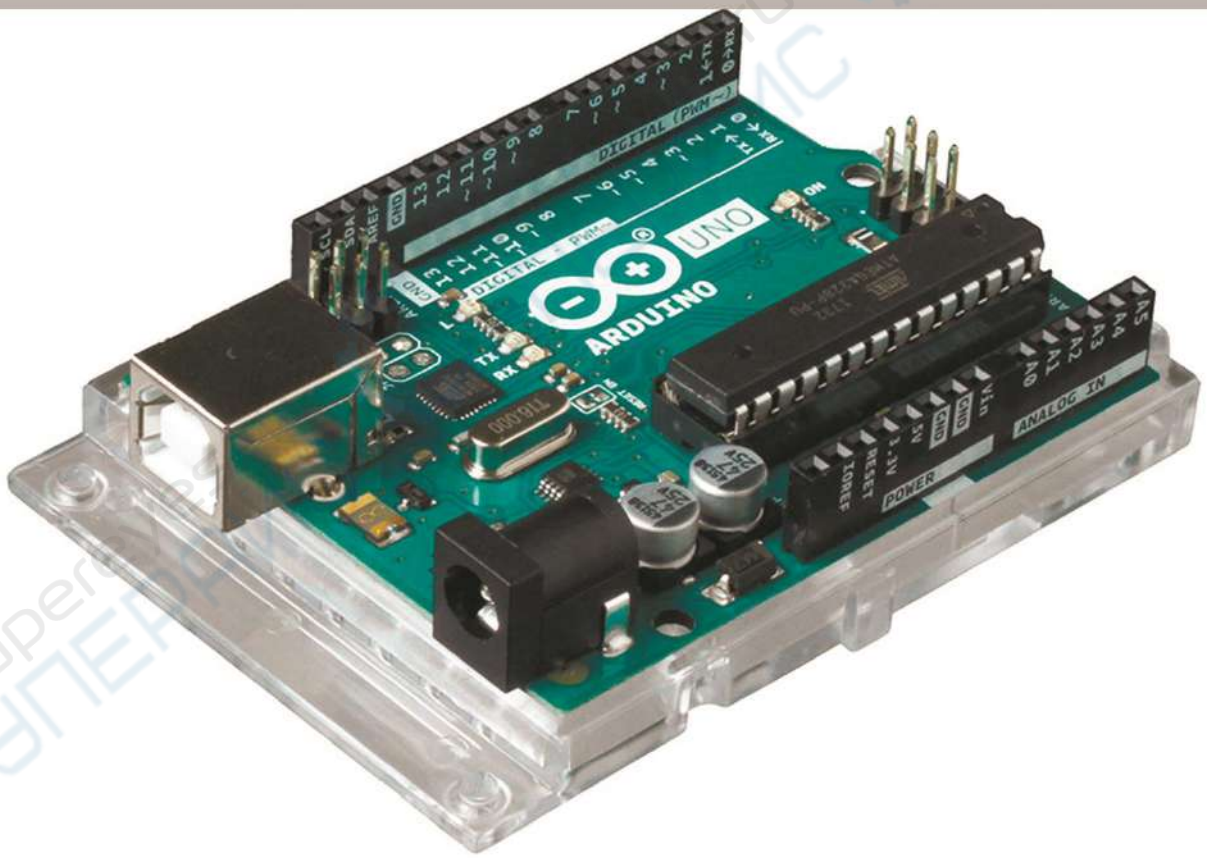
ARDUINO. НАЧАЛО



Оглавление

Как использовать инструкцию

Инструкция предназначена для использования в качестве шпаргалки. Не обязательно читать всё подряд, лучше заглянуть в оглавление и найти интересующий раздел. А ещё, перед каждым мини-уроком есть список ключевых слов. Это для того, чтобы можно было максимум времени уделить интересностям.

Обзор компонентов

Давайте сначала быстро пробежимся по компонентам из набора Arduino, чтобы разобраться что к чему.

