Arduino 程序手册参考翻译

Arduino 程序可分为四部分：结构，语法，数值（变量和常量）和函数。

Structure

setup()
loop()

Control Structures

if
if...else
for
switch case
while
do... while
break
continue
return
goto

Further Syntax

; (semicolon)
{} (curly braces)
// (single line comment)
/* */ (multi-line comment)
#define
#include

Arithmetic Operators

= (assignment operator)
+ (addition)

Variables

Constants

HIGH | LOW
INPUT | OUTPUT|INPUT_PULLUP
true | false
integer constants
floating point constants

Data Types

void
boolean
char
unsigned char
byte
int
unsigned int
word
long
unsigned long
short
float
double
string - char array
String - object
array

Conversion

char()
byte()
int()
word()
long()

Functions

Digital I/O

pinMode()
digitalWrite()
digitalRead()

Analog I/O

analogReference()
analogRead()
analogWrite() - *PWM*

Advanced I/O

tone()
noTone()
shiftOut()
shiftIn()
pulseIn()

Time

millis()
micros()
delay()
delayMicroseconds()

Math

min()
max()
abs()
constrain()
map()
pow()

- (subtraction)
* (multiplication)
/ (division)
% (modulo)

Comparison Operators

== (equal to)
!= (not equal to)
< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)

Boolean Operators

&& (and)
|| (or)
! (not)

Pointer Access Operators

* dereference operator
& reference operator

Bitwise Operators

& (bitwise and)
| (bitwise or)
^ (bitwise xor)
~ (bitwise not)
<< (bitshift left)
>> (bitshift right)

float()

Utilities

sizeof()

sqrt()

Trigonometry

sin()
cos()
tan()

Random Numbers

randomSeed()
random()

Bits and Bytes

lowByte()
highByte()
bitRead()
bitWrite()
bitSet()
bitClear()
bit()

External Interrupts

attachInterrupt()
detachInterrupt()

Interrupts

interrupts()
noInterrupts()

Communication

Serial
Stream

Compound Operators

++ (increment)
-- (decrement)
+= (compound addition)
-= (compound subtraction)
*= (compound multiplication)
/= (compound division)
&= (compound bitwise and)
|= (compound bitwise or)

Structure（结构）

基本结构

一个基本的 arduino 程序是由 setup()和 loop()函数组成。

setup()

setup()函数在每个程序开始时被调用，其作用是定义变量、引脚模式定义（即把相应引脚定义为输出或者输入等）和开始调用库中的函数等。setup 函数只在 arduino 板每次上电或复位后运行一次。

Loop()

Loop()函数在生成 setup()函数(完成初始值的设定等初始化操作)后实现程序的循环功能,其连续地循环，允许程序变化和响应以便主动地控制 arduino 板的运行。

Example:

```
/*
   Blink
LED 闪烁函数
重复地执行 LED 开 1s,关 1s。
 这个示例程序可以在公共示例域中找到。
 */

void setup() {
          //   pinMode()函数为引脚模式定义函数，选择一个引脚并将其定义为输入或输出
   pinMode(13, OUTPUT);      //定义 13 号引脚为输出模式
}

void loop() {
   //digitalWrite()函数为引脚写入函数，将某个引脚赋值高低电平
   digitalWrite(13, HIGH);   //将 13 号引脚赋值为高电平即点亮 LED
   // delay( )延时函数，
   delay(1000);                    //延时 1s
   digitalWrite(13, LOW);     //  将 13 号引脚赋值为低电平即熄灭 LED
   delay(1000);                    //延时 1s
}
```

控制结构

If

If（条件）和比较运算符（等于==,不等于！=,小于<,大于>）
If 通常和比较运算符联用，测试某一条件是否被满足，例如某次输入是否大于某个值。If 测试语句的格式如下：

```
If(某个变量>50)
{
//条件满足的话执行某条指令
}
```

Example:：

```
If（a>120）digitalwrite(LEDpin,HIGH);//如果 a>120, LEDpin 为高电平
```

if……else

Example:：

```
if(a>120) digitalread(LEDpin)//读取 LEDpin 的值
else    digitalread(LED2pin) //读取 LED2pin 的值
```

for

for 语句用于重复执行花括号里的内容。一个增量计数器用于定期增加并且终止循环。for 用于一些重复性的操作，并且经常与数组联用来收集数据。For 循环包括三个部分：

```
for(初始化；条件；增量)
{ 执行的语句  }
```

初始化仅执行一次。每次执行循环判断条件是否满足，如果满足，执行花括号里的语句，增量递增，然后继续判断条件；如果条件不满足，循环结束。

Example:：

```
/*依次点亮 8 个 LED 灯*/
void setup()
{  }
void loop()
{
for( int LEDpin=0; LEDpin <8; LEDpin ++)
{ digitalwrite(LEDpin,HIGH);}
}
```

switch ……case

像 if 语句一样，switch ……case 语句通过指定在不同的变量值时执行不同的指令代码来控制程序的运行。具体而言，switch 语句把变量值与 case 语句的值作对比，若 case 语句的值与 switch 语句的变量值相等时，case 语句里的代码被执行。关键词 break 存在于每一个 case 语句里。没有 break 语句，switch 语句继续执行下一条

语句，直到找到 break 语句或者 switch 语句结束。

```
switch (变量) {
    case 1:
      //当变量等于 1 时，执行 case 1 后面的语句
      break;
    case 2:
      //当变量等于 2 时，执行 case 2 后面的语句
      break;
    default:
      // 如果没有匹配的值，执行 default 后面的语句
      // default 语句可省略
  }
```

While

While 循环语句将无限地连续循环，直到圆括号里的表达式不成立。测试变量必须发生改变，否则 while 循环不会退出。这个可以在代码里，比如一个增加的变量，或者一个外部的条件像测试一个传感器。

Example:：

```
while(表达式){
  // 执行的语句
}
Int a,b;
while(a>20){b=a;}
```

do……while

do 循环和 while 循环以同样的方式工作，除了循环语句末尾的测试条件，所以 do 循环至少执行一次。

Example:：

```
do
{
    //执行的语句
} while (测试条件);
do
{
  delay(50);              // 延时 50ms，
  x = readSensors();   //把传感器的值赋值给 X
} while (x < 100);
```

break

break 用于跳出 do,for,while 循环，绕过正常的循环条件。它也用于跳出 switch 循环。

Example:：

```
/*16 个 LED 等依次闪烁,如果按下开关，灯常亮跳出循环*/
int ledPin,button
void setup()
{   pinMode(button, INPUT);
   pinMode(ledPin, OUTPUT);          // 设置 ledpin 为输出模式
void loop()
{
for ( int x = 0; x < 17; x ++)
   {     ledpin=x;
      digitalWrite(ledPin, HIGH);     // 打开 LED 灯
      delay(1000);                          //延时 1s
      digitalWrite(ledPin, LOW);      // 关闭 LED 灯
      delay(1000);                          //延时 1s
   if（button）
   digitalWrite(ledPin, HIGH);
   break；
   }
}

{
      digitalWrite(LEDpin, x);
      sens = analogRead(sensorPin);
      if (sens > threshold){          // bail out on sensor detect
           x = 0;
           break;
      }
      delay(50);
}
```

