Arduino Programming Part 3

ME 120

Mechanical and Materials Engineering

Portland State University

Overview

Variable Declarations

Variable Assignments

Built-in I/O functions

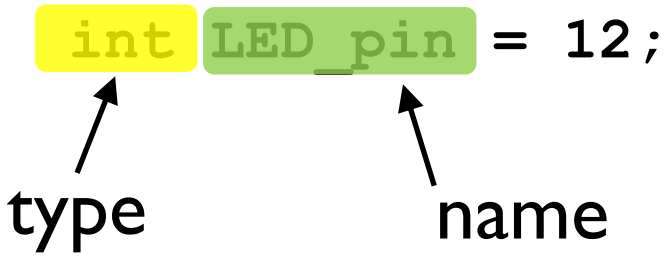
See on-line reference:

http://arduino.cc/en/Reference/HomePage

Variables in Arduino programs

Variables are containers

A variable has a name and a type



Common types:

- `int`, `unsigned int`
- `long`, `unsigned long`
- `float`
- `char`
- `byte`

Variable names:

- Start with a letter (a-z, A-Z, _)
- Can contain numbers
- Cannot contain +, -, =, /, *

see https://www.arduino.cc/reference/en/#variables

Basic types of variable

Three basic categories of variables

- integers: `int` and `long`
- floating point values: `float` and `double` on some boards
- character strings: `char`, `String`

Integers

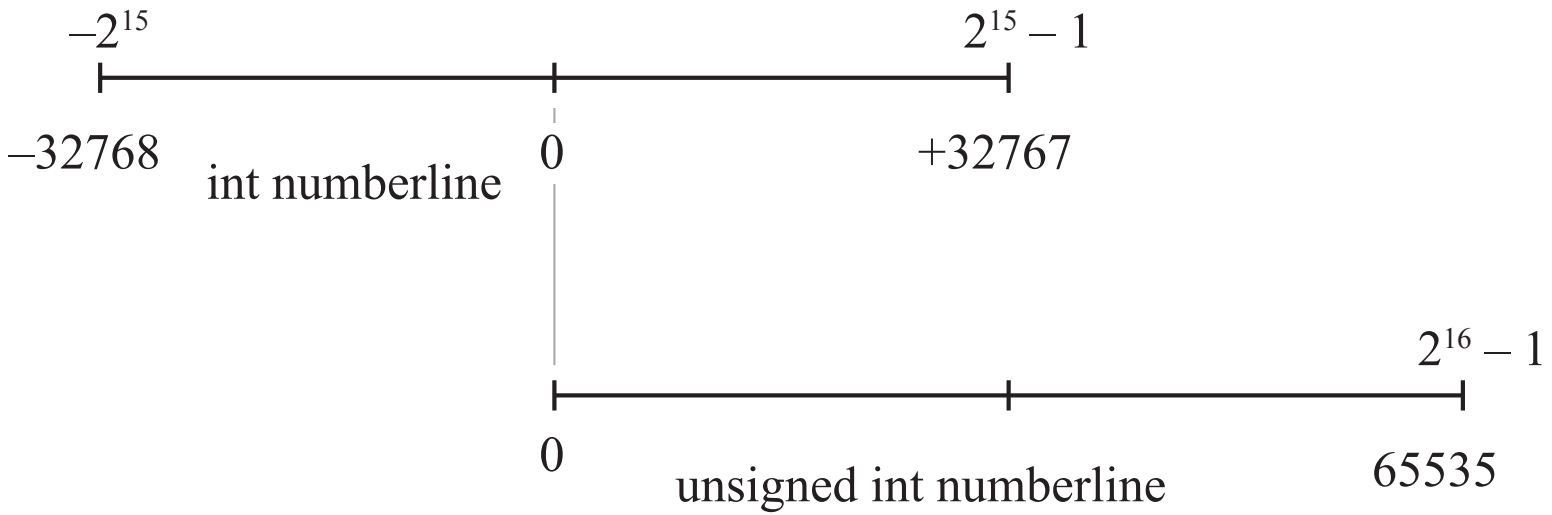
- No fractional part. Examples: 1, 2, 23, 0, –50213
- Used for counting and return values from some built-in functions
- Integer arithmetic results in truncation to integers