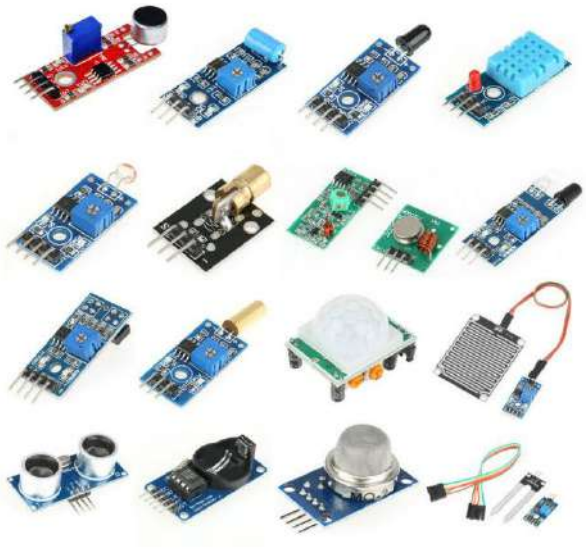
Arduino Uno

Штука, вокруг которой мы и будем крутиться. **Arduino Uno** считывает показания сенсоров, управляет актуаторами и делится результатами с людьми. Arduino Uno часто называют по-разному. Контроллер, плата, микроконтроллер - это всё про Arduino Uno.

Шилды Arduino

Шилд (**Shield**) - это способ прокачать Arduino. Каждый шилд даёт какую-нибудь супер-способность. Например, есть **Motor-shield**, который позволяет без проблем подключить моторы, есть **Ethernet-shield**, который соединяет плату с Интернетом, есть **Prototype-shield**, который ускоряет время сборки прототипов устройств в разы, много всякого есть.

Сенсоры Arduino



Сенсоры, или по-другому, **датчики**, помогают Arduino чувствовать мир. Разновидностей датчиков - целая куча. Хочешь - сенсор нажатия, хочешь - сенсор движения, а если очень хочешь, можно свой сенсор сделать.

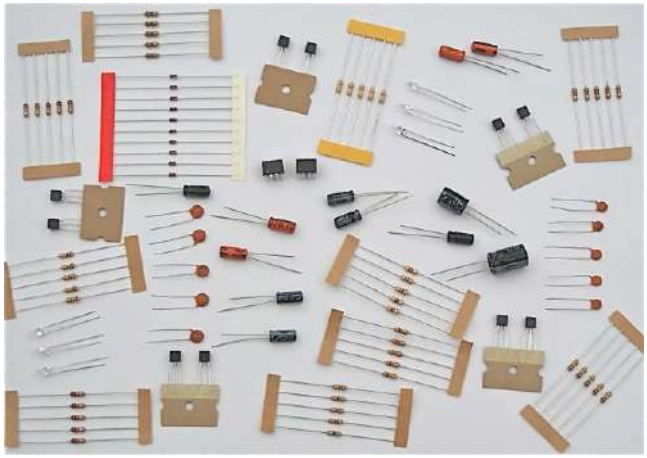
Актуаторы Arduino

Актуаторы помогают Arduino влиять на мир. Это всё, что крутится, светится, показывает, защелкивается и жужжит.

Соединения

Провода и **соединители** помогут подключить сенсоры и актуаторы к Arduino. Под разные типы соединений идут разные провода. В Arduino Uno используются соединения 2.54 mm. Их также называют **пинами**.

Электронные компоненты



Резисторы, **конденсаторы**, **микросхемы** и **транзисторы** помогут с решением задач на уровне электрических сигналов. Без них никуда. Arduino Uno - это куча транзисторов и резисторов, упакованных в красивую синеватую оболочку.