continue

continue 循环跳过当前循环（do,for,while）其后的程序。它通过判断循环的条件表达式继续运行，直接进入下一个子循环。

Example:：

```
for (x = 0; x < 255; x ++)
{
      if (x > 40 && x < 120){          // 在 40 和 120 之间的数值不执行 continue 下面的语句，直接跳到下一个子循环。
            continue;
      }
      digitalWrite(PWMpin, x);
      delay(50);
}
```

return

结束一个函数并且如果需要的话，返回一个函数值给调用函数。

Example:：

```
//一个比较传感器的值与极限值大小的函数
Void loop()
{
x=checkSensor();//如果条件成立，checkSensor 函数返回数值 1，即 x=1.

}
  int checkSensor(){
      if (analogRead(0) > 400) {
            return 1;
      else{
            return 0;
      }
}
```

goto

转移程序流到程序里有标签的点。

```
for(byte r = 0; r < 255; r++){
      for(byte g = 255; g > -1; g--){
            for(byte b = 0; b < 255; b++){
                  if (analogRead(0) > 250){ goto bailout;}//如果条件成立跳到并执行 bailout 行
                  // more statements ...
            }
      }
}
bailout:
```

syntax---语法

； ---分号

用于结束一个语句。

Example:：

```
int a = 13;
```

{}---花括号

表示一个语句块。

花括号在 ARDUINO 程序中，当选中括号的一边时，另一边会高亮显示，这表示两者是不可分割的。

//---双斜杠、/*……*/

程序中的注释符号

双斜杠后面跟着的是程序的注释，用于单行程序的注释，目的是让自己或者其他人知道程序运行的方式。注释在编译时是被忽略的并且不会输出给处理器，所以它们在 atmega 单片机中是不占用任何空间的。

/*…*/用于多行的注释

Example:：

x = 5; // 这是一个单行注释.

/* 这是一个多行注释 – 用于给整块代码添加注释。*/

#define ---定义

#define 在 C 语言中是一个很有用的语句，用于在程序编译之前给常量值定义一个常量名。在 Arduino 中，被定义的常量名不占用芯片的内存。编译器在编译时将会用常量值取代常量名。

虽然有时候也会有一些负面的影响，例如一个被#define 定义的常量名包含在其他的常量名或变量名中，这种情况下都会被#define 定义的常量值所取代。

Arduino 有着同样的 C 定义语法。

注意：“#”是必须使用的。

Example:：

#define ledPin 3

//编译器在编译时将会用数值 3 取代任何提到 ledPin 的位置。

#include---包含

#include 用于把外面的库函数包含在程序中。这使得程序员可以使用大量的 C 标准库函数（大量的前置函数），还有专门为 Arduino 写的库函数。

注意：#include 和#define 一样没有分号结束符。

Example:：

#include <avr/pgmspace.h>

运算符

算数运算符

=（赋值运算符）

储存等号右边的值到等号左边的变量里

Example:：

```
int sensVal;                    // 定义一个名为 sensVal 的整型变量
senVal = analogRead(0);         //储存模拟引脚 0 读取的数字化输入值到 sensVal 变量中。
```

+(加)、—（减）、*（乘）、/（除）

这些运算符返回两个操作数的和、差、积、商。这个运算是根据操作数的数据类型来指导的。例如 9/4=2，因为 9 和 4 都是整数。这也意味着如果结果大于数据类型所能储存的值，运算就会溢出。如果操作数是不同的类型，那么（存储空间）较大的类型用于储存计算结果。

Example:s：

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

%（取模）

当一个整数除以另一个整数时计算它的余数。

Example:s：

```
x = 7 % 5;    // x = 2
x = 9 % 5;    // x = 4
x = 5 % 5;    // x =0
x = 4 % 5;    // x =4
```

比较运算符（==、！=、<、>、<=、>=）

```
x == y (x 等于 y)
x != y (x 不等于 y)
x <  y (x 小于 y)
x >  y (x 大于 y)
x <= y (x 小于或等于 y)   x >= y (x 大于或等于 y)
```

布尔运算符（&&逻辑与、||逻辑或、！逻辑非）

&&

仅当两个操作数都为真时，结果才为真

```
if (digitalRead(2) == HIGH   && digitalRead(3) == HIGH) {
  // ...
}
```

||

如果一个操作数为真，结果就为真

```
if (x > 0 || y > 0) {
  // ...
}
```

!

如果操作数为假时，结果为真。

```
if (!x) {
   // ...
}
```

按位运算符

&（按位与）

按位与在附近的表达式的每位进行与运算。如果两位都是 1，那么结果输出为 1；否则的话结果输出为 0.另一种表达方式如下：

```
0   0   1   1     操作数 1
0   1   0   1     操作数 2
----------
0   0   0   1     结果
```

|按位或

按位或运算符是一个竖线（|）。两个二进制位的按位或运算只有当其中一位或两位为 1 时，结果为 1；否则的话结果为 0.

```
 0   0   1   1    操作数 1
0   1   0   1    操作数 2
 ----------
0   1   1   1     结果
```

^按位抑或

按位抑或操作符用插入符号^表示。当两个二进制位不同时，结果为 1；否则的话结果为 0.

```
 0   0   1   1     操作数 1
 0   1   0   1     操作数 2
  ----------
 0   1   1   0     结果
```

~按位非

按位非运算符用波浪符~表示。按位非把一个二进制位变成它的相反位。

```
 0   1     操作数 1
 ----------
 1   0   ~ 结果
```

《（逐位左移）和》（逐位右移）

这些运算符使左边操作数的二进制位向左或向右移动，移动的位数由右边操作数的数值决定。

Example::

```
int a = 5;              //二进制位: 0000000000000101
int b = a << 3;     //二进制位: 0000000000101000, or 十进制表示的 40
int c = b >> 3;     //二进制位: 0000000000000101
```

复合运算符

++（递增）/--（递减）

对于一个整型或长整型变量自增一

x++; // 变量 X 递增一并且返回增量前的 X 原值。
++x; // 变量 X 递增一并且返回新的 X 值
x-- ; //变量 X 递减一并且返回增量前的 X 原值。
--x ; //变量 X 递增一并且返回新的 X 值

+=（复合加法）, -=（复合减法）, *=（复合乘法）, /=（复合除法）

复合（加、减、乘、除）法运算符仅仅是一种速记写法，其展开形式如下：

x += y; // 等价于 x = x + y;
x -= y; //等价于 x = x - y;
x *= y; //等价于 x = x * y;
x /= y; //等价于 x = x / y;

Example::

x = 2;
x += 4; // x = 6
x -= 3; // x= 3
x *= 10; // x=30
x /= 2; // x=15

(&=)复合按位与运算符

按位与运算符通常用于一个变量和一个常量去强制某个特别的位为 0，这就是经常在编程指南中提及的清零位或者复位位。

Syntax:

x &= y; // 等价于 x = x & y;

Parameters:：

x: 整型、长整形或字符型变量

y:整型常量或字符型、整型或者长整型

Example::

和 0 按位与运算后的位将被清零，所以如果 myByte 是一个字节变量，那么：

myByte & B00000000 = 0;

和 1 按位与运算后的位将不会变化，所以

myByte & B11111111 = myByte;

(|=)复合按位或运算

复合按位或运算符经常用于变量和常量中使变量中特定位置为 1.