Floating point numbers

- Non-zero fractional parts. Examples 1.234, –2.728, 4.329×10^{-4}
- Large range of magnitudes
- Floating point arithmetic does not truncate, but has round-off

`int` Integer types

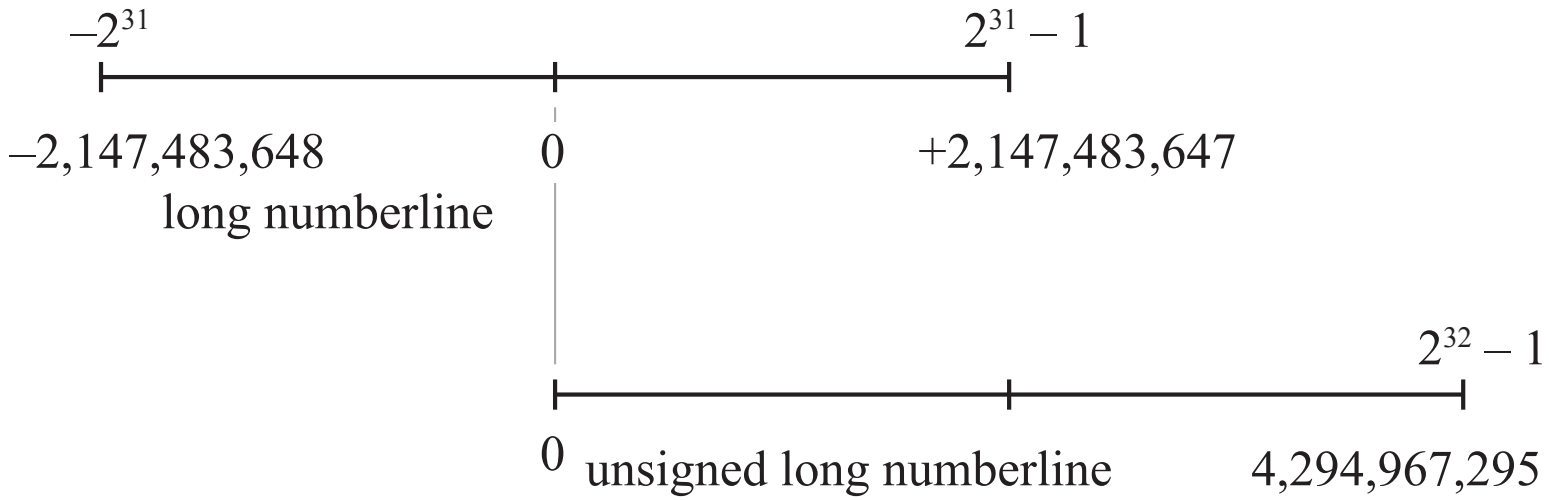
`int`	integer in the range –32,768 to 32,767
`unsigned int`	positive integer in the range 0 to 65,535



See http://arduino.cc/en/Reference/Int and http://arduino.cc/en/Reference/Long

`long` Integer types

`long`	integer in the range –2,147,483,648 to 2,147,483,647
`unsigned long`	positive integer in the range 0 to 4,294,967,295



See http://arduino.cc/en/Reference/Int and http://arduino.cc/en/Reference/Long

Floating point types

`float` values with approximately seven significant digits in the range $\pm(1.80 \times 10^{-38}$ to $3.40 \times 10^{38})$

`double` values with approximately thirteen significant digits in the range $\pm(2.2 \times 10^{-308}$ to $1.80 \times 10^{308})$

There is no double on an Arduino UNO. On an UNO, a double is the same as a float. On more advanced microcontroller boards, the double type has the range indicated above.