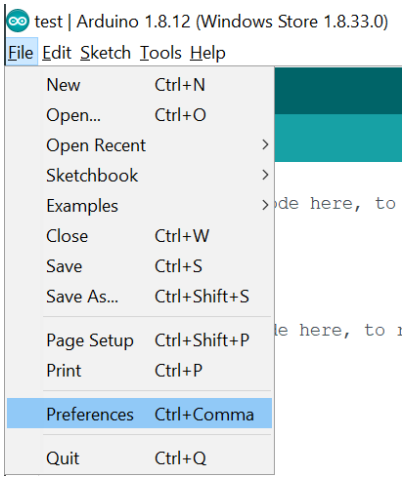
Как установить Arduino IDE

Для Arduino Uno можно писать полезные программы, которые будут служить инструкцией для контроллера. Хорошие новости - Arduino будет выполнять всё, что написано в программе. Плохие новости - Arduino будет выполнять всё, что написано в программе. Arduino можно программировать на разных языках программирования. Наш выбор - **C++**. Поддержим труды дяди Страуструпа.

Проблема микроконтроллера в том, что он не понимает C++, но понимает **машинный код**. У людей проблема ровно наоборот. Для решения проблемы придуманы **компиляторы**. Компилятор переводит C++ в понятный контроллеру язык и загружает машинные инструкции на плату. Хардкорные кодеры делают это вручную, а мы воспользуемся **Arduino IDE**.

Для этого идем на сайт arduino.cc, выбираем подходящую версию, и скачиваем Arduino IDE. Кстати, в настройках можно установить русский язык.

File -> Preferences -> Editor Language

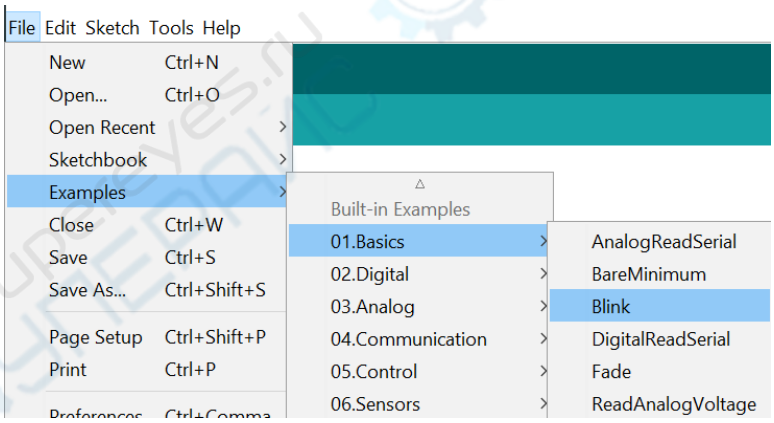


Если скачивать Arduino IDE не хочется, можно программировать в [Arduino Web Editor](), там нужно зарегистрироваться и скачать Arduino Create Plugin.

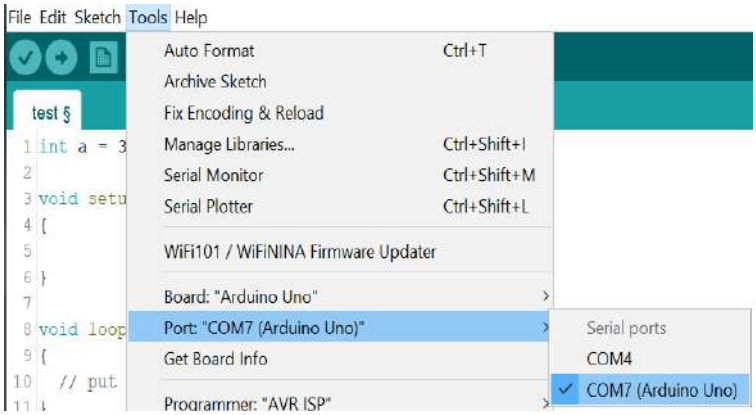
Как загрузить первую программу на плату

Так, Arduino IDE установили, теперь можно создать и загрузить первую программу. Обычно, когда изучаешь новый язык программирования, первым делом нужно запустить **“Hello World!”**. “Hello World!” мира Arduino - помигать светодиодом на плате. Так и поступим.