Syntax:

x |= y; // 等价于 x = x | y;

Parameters:

x: 整型、长整形或字符型变量

y:整型常量或字符型、整型或者长整型

Example::

和 0 按位或运算后的位将不会变化，所以如果 myByte 是一个字节变量，那么：

myByte | B00000000 = myByte;

和 1 按位或运算后的位将被置为 1，所以，

myByte | B11111111 = B11111111;

Variables（变量）

Constant---常量

在 arduino 语言中，常量是被预定义的变量。常量是程序更容易阅读。

布尔常量（真与假）

在 Arduino 语言中有两个常量用于代表正确和错误：真与假。

False---假

假是两者中更容易去定义的。假被定义为 0.

True---真

真通常被定义为 1，但是也可有一个更广泛的定义。在逻辑意义上，任何不为零的整数即为真。所以-1,2 和-200 都可以定义为真。

注意：真假常量是小写字母不像 HIGH, LOW, INPUT, & OUTPUT 等字符。

引脚常量（HIGH、LOW）

HIGH---高电平、LOW---低电平

定义引脚的电平：高电平和低电平。

当读或写一个数字引脚的时候，仅有两种可能的值：高电平和低电平。

高电平

对于一个引脚来说高电平的意思是有点不同的，根据引脚被设置成输入或输出。

当引脚被 pinMode 函数设置为输入并且用 digitalRead 函数读取数值时，如果引脚当时的电压是大于等于 3v，处理器将识别为高电平。

引脚也可能被 pinMode 函数配置为输入并且随后被 digitalWrite 设置为高电平时，将会设置内部的 20K 的上拉电阻来控制输入引脚为高电平状态，除非被外部电路拉低。

当引脚被 pinMode 函数设置为输出并且被 digitalWrite 设置为高电平时，引脚电压为 5v. 在这种情况下，引脚会成为源电流（即输出电流），例如点亮一个 LED 灯，这个灯和和一系列电阻串联接地或者和另一个被配置成低电压输出的引脚相连。

低电平

对于一个引脚来说低电平的意思是有点不同的，根据引脚被设置成输入或输出。

当引脚被 pinMode 函数设置为输入并且用 digitalRead 函数读取数值时，如果引脚当时的电压是小于等于 2v，处理器将识别为低电平。

当引脚被 pinMode 函数设置为输出并且被 digitalWrite 设置为低电平时，引脚电压为 0v. 在这种情况下，引脚会灌电流（即吸收电流），例如点亮一个 LED 灯，这个灯和穿过一系列电阻和 5v 电压相连或者和另一个被配置成高电压输出的引脚相连。

数字引脚状态定义：INPUT, INPUT_PULLUP, and OUTPUT

数字引脚可以被定义为**输入，内部上拉电阻输入或者输出。用** pinMode()函数改变引脚的状态将会改变引脚的电气特性。

配置为输入状态的引脚

用 pinMode()函数配置为输入状态的 Arduino 引脚据说是处在高阻态。被配置为输入状态的引脚在取样信号电路中有很低的要求，等价于在引脚前面串联一个 100 欧姆的电阻。这种状态在读取传感器的信号时是非常有用的，但是不适合于给 LED 供给电能。

如果你配置一个引脚为输入状态，会想把它接一个参考地，正如 Digital Read Serial 例程中描述的经常连接一个下拉电阻。

配置为内部上拉电阻输入状态的引脚

Arduino 板上的 Atmega 芯片有内部上拉电阻（内部电路中连接电源的电阻）。如果你宁愿用这些代替外部下拉电阻，你可以用 pinMode()函数设置为 **INPUT_PULLUP 状态。这实际上会反转电气特性，高电平意味着传感器关并且低电平意味着传感器开。**Input Pullup Serial 例程中有相关的应用。

配置为输出状态的引脚

用 pinMode()函数配置为输出状态的 Arduino 引脚据说是处在低阻态。这意味着可以给别的电路提供大量的电流。Atmega 引脚可以给其他设备或者电路 提供最大 40ma 的正电流或者负电流。这使得它们可以给 LED 供给能量但是不能用于读取传感器的值。被配置为输出的引脚也会被毁坏如果端接到地或者 5V 电源。Atmega 引脚提供的电流值也是不足以供给大多数的继电器或着电动机，因此一些接口电路是必不可少的。

整型常量

整型常量是直接用于程序中的数字，比如 123.默认地，这些数字被当做整型常量但是你可以用 U 和 L 格式器 修改（如下）。

正常情况下，整型常量是被当做十进制整数，但是特别的格式符号也会用于输入其他进制的数值。

Base	Example:	Formatter（格式符号）	Comment
10 (decimal)	123	none	
2 (binary)	B1111011	leading 'B'	only works with 8 bit values (0 to 255) characters 0-1 valid

8 (octal)	0173	leading "0"	characters 0-7 valid
16 (hexadecimal)	0x7B	leading "0x"	characters 0-9, A-F, a-f valid

Decimal：十进制是大家熟知的常识性的数学。没有前缀的常量是被默认为十进制。
Example::
101 // same as 101 decimal ((1 * 10^2) + (0 * 10^1) + 1)

Binary：只有 0 和 1 是有效值。
Example::
B101 // same as 5 decimal ((1 * 2^2) + (0 * 2^1) + 1)

Octal i：只有字符 0 到 7 是有效值。八进制用前缀 "0"表示。
Example::
0101 // same as 65 decimal ((1 * 8^2) + (0 * 8^1) + 1)
Hexadecimal (or hex)：有效字符是 0 到 9 和字母 A 到 F.A 代表 10，B 代表 11，直到 F。十六进制用前缀“ox”表示。注意：A-F（或者 a-f）大小写不能混用。
Example::
0x101 // same as 257 decimal ((1 * 16^2) + (0 * 16^1) + 1)

U & L 格式器

指定一个整型常量为另一个数据类型，如下：
'u' or 'U'强制常量为无符号数据格式。Example:: 33u
'l' or 'L'强制常量为无长整形数据格式。Example:: 100000L
'ul' or 'UL'强制常量为无符号长整形数据格式。Example:: 32767ul