The Adafruit Feather nRF52840 Sense has built-in support for single precision floating point arithmetic, i.e. variables of type "float"

See http://arduino.cc/en/Reference/Float and http://arduino.cc/en/Reference/Double

Declaring and assigning values

Declarations are necessary. Assignments are optional.

```
int n;                  //  single declaration
int i,j,k,n;            //  multiple declaration
int i=5;                //  single declaration and assignment
int i=5, j=2;           //  multiple declaration and assignment

float x;
float x,y,z;
float x=0.0, y=-1.23e5;   //  assignment with ”e” notation
```

Note:

- Integer values do not use decimal points
- Floating point values can use “e” notation
  - 1.23e5 is equal to 1.23×10^5
  - DO NOT write x = 1.23*10^5, use x = 1.23e5 instead

Use = to assign values

The equals sign is the *assignment operator*

- The statement x = 3 assigns a value of 3 to x.
  - The actual operation involves storing the value 3 in the memory location that is reserved for x.
- The equals sign *does not mean* that x and 3 are the same!

Symbolically you can replace x = 3 with x ← 3.

Assigning values

Consider the following sequence of statements

```
x = 3;
y = x;
x = 5;
```

The preceding statements are executed in sequence.

- The last assignment determines the value stored in x.
- There is no ambiguity in two “x = ” statements:
  - x = 3 stores the value of 3 into the memory location named x
  - x = 5 *replaces* the 3 stored in x with a new value, 5.

Test your understanding

What are the values of n and z at the end of the following sequences of statements?

```
int i,j,k,n;

i = 2;
j = 3;
k = i + 2*j;
n = k - 5;
```

n = ?

```
int i,j,k,n;

i = 2;
j = 3;
n = j - i;
n = n + 2;
```

n = ?

```
int n;
float x,y,z;

x = 2.0;
y = 3.0;
z = y/x;
n = z;
```

z = ?

n = ?

Test your understanding

What are the values of n and z at the end of the following sequences of statements?

```
int i,j,k,n;

i = 2;
j = 3;
k = i + 2*j;
n = k - 5;
```

```
int i,j,k,n;

i = 2;
j = 3;
n = j - i;
n = n + 2;
```

```
int n;
float x,y,z;

x = 2.0;
y = 3.0;
z = y/x;
n = z;
```

The n = n + 2; statement shows why it is helpful to think of the equal sign as a left facing arrow.

You can mentally replace n = n + 2; with n ← n + 2;

Integer arithmetic

We have to understand the rules of numerical computation used by Arduino hardware (and computers, in general).

Integer arithmetic always produces integers

```
int i,j;
i = (2/3)*4;
j = i + 2;
```

What values are stored in i and j?

Integer arithmetic

We have to understand the rules of numerical computation used by Arduino hardware (and computers, in general).

Integer arithmetic always produces integers

```
int i,j;
i = (2/3)*4;
j = i + 2;
```

What values are stored in i and j?

Answer: i ← 0, j ← 2

Integer arithmetic

Integer arithmetic always produces integers

```
int i,j;
i = (2.0/3.0)*4.0;
j = i + 2;
```

What values are stored in i and j?

Answer: i ← 2, j ← 4

Review Preceding Slides on Integer arithmetic

Code A:

```
int i,j;
i = (2/3)*4.0;
j = i + 2;
```

What values are stored in i and j?

Answer: i ← 0, j ← 2

Code B:

```
int i,j;
i = (2.0/3.0)*4.0;
j = i + 2;
```

What values are stored in i and j?

Answer: i ← 2, j ← 4

Floating point arithmetic

Floating point arithmetic preserves the fractional part of numbers, but it does so approximately

```
float w,x,y,z;
w = 3.0;
x = 2.0;
y = w/x;
z = y - 1.5;
```

What values are stored in y and z?