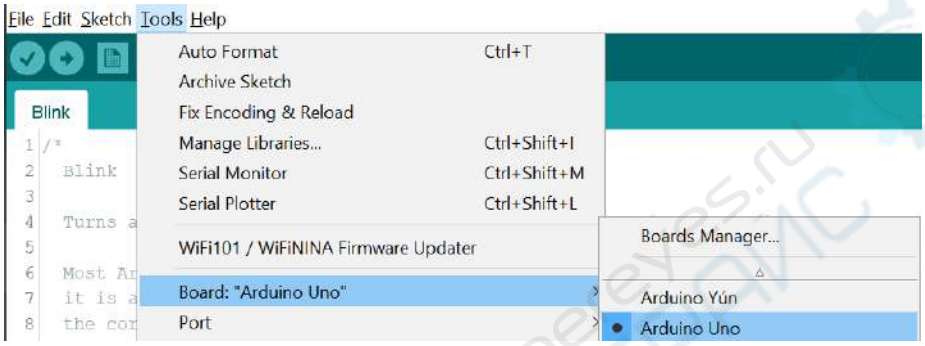
Сначала раздобудем код. В Arduino IDE есть готовые примеры с кодом. В любой непонятной ситуации можно воспользоваться кусочком хорошего кода. Нам нужен пример **Blink:**



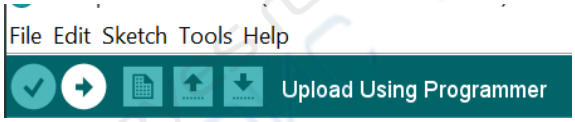
Теперь, когда код готов к запуску, нужно подключить Arduino Uno через USB кабель к компьютеру. Если всё сделать правильно, то здесь появится новая запись:



Во вкладке **Tools** выбираем тип платы. В нашем случае - Arduino Uno, но если у вас другая плата, то выбираем её.



С настройками подключения платы разобрались, теперь можно загружать программу. Смело жмём на стрелочку. Стрелочка скомпилирует и загрузит программу на микроконтроллер, галочка - скомпилирует, но загружать не будет.



Программирование на C++

Этот раздел пригодится тем, кто не программировал на C-подобных языках или забыл что к чему. Если попадётся что-то знакомое, можно смело пропускать.

Структура программы

Каждая программа для Arduino состоит из двух главных частей-функций:

```
void setup()
{
  // put your setup code here, to run once:
}
```

```
void loop()
{
  // put your main code here, to run repeatedly:
}
```

void setup() - выполнится один раз. В неё обычно засовывают подготовку перед исполнением программы. Инициализация датчиков и переменных, предстартовая проверка - это всё стоит включить в **setup**.

void loop() - будет выполняться снова и снова, пока не сядет батарейка, или плата не будет перезагружена. В ней пишется основная программа.

Не обязательно что-либо писать в теле **void setup()** или **void loop()**. Если нужна программа, которая исполняется один раз, оставляем **void loop()** пустым, или наоборот. Например:

```
void setup()
{
  Serial.begin(9600);
  Serial.println("Hello");
}

void loop()
{
}
```

Переменные

Классно, когда можно ненадолго запомнить что-то, а потом использовать. Для этих целей и существуют переменные. **Переменная** - это “коробочка”, в которую можно что-то положить, чтобы потом достать. Чтобы не запутаться, когда коробочек станет много, мы каждой дадим свое имя - **имя переменной**.

```
int a = 3;
```

Строчка выше означает: “Компьютер, создай, пожалуйста, для меня коробочку с именем **a** и типом **int,** и положи в нее число 3”.

```
int a = 5;
int b = a;
```

Строчки выше означают: “Компьютер, создай, пожалуйста, для меня коробочку с именем **a** и типом **int,** и положи в нее число 5. Потом создай коробочку с именем **b** и типом **int**, и положи в нее копию того, что лежит в коробочке **a**”.

Особенностью С-подобных языков программирования является **статическая типизация**. По-простому, у каждой переменной есть свой **тип значений**, которые можно засунуть в эту переменную. Посмотреть доступные типы переменных можно здесь, в разделе Variables.