浮点型常量

浮点型常量和整型常量一样是为了阅读方便。
Example:s:
n = .005;
浮点型常量也可以表达成科学技术法的形式。’E’和’e’都可以是有效的指数。

floating-point constant	evaluates to:	also evaluates to:
10.0	10	
2.34E5	2.34 * 10^5	234000
67e-12	67.0 * 10^-12	.000000000067

数据类型：

void

关键字 void 只用于函数声明。它表明函数不希望给调用它的函数返回任何信息。

Example::

```
void setup()//setup()函数不返回任何值给主函数
{
  // ...
}
void loop()
{
  // ...
}
```

Boolean 布尔型

一个布尔型变量持有两个之中的一个：真或假。（每个布尔变量占用一个字节的内存）

Example:：

```
int LEDpin = 5;        //定义引脚 5 为 LEDpin
int switchPin = 13;    //定义 13 号引脚为 switchPin（13 号引脚接了一个瞬时开关），另一端接地。
boolean running = false;//定义布尔型常量 running 为 false（假）。
void setup()
{
  pinMode(LEDpin, OUTPUT);//用 pinMode（）函数设置 LEDpin 为输出
  pinMode(switchPin, INPUT);// 用 pinMode（）函数设置 switchPin 为输入
  digitalWrite(switchPin, HIGH);       // 打开上拉电阻
}
void loop()
{
  if (digitalRead(switchPin) == LOW)//开关被按下，
  {  // switch is pressed - pullup keeps pin high normally
    delay(100);                          // 延时 100ms(去除按键开关的抖动)
    running = !running;                  // 反转 running 变量
    digitalWrite(LEDpin, running)        // 通过 LED 灯指示出来 running 变量的改变。
  }
}
```

Char 字符型

存储一个字符值占用一个字节内存的数据类型。字符常量常用单引号表示，例如'A'（多字符串用双引号表示）
字符通常是以数字的形式存储的。你可以在 ASCII 码图表中看到具体的编码。这意味着可以用字符进行算术运算（实际上用的是字符的 ASCII 值，例如'A' + 1 =66, 大写字母 A 的 ASCII 值为 65.）在 Serial.println 例程中你可以找到更多的字符转换成数字的例子。
字符型数据类型是一种有符号类型，意味着它可以编码数字从-128 到 127。
Example:：

```
char myChar = 'A';
char myChar = 65;       // 这两种表达方式都可以。
```

unsigned char

无符号数据类型占用 1 个字节的内存，无符号字符数据类型可以编码数字 0 到 255.
Example:：

```
unsigned char myChar = 240;
```

byte

一个字节存储 8 位二进制数字，从 0 到 255.
Example:

```
byte b = B10010;  // "B" 是二进制格式标示符(B10010 = 18 decimal)
```

int

Description:

Integers are your primary data-type for number storage.
On the Arduino Uno (and other ATMega based boards) an int stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).
整数是数字存储的原始数据类型。在 Arduino Uno 板（和其他的 ATMega 基板）中，一个整数存储 16 位二进制数（即两个字节）。取值范围从-32,768 到 32,767（即最小值-2^15 和最大值(2^15) - 1）。
在 Arduino Due 板中，一个整数存储 32 个二进制位（即 4 个字节）。取值范围从 -2,147,483,648 t 到 2,147,483,647 (即最小值-2^31 和最大值 (2^31) - 1).
整型用一种称为 2 的补数的技巧存储负数。最高位有时也称为符号位，标记数值为一负数。其余位取反然后再加 1.。
Example:

```
int ledPin = 13;
```

Syntax

```
int var = val;
```

- var – 你定义的整数变量名
- val – 指定给变量的值

unsigned int

在 Uno 和其他的 ATMEGA 基板中，无符号整型和整型都是存储 2 个字节的值。与存储负数不同的是，无符号整型只存储取值范围为 0 到 65,535 (2^16) - 1)的正数。Due 基板中的无符号整型存储 4 个字节的值，取值范围从 0 到 4,294,967,295 (2^32 - 1)。无符号整型和符号整型的区别在于最高位，有时被称为符号位，在 Arduino 有符号整型类型中，如果高电位为 1，数值被翻译成负数并且其他的 15 位是用 2 的补数翻译。

Example:

```
unsigned int ledPin = 13;
```

Syntax

```
unsigned int var = val;
```

- var –定义的无符号整型变量名
- val – 赋给变量的值

word

一个字存储 16 位二进制数，从 0 到 65535.

Example:

```
word w = 10000;
```

long

长整型变量是容量扩展的数字存储，存储 32 个二进制位，从-2,147,483,648 到 2,147,483,647.

Example:

```
long speedOfLight = 186000L;     // 在整型常量的例子中可以知道 18600L 是指强制转化为长整型数据。
```

Syntax

```
long var = val;
```

- var – 定义的变量名
- val – 赋给变量的值

unsigned long

Description:

Unsigned long variables are extended size variables for number storage, and store 32 bits (4 bytes). Unlike standard longs unsigned longs won't store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).
无符号长整型变量是扩展容量的数据存储变量，能够存储 32 位二进制数。不想标准的长整型，无符号长整形不会存储负数，取值范围为 0 t 到 4,294,967,295 (2^32 - 1)。

Example:

```
unsigned long time;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.print("Time: ");
  time = millis();
  //打印程序开始运行到该行的时间
  Serial.println(time);
  // 为了不去发送大量的数据而延时 1s.
  delay(1000);
}
```

Syntax

```
unsigned long var = val;
```

var –定义的变量名

- val –赋给变量的值

short

短整型是一个 16 位的数据类型。
在所有的 Arduinos 板(ATMega 基板 和 ARM 基板)中，短整型能存储 16 位二进制数。取值范围从-32,768 到 32,767

Example:

```
short ledPin = 13;
```

Syntax

```
short var = val;
```

- var –定义的变量名
- val –赋给变量的值

float

浮点数数据类型有一个小数点。浮点数经常用于近似模拟的连续值，因为相比整数而言，它们有更大的分辨率。浮点数可以和 3.4028235E+38 一样大，也可以和-3.4028235E+38 一样小。浮点数可以存储 32 个二进制位的数值。
浮点数只有 6-7 位的小数位数的精度。这意味着所有的数字位，而不仅仅是小数点右边的数字位数。不像其他的平台那样可以用 double（双精度浮点数）来存储更多的精度位数，Arduino 平台中，double（双精度浮点数）和 float（单精度浮点数）有着同样的容量。
浮点数是不精确的并且当用来比较时可能会放弃奇怪的结果。

Example:s

```
float myfloat;
float sensorCalbrate = 1.117;
```

Syntax

```
float var = val;
```

- var –定义的变量名
- val –赋给变量的值

Example: Code

```
int x;
int y;
float z;
x = 1;
y = x / 2;              // y=0,整数相除结果还是整数。
z = (float)x / 2.0;     // z=0.5,(float)为强制转换并且除数是浮点数 2.0 而不是整数 2
```

double

在 Uno 和 ATMEGA 基板中，双精度浮点型数值占用 4 个字节的空间。也就是说和单精度浮点型数值一样，并没有增加精度。在 Due 基板中，双精度浮点数占用 8 个字节的空间。

