230(æ) 178(²) 153() 230(æ) 178(²) 179(³)
104(h) 101(e) 108(l) 108(l) 111(o) 27801(沙) 27827(河)
```

因为UTF8编码下一个中文汉字由3~4个字节组成，所以我们不能简单的按照字节去遍历一个包含中文的字符串，否则就会出现上面输出中第一行的结果。

字符串底层是一个byte数组，所以可以和`[]byte`类型相互转换。字符串是不能修改的 字符串是由byte字节组成，所以字符串的长度是byte字节的长度。 rune类型用来表示utf8字符，一个rune字符由一个或多个byte组成。

修改字符串

要修改字符串，需要先将其转换成`[]rune`或`[]byte`，完成后再转换为`string`。无论哪种转换，都会重新分配内存，并复制字节数组。

```
func changeString() {
    s1 := "big"
    // 强制类型转换
    byteS1 := []byte(s1)
    byteS1[0] = 'p'
    fmt.Println(string(byteS1))

    s2 := "白萝卜"
    runeS2 := []rune(s2)
    runeS2[0] = '红'
    fmt.Println(string(runeS2))
}
```

类型转换

Go语言中只有强制类型转换，没有隐式类型转换。该语法只能在两个类型之间支持相互转换的时候使用。

强制类型转换的基本语法如下：

```
T(表达式)
```

其中，T表示要转换的类型。表达式包括变量、复杂算子和函数返回值等.

比如计算直角三角形的斜边长时使用math包的Sqrt()函数，该函数接收的是float64类型的参数，而变量a和b都是int类型的，这个时候就需要将a和b强制类型转换为float64类型。

```go
func sqrtDemo() {
    var a, b = 3, 4
    var c int
    // math.Sqrt()接收的参数是float64类型，需要强制转换
    c = int(math.Sqrt(float64(a*a + b*b)))
    fmt.Println(c)
}
```

运算符

Go 语言内置的运算符有：

1. 算术运算符
2. 关系运算符
3. 逻辑运算符
4. 位运算符
5. 赋值运算符

算数运算符

运算符	描述
+	相加
-	相减
*	相乘
/	相除
%	求余

注意： `++`（自增）和`--`（自减）在Go语言中是单独的语句，并不是运算符。

关系运算符

运算符	描述
==	检查两个值是否相等，如果相等返回 True 否则返回 False。
!=	检查两个值是否不相等，如果不相等返回 True 否则返回 False。
>	检查左边值是否大于右边值，如果是返回 True 否则返回 False。
>=	检查左边值是否大于等于右边值，如果是返回 True 否则返回 False。
<	检查左边值是否小于右边值，如果是返回 True 否则返回 False。
<=	检查左边值是否小于等于右边值，如果是返回 True 否则返回 False。

逻辑运算符

运算符	描述
&&	逻辑 AND 运算符。 如果两边的操作数都是 True，则为 True，否则为 False。
||	逻辑 OR 运算符。 如果两边的操作数有一个 True，则为 True，否则为 False。
!	逻辑 NOT 运算符。 如果条件为 True，则为 False，否则为 True。

位运算符

位运算符对整数在内存中的二进制位进行操作。

运算符	描述
&	参与运算的两数各对应的二进位相与。 （两位均为1才为1）
|	参与运算的两数各对应的二进位相或。 （两位有一个为1就为1）
^	参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为1。 （两位不一样则为1）
<<	左移n位就是乘以2的n次方。 “a<<b”是把a的各二进位全部左移b位，高位丢弃，低位补0。
>>	右移n位就是除以2的n次方。 “a>>b”是把a的各二进位全部右移b位。

赋值运算符

运算符	描述
=	简单的赋值运算符，将一个表达式的值赋给一个左值
+=	相加后再赋值
-=	相减后再赋值
*=	相乘后再赋值
/=	相除后再赋值
%=	求余后再赋值
<<=	左移后赋值
>>=	右移后赋值
&=	按位与后赋值
|=	按位或后赋值
^=	按位异或后赋值