Точка с запятой

Команды нужно заканчивать **точкой с запятой**. Особенно доставляет программировать на Arduino после **Python**. Если вдруг забыть, Arduino IDE ласково выведет “expected ';' before...”.

```
int a = 3;
Serial.begin(9600);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
```

Комментарии

В тексте программы можно оставлять **комментарии**. Комментарии никак не используются компьютером и нужны только человеку. С помощью комментариев можно объяснить непонятный код, оставить напоминание дописать проверку входящих данных, или оставить послание потомкам. Привычка писать комментарии к коду - полезная привычка.

```
/*
 Многострочный комментарий
*/

// Однострочный комментарий
// int a = 10; - это тоже комментарий. Переменная не будет создана
```

Методы Arduino

В языке Arduino добавили методы, которые помогают работать с контроллером:

```
// Попросить пин PIN работать в режиме входа
pinMode(PIN, INPUT);

// Считать показание на пине PIN. (1 или 0)
```

```
int digital_result = digitalRead(PIN);
// Установить LOW на пине PIN
digitalWrite(PIN, LOW);

// Считать показание на пине PIN. (0 - 1023)
int analog_result = analogRead(PIN);

// Ничего не делать 1 секунду (1000 мс)
delay(1000);
```

Тут можно найти другие полезные методы.

Serial monitor

В Arduino IDE есть полезная штука, называется **Serial Monitor**. В Serial Monitor можно смотреть данные, которые плата передает на компьютер и отправлять команды на устройство.

Чтобы передать данные в Serial Monitor, сначала нужно открыть соединение с помощью **Serial.begin(speed_in_bauds)**, а потом воспользоваться функциями **Serial.print()** или **Serial.write()** для отправки данных. Чтобы считать данные, пригодится функция **Serial.read().**

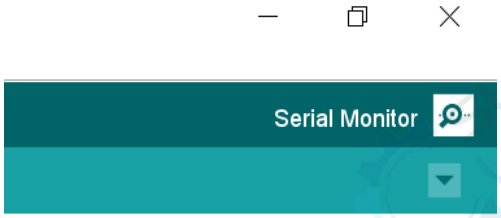
- **Serial.begin(speed_in_bauds)** - откроет Serial соединение со скоростью передачи символов в **speed_in_bauds** символов в секунду. **Бод** (**Baud**) - это количество символов, которые пересылаются за секунду. В нашем случае один символ - это 8 бит, значит скорость передачи в битах в секунду - это **speed_in_bauds,** умноженная на восемь.
- **Serial.print()** - отправит данные в человеко-читаемом **ASCII формате**
- **Serial.write()** - отправит данные в **бинарном формате**

Вот пример записи и чтения Serial Monitor:

```
// Запускаем соединение со скоростью 9600 бод
Serial.begin(9600);
// Отправляем сообщение в Serial Monitor
Serial.println("Yay, new Serial Connection!");

// Проверяем, появились ли новые данные
if (Serial.available() > 0)
{
  // Считываем байт данных
  byte data = Serial.read();
  // Выводим данные в десятичном формате
  Serial.print("New data byte: ");
  Serial.println(data , DEC);
}
```

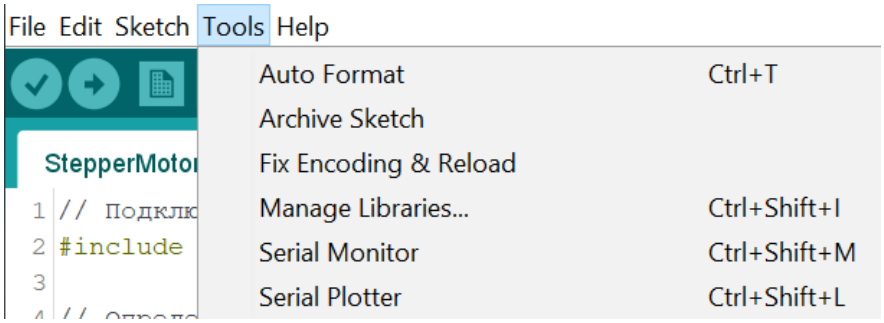
Открыть Serial Monitor можно нажав на кнопку в правом верхнем углу или применив комбинацию **Ctrl+Shift+M**



В правом нижнем углу Serial Monitor можно установить скорость соединения. Если скорости в **Serial.begin()** и в окне Serial Monitor не будут совпадать, при получении данных появятся калябушки. Это из-за того, что плата и компьютер ожидают разное количество символов в один и тот же промежуток времени.