String-char array(字符串数组)

Example:s

```
char Str1[15];//声明一个未初始化的字符串数组
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};//声明一个字符串数组，编译器会自动在其末尾加一个零终止符。
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};//明确地加上了零位终止符。
char Str4[ ] = "arduino";//以引号的格式初始化一个字符串，编译器自动设置存储空间并且添加零位终止符。
```

```
char Str5[8] = "arduino";//用明确的容量和字符串常量来初始化字符数组。
char Str6[15] = "arduino";//初始化数组，留下额外的空间给更大的字符串。
```

Null termination（零位终止符）

通常地，字符串末尾用一个零字符（ASCII 码 0）表示终止。这使得函数可以分辨字符串的末尾。否则的话，将会继续读取随后的不属于这个字符串的内存字节。这意味着你定义的字符串需要有额外的空间来存储比你期望的更多的字符。这就是为什莫 Str2 和 Str5 需要 8 个字符，即使 arduino 仅仅是 7 个字符，这个最后的位置是用一个零字符自动填充的。Str4 将自动默认为 8 个字符长度，一个分配给额外的零字符。在 Str3 字符串中，明确地包含了末尾的零字符。

字符串总是用双引号（“abc”）定义并且单个字符通常用单引号（‘a’）定义。

You can wrap long strings like this:

```
char myString[] = "This is the first line"
" this is the second line"
" etcetera";
```

Example:

```
char* myStrings[]={"This is string 1", "This is string 2", "This is string 3",
"This is string 4", "This is string 5","This is string 6"};

void setup(){
Serial.begin(9600);
}

void loop(){
for (int i = 0; i < 6; i++){
   Serial.println(myStrings[i]);
   delay(500);
   }
}
```

Arrays

数组是许多用索引号存取的变量。Arduino 中的数组是基于 C 语言的，可能会更复杂。但是用简单的数组通常更直接。

Creating (Declaring) an Array

创建或者声明一个数组：

```
int myInts[6];//没有初始化的数组
int myPins[] = {2, 4, 8, 3, 6};//一个没有具体指定其大小的数组
int mySensVals[6] = {2, 4, -8, 3, 2};//初始化并且指定大小的数组。
```

```
  char message[6] = "hello";
```

Accessing an Array（访问一个数组）

数组是零索引的，也就是说，上文中提到的数组初始化，数组的第一个元素是以 0 为起始索引值的。

```
mySensVals[0] == 2, mySensVals[1] == 4,……
```

这也意味着含有 10 个元素的数组，9 就是最后的索引值。

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
     // myArray[9]    包含 11
```

For this reason you should be careful in accessing arrays. Accessing past the end of an array (using an index number greater than your declared array size - 1) is reading from memory that is in use for other purposes. Reading from these locations is probably not going to do much except yield invalid data. Writing to random memory locations is definitely a bad idea and can often lead to unhappy results such as crashes or program malfunction. This can also be a difficult bug to track down.

Unlike BASIC or JAVA, the C compiler does no checking to see if array access is within legal bounds of the array size that you have declared.

给数组赋值

```
mySensVals[0] = 10;
```

从数组中查找一个值

```
x = mySensVals[4];
```

Arrays and FOR Loops（数组和循环语句）

数组经常用于循环语句中，循环计数器用每个数组元素的索引值来表示。例如，通过串口打印一个数组的元素，如下：

```
int i;
for (i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

类型转换

char()

Description:

转换一个值为字符类型

Syntax

char(x)

Parameters:

x: 一个任意类型的值

Returns:

char

byte()

Description:

转换一个值为字节类型

Syntax

byte(x)

Parameters:

x: 一个任意的值

Returns:

byte

int()

Description:

转换一个值为整型

Syntax

int(x)

Parameters:

x: 一个任意的值

Returns:

int

word()

Description:

转换一个值为字类型或者从两个字节中创建一个字。

Syntax

word(x)

word(h, l)

Parameters:

x: 一个任意类型的值

h:字类型的高位字节（最左边的）

l:字类型的低位字节（最右边的）

Returns:

word

long()

Description:

转换一个值为长整型

Syntax

long(x)

Parameters:

x: 一个任意的值

Returns:

long

float()

Description:

转换一个值为浮点型

Syntax

float(x)

Parameters:

X：一个任意的值

Returns:

float

变量作用域和限定符

全局变量通常在函数外部声明，可以被所有函数调用。局域变量仅在某个函数内部声明，并且只能用于该函数。在 Arduino 环境中，所有在函数外部声明的变量为全局变量。当程序很庞大很复杂时，局域变量是一种有效的方式来确保只有一个函数可以调用其内部的变量。

当一个函数不注意地用另一个函数修改变量时，这种方式可以阻止编程错误。

Example::

```
int gPWMval;  // 任何函数都可以调用该变量
void setup()
{
  // ...
}

void loop()
{
  int i;    // "i" 是 "loop"函数的内部变量
  float f;  // "f" 是 "loop"函数的内部变量
  // ...

  for (int j = 0; j <100; j++){
  // j 为 for 循环内部的局域变量
  }

}
```

const keyword

const关键字代表常量。它是一个变量限定符，限定一个变量为只读。如果你试图赋值给const变量，就会产生编译错误。

Example:

```
const float pi = 3.14;
float x;
```

Functions（函数）

Digital I/O---数字输入输出

pinMode()

Description:

配置具体的引脚为输入或者输出状态。

Syntax：

pinMode(pin, mode)

Parameters:

pin: 你想设置的引脚号

mode: INPUT, OUTPUT, or INPUT_PULLUP.

Returns:

None

Example:

```
int  ledPin  =  13;                // LED connected to digital pin 13
void  setup()
{
    pinMode(ledPin,  OUTPUT);      // sets the digital pin as output
}
void  loop()
{
    digitalWrite(ledPin,  HIGH);   // sets the LED on
    delay(1000);                   // waits for a second
    digitalWrite(ledPin,  LOW);    // sets the LED off
    delay(1000);                   // waits for a second
}
```

digitalWrite()

Description:

Write a HIGH or a LOW value to a digital pin.
设置数字引脚为 HIGH 或者 LOW.
如果引脚被 pinMode()配置为 OUTPUT 状态，电压将会被设置为相关的值：高电平 5v（或者 3.3v），低电平 0v。
如果引脚配置为 INPUT,用 digitalWrite()设置为 HIGH 将会使能内部的上拉电阻 20K。设置为低电平时将禁止内部的上拉电阻。这个上拉电阻足够暗淡地点亮一个 LED 灯，所以如果 LED 灯亮了但是比较暗的话，这很可能就是原因所在。补救措施是设置引脚为输出状态。

Syntax：

digitalWrite(pin, value)

Parameters:

pin: the pin number

value: HIGH or LOW

Returns:

none

Example:

```
 int ledPin = 13;                // LED connected to digital pin 13
void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}
void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

digitalRead()

Description:

从一个数字引脚中读取数值：高电压或者低电压。

Syntax

digitalRead(pin)

Parameters:

pin: 你想读取的引脚号 (*int*)

Returns:

HIGH or LOW

Example:

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
  pinMode(inPin, INPUT);      // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);    // sets the LED to the button's value
}
```

Analog I/O---模拟输入输出

analogReference(type)

Description:

Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options are:

配置用于模拟输入的参考电压。选项如下：

DEFAULT(默认)：Arduino 板默认的参考电压是 5V 或者 3.3V。

INTERNAL(内部的)： 在 ATmega168 或者 ATmega32 板中的内部参考电压为 1.1v,在 ATmega8 中内部的参考电压为 2.56v.