Floating point arithmetic

Floating point arithmetic preserves the fractional part of numbers, but it does so approximately

```
float w,x,y,z;
w = 3.0;
x = 2.0;
y = w/x;
z = y - 1.5;
```

What values are stored in y and z?

Answer: $y \leftarrow 1.5$, $z \leftarrow 0$

Floating point arithmetic

Consider this alternate test*

```
float w,x,y,z;
w = 4.0/3.0;
x = w - 1;
y = 3*x;
z = 1 - y;
```

*See, e.g. C. Moler, *Numerical Computing in MATLAB*, 2004, SIAM, p. 38

Floating point arithmetic

Consider this alternate test*

```
float w,x,y,z;
w = 4.0/3.0;
x = w - 1;
y = 3*x;
z = 1 - y;
```

which produces x = 0.333 and y = 1.000 and z = –1.19e-7

z is not exactly zero because of roundoff

*See, e.g. C. Moler, *Numerical Computing in MATLAB*, 2004, SIAM, p. 38

Global and local variables

In this sketch, **`LED_pin`** is a global variable, accessible to other functions in the file

```
int LED_pin = 13;

void setup() {
  pinMode( LED_pin, OUTPUT );
}

void loop() {
  digitalWrite( LED_pin, HIGH );
  delay(1000);
  digitalWrite( LED_pin, LOW );
  delay(1000);
}
```

In this sketch, **`LED_pin`** is a local variable in the setup function. **`LED_pin`** is not accessible to the code in the loop function. *This sketch will not compile. It will not run*.

```
void setup() {
  int LED_pin = 13;
  pinMode( LED_pin, OUTPUT );
}

void loop() {
  digitalWrite( LED_pin, HIGH );
  delay(1000);
  digitalWrite( LED_pin, LOW );
  delay(1000);
}
```

In general, it is wise to avoid global variables unless it is absolutely necessary. In this example, **`LED_pin`** must be accessible to both **`setup`** and **`loop`**, so it must be a global variable.

Code Interlude:

Getting messages from the Arduino board in the Serial Monitor

Use these commands for serial communication with the host computer

`Serial.begin(speed)`

- Initializes the Serial port at specified **`speed`**. Typical speed is 9600

`Serial.print(value)`

- Sends **`value`** to the serial port
- **`value`** can be a single number or a character string
- *No newline* after **`value`** is sent

`Serial.println(value)`

- Sends **`value`** to the serial port
- **`value`** can be a single number or a character string
- *Add a newline* after **`value`** is sent

See https://www.arduino.cc/reference/en/language/functions/communication/serial/print/

Wait for the USB connection

Early Arduino boards had simpler USB interfaces

- On an Arduno UNO, opening the Serial Monitor would reset the connection
- Later boards, e.g. Feather nRF52840 Sense, provide full USB support, which slightly complicates use of the Serial Monitor