Кстати, с помощью **Serial Plotter**-а можно визуализировать данные, которые приходят в Serial Monitor, а если возможностей Plotter-а недостаточно, то можно перейти в **Processing**, но это уже совсем другая история.

Serial Plotter открывается через меню **Tools** или с помощью **Ctrl+Shift+L.**



Ветвление

Ветвление пригодится когда нужно выполнить разные действия в зависимости от какого-нибудь условия. Вот пример:

```
int a = digitalRead(13);

if (a > 10)
{

}
else if (a < 10 && a >= 5)
{

}
else
{

}
```

Более сложные условия в **if** можно создавать с помощью **логических операторов**, например: **|| - логическое или, && - логическое и, ! - логическое не** и других. Информацию про логические операторы можно найти тут в разделе Structure.

Циклы

Часто нужно несколько раз запустить одни и те же строчки кода. Чтобы не копировать одно и то же много раз, придумали **циклы**.

Например, нам нужно три раза подряд вывести привет в консоль. Можно сделать так:

```
Serial.begin(9600);
Serial.println("Hello");
Serial.println("Hello");
Serial.println("Hello");
```

Жить можно, но что если привет нужно вывести сто раз? В такой ситуации на помощь придут циклы. Есть несколько типов циклов.

Цикл for

For пригодится когда мы точно знаем сколько раз нужно повторять действие.

```
Serial.begin(9600);
for(int i = 0; i < 10; i++)
{
  Serial.println("Hello!");
}
```

Циклу for нужна своя переменная-счётчик, которая будет изменяться в соответствии с правилом. Цикл for будет работать пока условие верно.

```
for(переменная; условие выхода; правило изменения)
{
  // Полезный код
}
```

переменная - здесь мы создаём переменную-счётчик. Например: **int i = 0;**
условие выхода - когда условие не выполнится, цикл прекратится. Например: **i < 3;**
правило изменения - правило, по которому переменная-счётчик будет изменяться в конце каждой итерации цикла. Например: **i *= 2;**

Цикл while

Что делать если мы не знаем сколько раз нам понадобится цикл? Использовать цикл while.

```
while(условие)
{
  // Полезный код
}
```

условие - когда условие не выполнится, цикл прекратится.

Кстати, цикл for - это цикл while, только немного приукрашенный. Вот во что превращается for в процессе компиляции:

```
int i = 0;
```

```
while(i < 10)
{
   i++;
}
```

Цикл do while

Это немного другая версия цикла while. Do while сначала сделает действие, а потом проверит условие. Полезно, когда действие нужно сделать перед проверкой условия цикла.

```
do
{
  // Полезный код
} while (условие);
```

условие - когда условие не выполнится, цикл прекратится.

Функции

Компьютеры любят за то, что они берутся за любую работу. Даже за самую скучную. Правда программисту всё равно нужно описать как и что делать компьютеру. **Функция** - это инструкция по выполнению мини-задания, которое нужно объяснить всего один раз, а потом можно просто давать входные данные для обработки и ждать результата.

```
bool usefulFunction(int arg1, float arg2)
{
bool result = false;

// Полезный код

// Возвращаем результат
return result;
}

// Вызываем одну и ту же функцию на разных входных данных
bool res1 = usefulFunction(1, 1.0);
bool res2 = usefulFunction(10, -3.0);
bool res3 = usefulFunction(0, 55.0);
```

Код выше создаст функцию с именем **usefulFunction**, которая принимает на вход два **аргумента arg1** и **arg2**, типа **int** и **float** соответственно, делает полезную работу и

возвращает результат в виде значения переменной типа **bool**. Потом эту функцию можно использовать много раз.