INTERNAL1V1（内部的 1.1Vz）:内部的 1.1v 参考电压（仅适用于 *Arduino Mega*）

INTERNAL2V56(内部的 2.56v):内部的 2.56v 参考电压(仅适用于 *Arduino Mega*)

EXTERNAL（外部的）：应用于引脚 AREF 的电压（0~5v）为参考电压。

Returns:

None.

Note:

在改变了模拟参考电压之后，来自 analogRead()函数的前几个读数也许会不准确。

Warning:

不要把任何小于 0V 或者大于 5v 的电压加在外部参考电压 AREF 引脚上。如果你在 AREF 引脚上加一个参考电压，你必须在调用 **analogRead()函数之前设置模拟参考为 EXTERNAL.**

或者，你可以通过一个 5K 的电阻连接外部参考电压到 AREF 引脚上，这样可以允许你在外部和内部参考电压之间切换。注意：电阻会改变用作参考的电压因为有一个内部的 32K

的电阻在 AREF 引脚上。两个电阻充当分压器，所以，举个例子来说，2.5V 的电阻在 AREF 引脚上会产生 2.5 * 32 / (32 + 5) = ~2.2V 的电压。

analogRead()

Description:

该函数主要作用是从具体的模拟引脚中读取数值。Arduino 板包含一个 6 通道（在 Mini 和 Nano 板中为 8 通道，Mega 板中为 16 通道），10 位的模数转换器。这意味着 0~5V 的输入电压映射为 0~1023 的整数值。结果是每单位 5/1024v 或者每单位 0,0049v。输入范围和精度可以通过 analogReference()函数修改。读一个模拟引脚占用大约 100um 的时间，所以最大读取速率大约是 10000 次/秒。

Syntax:

analogRead(pin)

Parameters:

pin: 模拟输入的引脚号(大多数板上是 0 to 5 号引脚， Mini 和 Nano 板上是 0 to 7 , Mega 板上是 0 to 15)

Returns:

int (0 to 1023)

Note:

If the analog input pin is not connected to anything, the value returned by will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

如果模拟输入引脚没接任何东西的话，analogRead()函数返回的值将会基于某种因素而波动（比如，其他模拟引脚的值，手离板子的远近……）

Example:

```
int analogPin = 3;
int val = 0;                // 存储读取值的变量
void setup()
{
  Serial.begin(9600);             //  设置串口
}
void loop()
{
  val = analogRead(analogPin);     // 读取输入引脚的值
  Serial.println(val);               // 串口打印
}
```

analogWrite()

Description:

该函数的作用是：给一个引脚写入模拟值（PWM 方波的形式）。可以让一个 LED 灯忽明忽暗或者变速驱动一个电机。在调用 **analogWrite()函数之后，引脚上会产生一定占**

空比的稳定方波直到 analogWrite()函数的下一次调用。PWM 信号的频率大约是 490HZ.

在大多数的 Arduino 板（基于 ATmega168 或者 ATmega328）上，这个函数将工作在引脚 3,5,6,9,10 和 11 上。在其他的 Mega 板上，该函数工作在 2~13 上。基于 ATmega8 的老板子仅仅在 9,10,11 引脚上支持 analogWrite()。

Arduino Due 板在引脚 2~13、DAC0 和 DAC1 上支持 analogWrite()函数。和 PWM 引脚不一样。DAC0 和 DAC1 是数模转换器，并且担当实际模拟输出的任务。

在调用 analogWrite()函数之前不需要调用 pinMode()来设置引脚为输出模式。analogWrite()函数跟 *analogRead*（）函数和模拟引脚毫无关系。

Syntax:

analogWrite(pin, value)

Parameters:

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

Returns:

nothing

Notes and Known Issues

在引脚 5 和 6 上产生的 PWM 输出将会有着高于预期的占空比。这是因为 mills()函数和 delay()函数的相互作用导致，两个函数共享产生 PWM 输出的内部计时器。关于这个问题需要在低占空比的设置时注意。并且这也许会导致引脚 5 和 6 上的零电位不为零。

Example:

设置 LED 灯的输出和电位器的值成正比。

```
int ledPin = 9;       // LED connected to digital pin 9
int analogPin = 3;    //电位计连接到模拟引脚 3
int val = 0;          // 存储读取值的变量
void setup()
{
  pinMode(ledPin, OUTPUT);
}
void loop()
{
  val = analogRead(analogPin);
  analogWrite(ledPin, val / 4);  // 模拟引脚读取的值从 0 到 1023 变化，模拟输出的值从 0 到 255 变化
}
```

Advanced I/O---高级 IO

tone()

Description:

该函数的作用是在一个引脚上产生一个指定频率的方波（50%的占空比）。需要指定它的持续时间，否则的话这个方波将会一直持续直到调用 noTone()函数为止。这个引脚可以连接一个压电式蜂鸣器或者其他的扩音器来发声。

一次只能产生一个音调。如果一个音调正在另一个引脚上发出，函数 tone()的调用将会失效。如果是在同一个引脚上，tone()函数的再次调用会设置它的频率。tone()函数的调用将会干扰引脚 3 和 11 上的 PWM 输出(除了 Mega 板以外的板)。

NOTE: 如果你想在多个引脚上发出不同音高的声音，你需要在下一个引脚调用 tone()函数之前在此引脚上调用 noTone()函数。

Syntax

tone(pin, frequency)

tone(pin, frequency, duration)

Parameters:

pin:产生音调的引脚

frequency:音调的频率（单位：Hz） - *unsigned int*

duration:音调的持续时间（mm）(可选的)- *unsigned long*

Returns:

nothing

noTone()

Description:

该函数的作用是：终止 tone()函数触发的方波信号。如果没有音调产生的话也不会有任何影响。

Syntax

noTone(pin)

Parameters:

pin: 终止音调的引脚

Returns:

nothing

pulseIn()

Description:

该函数的作用是在某一个引脚上读取脉冲（高电平或者低电平）。例如，如果是高电平，pulseIn()函数等待引脚变为高电平并且开始计时，然后等到引脚变为低电平并且停止计时。以微秒的形式返回脉冲的长度。如果在一个指定的时间内没有检测到脉冲，将会放弃检测 并且返回零值。这个函数的时间是依靠经验决定的，因此在长脉冲中很可能产生一些错误。这个时间大约为 10 微秒到 3 分钟。