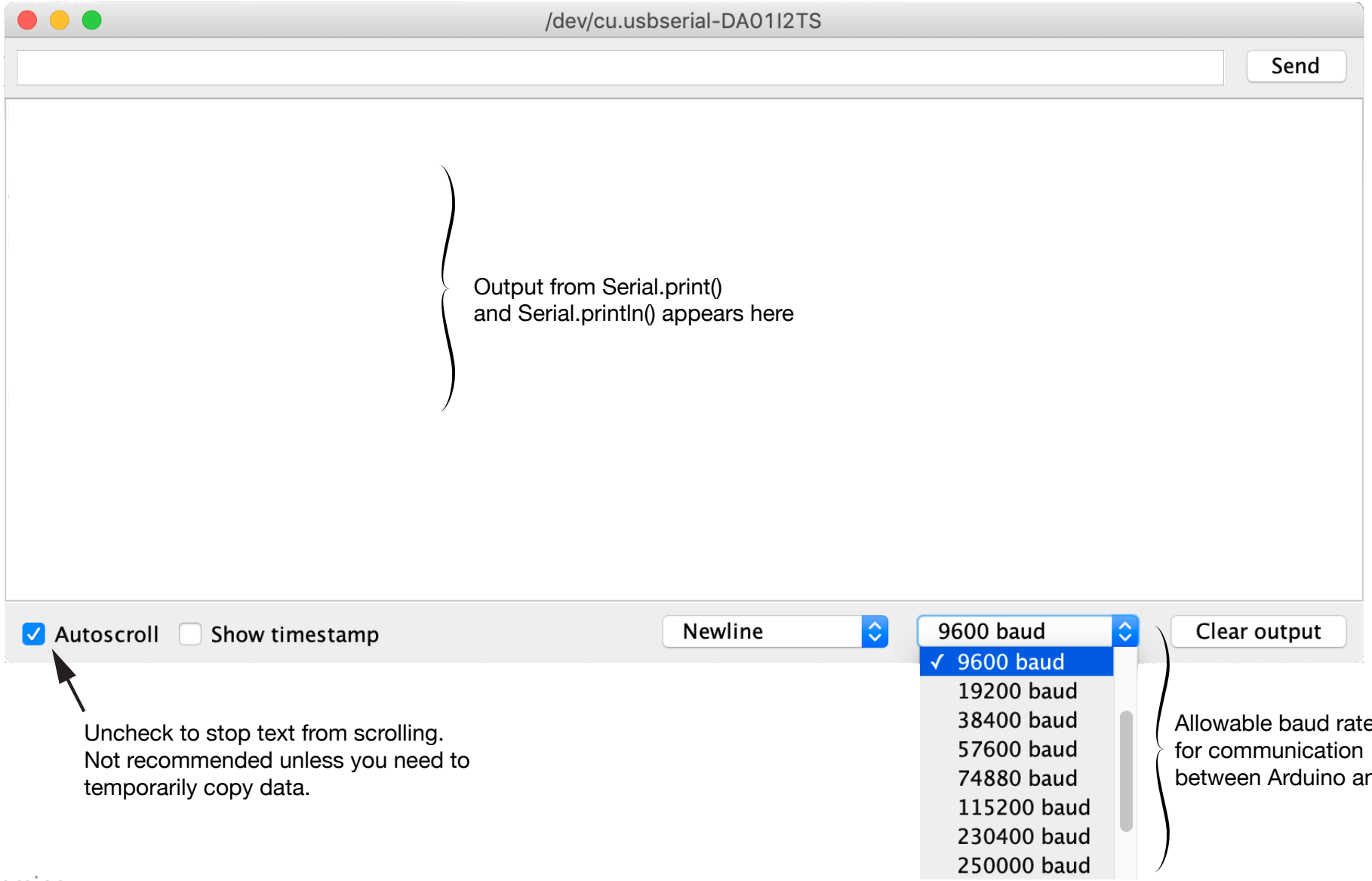
On the Feather nRF52840 Sense, you establish a Serial monitor connection with two lines of code

```
Serial.begin(nnnn);           // Set the baud rate for serial communication
while (!Serial) yield();      // Wait for serial port to connect
```

The value of **`nnnn`** must be the same as the setting in the Serial Monitor.

Typically we use **`Serial.begin(9600)`** or **`Serial.begin(115200)`**

Sample Serial Monitor Window



Demonstrate Serial.print and Serial.println

```
//  File:  demoSerialMonitor.ino
//
//  Show how to use the Serial Monitor and demonstrate different
//  behaviors of printing from setup() and loop()

void setup() {

  Serial.begin(115200);
  while (!Serial) yield();   //  Wait for Serial port to connect

  Serial.print("Here");    //  Message could be on one line. See loop()
  Serial.print(" in ");
  Serial.println(" setup()");

}

void loop() {

  Serial.println("Here in loop()");
  delay(2000);                        //  Slow down printing
}
```