Кстати, функции могут ничего не принимать и ничего не возвращать. Например:

```
void function()
{
  // Полезный код
}

// Два раза вызываем функцию
function();
function();
```

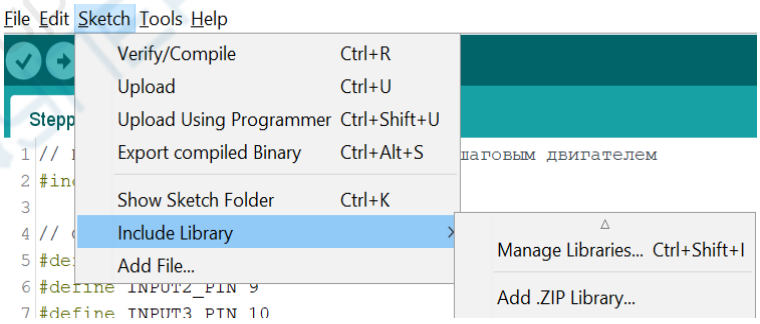
Библиотеки для Arduino

Кода уже написано много, зачем снова изобретать колесо? Давайте использовать библиотеки.

Библиотека - код, готовый для повторного использования. Добавить библиотеку в программу можно с помощью **#include**, например:

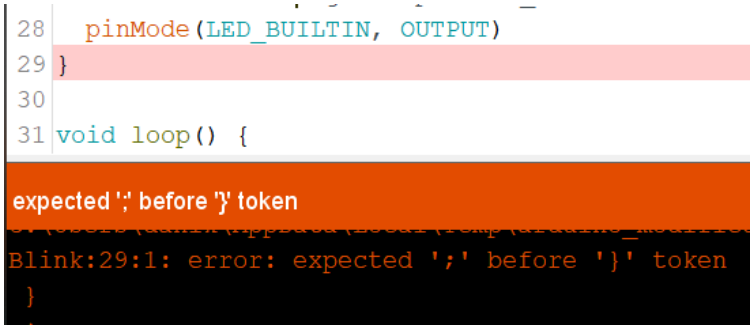
```
#include <Servo.h>
// Теперь можно использовать класс Servo из библиотеки Servo.h
Servo myservo;
```

Для установки библиотек можно использовать встроенный **менеджер библиотек** через **Manage Libraries**, добавить архив библиотеки через **Add .ZIP Library**, или просто скопировать папку с библиотекой в **путь/до/Arduino/libraries:**



Чего-то строчка покраснела

Не нужно бояться **ошибок компиляции**. Arduino IDE пытается помочь, а не навредить. Обычно, если внимательно прочитать сообщение, всё становится понятно.



Снова запятую забыл…

Магические символы --, --, +=

Программиста хлебом не корми, дай чего-нибудь оптимизировать. Вот и арифметика с переменными попала под оптимизацию.

```
int a = 10;
// Все три строчки ниже увеличат a на 1
a = a + 1;
a += 1;
a++;

// Все три строчки ниже вычтут 1 из a
a = a - 1;
a -= 1;
a--;

// Обе строчки ниже умножат a на 10
a = a * 10;
a *= 10;
```

#define

Часто нужно добавить какое-нибудь постоянное значение в нескольких местах, но потом замучаешься менять, если вдруг понадобится изменить. Для этого можно использовать **макрос** #define:

```
#define ИМЯ ЗНАЧЕНИЕ;
```

В процессе компиляции, **препроцессор** заменит все объявленные имена на значения.

```
// До препроцессора
```

```
#define VALUE 10;

int a = VALUE;
int b = VALUE;

// После препроцессора
int a = 10;
int b = 10;
```

Чтобы различать переменные и #define, имена в #define пишут капсом. Делать так не обязательно, но желательно, чтобы ваш код сразу правильно поняли.