Syntax

pulseIn(pin, value)

pulseIn(pin, value, timeout)

Parameters:

pin: 你想读脉冲的引脚号. (*int*)

value: 读取的脉冲类型: either HIGH or LOW. (*int*)

timeout (optional): 等待脉冲开始的微妙值; 默认的是 1s (*unsigned long*)

Returns:

脉冲的长度(in microseconds) 或者在时间周期之前如果没有脉冲的话就会返回零值 (*unsigned long*)

Example:

```
int pin = 7;
unsigned long duration;

void setup()
{
  pinMode(pin, INPUT);
}

void loop()
{
  duration = pulseIn(pin, HIGH);
}
```

Time---时间函数

millis()

Description:

该函数的作用是在 Arduino 板开始运行当前的程序后返回毫秒值。在大约 50 天后这个值将会溢出（回到零）

Parameters:

None

Returns:

自程序开始后的毫秒值 (*unsigned long*)

Example:

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = millis();
  //prints time since program started
```

```
  Serial.println(time);
  // 为了放慢传输速度延时 1s.
  delay(1000);
}
```

micros()

Description:

该函数的作用是在 Arduino 板开始运行当前的程序后返回微秒值。在大约 70 分钟后这个值将会溢出（回到零）。在 16MHz 的 Arduino 板上（例如 Duemilanove 和 Nano），这个函数有 4 微秒的分辨率。（例如，返回值总是 4 的倍数）

在 8MHz 的 Arduino 板上（例如 LilyPad），这个函数有 8 微秒的分辨率。

Parameters:

None

Returns:

自程序开始后的毫秒值(*unsigned long*)

Example:

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = micros();
  //prints time since program started
  Serial.println(time);
  //为了放慢传输速度延时 1s.
  delay(1000);
}
```

delay()

Description:

该函数的作用是中断程序一段指定的时间。

Syntax

delay(ms)

Parameters:

ms: 中断的毫秒数 (*unsigned long*)

Returns:

nothing

Example:

```
int ledPin = 13;                // LED connected to digital pin 13
```

```
void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

delayMicroseconds()

Description:

该函数的作用是中断程序一段指定的时间。1ms=1000us,1s=1000ms.
一般地，16383 为能产生的精确最大值。这个在以后的 Arduino 版本中会变化。如果中断时间长于几千微秒，应当调用 delay()函数。

Syntax

delayMicroseconds(us)

Parameters:

us: 中断的微秒数 (*unsigned int*)

Returns:

None

Example:

```
int outPin = 8;                 // digital pin 8

void setup()
{
  pinMode(outPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(outPin, HIGH);   // sets the pin on
  delayMicroseconds(50);        // pauses for 50 microseconds
  digitalWrite(outPin, LOW);    // sets the pin off
  delayMicroseconds(50);        // pauses for 50 microseconds
}
```

Math---数学函数

min(x, y)

Description:

计算两个数中的较小值

Parameters:

x: the first number, any data type

y: the second number, any data type

Returns:

The smaller of the two numbers.

Example:s

```
sensVal = min(sensVal, 100); // assigns sensVal to the smaller of sensVal
or 100
                                // ensuring that it never gets above 100.
```

max(x, y)

Description:

计算两个数中的较大值

Parameters:

x: the first number, any data type

y: the second number, any data type

Returns:

The larger of the two parameter values.

Example:

```
sensVal = max(senVal, 20); // assigns sensVal to the larger of sensVal
or 20
                            // (effectively ensuring that it is at least
20)
```

abs(x)

Description:

计算某个数的绝对值。

Parameters:

x: the number

Returns:

x: if **x** is greater than or equal to 0.

-x: if **x** is less than 0.

Warning

由于函数 abs()的计算方式，应用中应该在括号内部应该避免用其他的函数，这也许会导致不正确的结果。

```
abs(a++);   // 避免用这种方式- 产生不正确的结果

a++;        // 应该用这种方式代替。
abs(a);     // 应当保持其他的公式在函数外部。
```

constrain(x, a, b)

Description:

限制一个数在某个范围之内。

Parameters:

x: the number to constrain, all data types

a: the lower end of the range, all data types

b: the upper end of the range, all data types

Returns:

x: if **x** is between **a** and **b**

a: if **x** is less than **a**

b: if **x** is greater than **b**

Example:

```
sensVal = constrain(sensVal, 10, 150);
// limits range of sensor values to between 10 and 150
```

map(value, fromLow, fromHigh, toLow, toHigh)

Description:

该函数的作用是重新映射一个数值的领域到另一个领域。也就是说，一个 fromLow 值将被映射为 toLow，一个 fromHigh 值将被映射为 toHigh，中间值仍然为中间值。

Parameters:

value: the number to map
fromLow: the lower bound of the value's current range
fromHigh: the upper bound of the value's current range
toLow: the lower bound of the value's target range
toHigh: the upper bound of the value's target range

Returns:

The mapped value.

Example:

```
/* Map an analog value to 8 bits (0 to 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);
```

```
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

sqrt(x)

Description:

计算一个数的平方根。
Parameters:
x: the number, any data type
Returns:

double, the number's square root.

Trigonometry---三角函数

sin(rad)

Description:

计算一个角度的正弦值。结果为-1 到 1 之间。
Parameters:

rad: 弧度表达的角度值(*float*)
Returns:

角度的正弦值(*double*)

cos(rad)

Description::

计算一个角度的余弦值。结果为-1 到 1 之间。
Parameters:

rad: 弧度表达的角度值(*float*)
Returns:

角度的余弦值(*double*)

tan(rad)

Description::

计算角度的正切值。结果在负无穷到正无穷。
Parameters:

rad: 弧度表达的角度值(*float*)
Returns:

角度的正切值(*double*)

Random Numbers---随机数

randomSeed(seed)

Description:

randomSeed() initializes the pseudo-random number generator, causing it to start at an arbitrary point in its random sequence. This sequence, while very long, and random, is always the same.

If it is important for a sequence of values generated by random() to differ, on subsequent executions of a sketch, use randomSeed() to initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.

Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling randomSeed() with a fixed number, before starting the random sequence.

Parameters:

long, int - pass a number to generate the seed.