Measure the time for USB start-up

```
//   File:  demoSerialStartupTime.ino
//
//   Measure the time to wait for the USB connection after the start of
//   an Arduino sketch.  Waiting for the USB to start is necessary for
//   Arduino boards with full USB support.  Without waiting, the first
//   few messages to the Serial Monitor will be lost.
//
//   This code simply times how long it takes the Serial object to
//   return a non-null response.  The while ( !Serial ) yield(); code
//   should be included after the Serial.begin() for all Arduino boards
//   with full USB support, e.g. the Feather nRF52840 Sense.
//
void setup() {

  unsigned long  tstart, tready;   //  storage for timing data

  tstart = millis();               //  start time
  Serial.begin(115200);            //  Initiate Serial object
  while ( !Serial ) yield();       //  Wait for USB to start
  tready = millis();               //  stop time

  Serial.print("Time waiting for USB start-up = ");
  Serial.print(tready - tstart);
  Serial.println(" milliseconds");
 }

void loop() {  }   //   Empty on purpose
```

Codes to demonstrate integer and floating-point arithmetic

Integer arithmetic

```
//   File:  int_test.ino
//
//   Demonstrate truncation with integer arithmetic

void setup() {
  int i,j;

  Serial.begin(9600);
  while (!Serial)  yield();   //  Wait for USB port to initialize

  // -- First example: Right hand side uses integer math and truncation
  //    occurs before the result is stored in variable i
  i = (2/3)*4;   //  result of evaluating (2/3) is zero
  j = i + 2;
  Serial.println("First test");
  Serial.print(i);  Serial.print("  ");   Serial.println(j);

  // -- Second example: Right hand side used floating point.  No truncation
  //    occurs until the result is store in variable i
  i = (2.0/3.0)*4.0;  //  result of evaluating (2.0/3.0) is 0.6666667
  j = i + 2;
  Serial.println("Second test");
  Serial.print(i);  Serial.print("  ");   Serial.println(j);
}

void loop() {}     // Loop does nothing.  Code in setup() is executed only once
```

Floating point arithmetic: test 1

```
//   File:  float_test.ino
//
//   Demonstrate floating point arithmetic computations that happen to
//   have no obvious rounding errors.  That DOES NOT always happen
//
//   Use two-parameter form of Serial.print.  The second parameter specifies
//   the number of digits in value sent to the Serial Monitor

void setup() {
  float w,x,y,z;

  Serial.begin(9600);
  while (!Serial) delay(10);   //  Wait for USB port to initialize

  // -- Computations that return results that you would expect; No rounding
  w = 3.0;
  x = 2.0;
  y = w/x;
  z = y - 1.5;
  Serial.println("Floating point arithmetic test");
  Serial.print(w,8);  Serial.print("  ");
  Serial.print(x,8);  Serial.print("  ");
  Serial.print(y,8);  Serial.print("  ");
  Serial.print(z,8);  Serial.print("  ");
  Serial.println(z*1.0e7,8);
}

void loop() {}     // Loop does nothing.  Code in setup() is executed only once
```

Floating point arithmetic: test 2

```
//   File:  float_test_2.ino
//
//   Demonstrate well-known round-off error problem with floating point arithmetic
//   See, e.g., Cleve Moler, Numerical Computing in MATLAB, p. 38
//
//   Use two-parameter form of Serial.print.  The second parameter specifies
//   the number of digits in value sent to the Serial Monitor

void setup() {
  float w,x,y,z;

  Serial.begin(9600);
  while (!Serial) delay(10);   //  Wait for USB port to initialize

  // -- Computations that show rounding
  w = 4.0/3.0;
  x = w - 1;
  y = 3*x;
  z = 1 - y;
  Serial.println("\nFloating point arithmetic test 2");
  Serial.print(w,8);  Serial.print("  ");
  Serial.print(x,8);  Serial.print("  ");
  Serial.print(y,8);  Serial.print("  ");
  Serial.print(z,8);  Serial.print("  ");
  Serial.println(z*1.0e7,8);
}

void loop() {}     // Loop does nothing.  Code in setup() is executed only once
```