Returns:

no returns:

Example:

```
long randNumber;

void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop(){
  randNumber = random(300);
  Serial.println(randNumber);

  delay(50);
}
```

random()

Description:：

该随机函数产生伪随机数。

Syntax：

random(max)

random(min, max)

Parameters:

最小值-随机数的下限值，包括这个最小值（可选的）

最大值-随机数的上限值，不包含这个最大值

Returns:

在最小值到最大值减一的随机数（long）

Note:

如果 **random**（）函数产生的一系列值非常重视互异性，在后面的程序执行中用 **randomSeed()**函数相当随机的输入来初始化随机数发生器，例如在一个未连接的引脚上用 **analogRead()**函数读取的值。

Example:

```
long randNumber;

void setup(){
  Serial.begin(9600);

  // if analog input pin 0 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  randomSeed(analogRead(0));
}

void loop() {
  // print a random number from 0 to 299
  randNumber = random(300);
  Serial.println(randNumber);

  // print a random number from 10 to 19
  randNumber = random(10, 20);
  Serial.println(randNumber);

  delay(50);
}
```

Bits and Bytes---位和字节

lowByte()

Description:：

提取变量的低位字节（例如一个字变量）

Syntax：

lowByte(x)

Parameters:

x: a value of any type

Returns:

byte

highByte()

Description:：

提取一个字变量的高字节（最左边的）（或者一个更大的数据类型的次低位字节）

Syntax

highByte(x)

Parameters:

x: a value of any type

Returns:

byte

bitRead()

Description:：

读取一个数据位

Syntax：

bitRead(x, n)

Parameters:

x:所要读取的数据值

n: 要读的位数，从最右边也就是最低有效位开始的是零位

Returns:

the value of the bit (0 or 1).

bitWrite()

Description:：

给一个数字变量的二进制位赋值

Syntax：

bitWrite(x, n, b)

Parameters:

x: 所要赋值的数字变量

n: 所要赋值的二进制位，从最右边的最低有效位开始为第零位。

b: 赋给二进制位的值 (0 or 1)

Returns:

none

bitSet()

Description:：

给一个数字变量的二进制位置位。

Syntax：

bitSet(x, n)

Parameters:

x: 所要置位的数字变量

n: 所要设置的位从最右边的最低有效位开始为第零位。

Returns:

none

bitClear()

Description:：

给一个数字变量的某个二进制位清零

Syntax：

bitClear(x, n)

Parameters:

x: 所要清零的数字变量

n: 所要清零的位,从最右边的最低有效位开始为第零位。

Returns:

none

bit()

Description:：

计算具体二进制位的十进制数（0 位为 1,1 位为 2，二位为 4）

Syntax：

bit(n)

Parameters:

n: 所要计算的二进制位

Returns:

十进制数值

External Interrupts---外部中断

attachInterrupt()

Description:：

当一个外部中断发生时，调用某个具体的函数。用任何以前附加到中断中的函数来替代。

大多数的 Arduino 板有两个外部中断：0 号中断（在数字引脚 2 上）和 1 号中断（在数字引脚 3 上）。下面的表展示了一些在各种各样的板子上可以获得的中断引脚。

Board	**int.0**	**int.1**	**int.2**	**int.3**	**int.4**	**int.5**
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1		
Due	**(see below)**					

Arduino Due 板有很强大的中断能力，允许你在所有可获得的引脚上附加一个中断函数。你可以直接在 attachInterrupt()函数中指定引脚号。

Syntax：

attachInterrupt(interrupt, function, mode)

attachInterrupt(pin, function, mode) (*Arduino Due only*)

Parameters:

interrupt: 中断号 (*int*)

pin: **引脚号** **(*Arduino Due only*)**

function: *当中断发生时调用的函数；这个函数不能有任何参数和返回值。这个函数有时被当做一个中断服务程序。*

mode: 定义中断何时出发。四个常量被定义为有效值。

LOW：当引脚为低电平时，触发中断。
CHANGE：当引脚的值改变时触发中断。
RISING：当引脚的电平由低到高时触发。
FALLING：当引脚的电平由高到低时触发。

HIGH：当引脚为高电平时触发中断（仅适用于 *Arduino Due 板*）

Returns:

none

detachInterrupt()

Description::

关闭给定的中断。

Syntax:

detachInterrupt(*interrupt*)

detachInterrupt(*pin*) (*Arduino Due only*)

Parameters:

- *interrupt*: 所要禁止的中断号(see attachInterrupt() for more details).
- *pin*: 所要禁止中断的引脚号 (*Arduino Due only*)

Interrupts---中断

interrupts()

Description::

重新使能中断（在中断被 noInterrupts()函数禁止之后）。中断允许某个重要的任务在后台运行并且被默认使能。当中断禁止后，某些函数将会停止工作并且输入通信也许会被忽略。中断可能轻微地干扰代码的运行时间，甚至会关闭某些关键代码段。

Parameters:

None

Returns:

None

Example:

```
void setup() {}

void loop()
{
  noInterrupts();
  //临界的时间敏感的代码区
  interrupts();
  // other code here
}
```

noInterrupts()

Description::

关闭中断（你可以用 interrupts()函数重新使能）。中断允许某个重要的任务在后台运行并且默认使能。当中断禁止后某些函数会停止运行，并且输入通信也许会被忽略。中断可能轻微地干扰代码的运行时间，甚至会关闭某些关键代码段。

Parameters:

None.

Returns:

None.

Example:

```
void setup()
{}
void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Communication---通信

Serial

Serial 函数用于 Arduino 板和电脑或者其他的设备通信。所有的 Arduino 板至少有一个串口（也叫做 UART 或者 USART）:Serial.这个串口通过 USB 与电脑连接，用数字引脚 0（RX）和 1（TX）通信。因此，如果你用这个函数，你不能同时把引脚 0 和 1 用作数字输入输出。

你可以用 Arduino 环境下自带的串口监视程序和另一个 Arduino 板通信。在工具栏中点击串口监视按钮并且在调用 begin()函数时选择相同的波特率。

Arduino Mega 板有三个额外的串口：19(RX) 和 18 (TX)号引脚上的 **Serial1** 、, 17 (RX) 和 16 (TX)号引脚上的 **Serial2** 还有 15 (RX)和 14 (TX)号引脚上的 **Serial3** 。

为了用这三个引脚和你的个人电脑通信，你需要一个 USB 转串口的适配器，因为它们没有和 Mega 板的 USB 转串口连接。为了用这三个引脚和一个外部的 TTL 串口设备通信，连接 TX 到设备上的 RX 引脚、连接 RX 到设备的 TX 引脚并且 Mega 板的地线接到设备的地线上。（不能直接把这三个引脚接到 RS232 串口上；RS232 的运行电压为+/- 12V 并且会损坏 Arduino 板。）

Arduino Due 板有三个额外的 3.3V TTL 串口：19(RX) 和 18 (TX)号引脚上的 **Serial1** 、, 17 (RX) 和 16 (TX)号引脚上的 **Serial2** 还有 15 (RX)和 14 (TX)号引脚上的 **Serial3** 。引脚 0 和 1 也连接到 ATmega16U2 板的 USB 转 TTL 串口芯片的相关的引脚上了，用于 USB 调试端口。另外，在 SAM3X 芯片上有一个本地的 USB 转串口端口 *SerialUSB*'.

Arduino Leonardo 板通过引脚 0（RX）和 1（TX）上的 TTL(5V)电压用 Serial1 来通信。**Serial 函数留作 USB CDC 通信。**