

Содержание

Урок №1: Знакомство с Arduino. Введение.	2
Урок №2: Введение в электронику и программирование	16
Урок №3: Математические операторы, переменные, знакомство с условными операторами	31
Урок №4: Работа с портами ввода и вывода, константы, булевы переменные	52
Урок №5: Работа с аналоговыми портами, считывание данных с датчика света	63
Урок №6: Работа с аналоговыми портами, подключение тонального динамика, работа с датчиком температуры	69
Урок №7: Знакомство с массивами данных, циклы, перебор массивов, работа с потенциометром	74
Урок №8: Подключение библиотек, знакомство с ультразвуковым датчиком	81
Урок №9: Управление DC двигателем через Arduino	88
Урок №10: Создание своих функций	95
Урок №11: Знакомство с серводвигателем	103
Урок №12: Проекты с серводвигателем	107
Урок №13: Проекты на закрепление	108
Урок №14: Знакомство с bluetooth модулем	111
Урок №15: Знакомство с цифровым тахометром	115

Урок №1

Тема: Знакомство с Arduino. Введение.

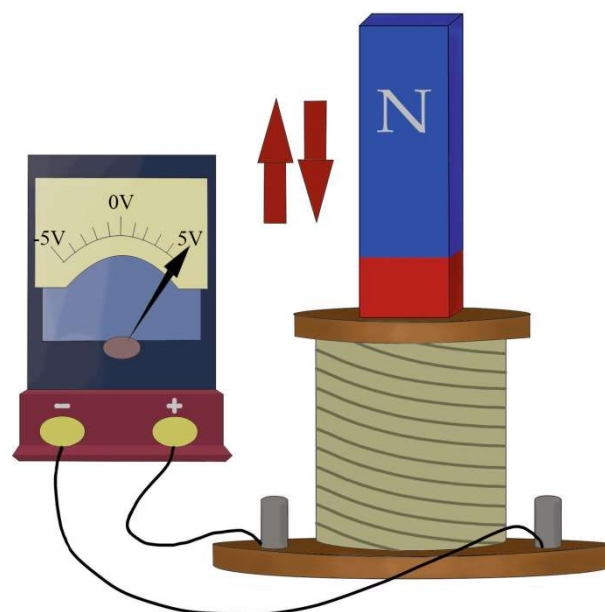
Понятие электричества: Электрическим током называют поток заряженных частиц (электронов), которые движутся по проводнику от источника тока к потребителю тока. Источником тока могут быть как аккумуляторы, солнечные батарейки, так и электростанции. Проводником может быть любое вещество или предмет проводящим ток: металлический провод, вода и даже человек может являться проводником.

Частицы, которые движутся по проводнику называются электроны.

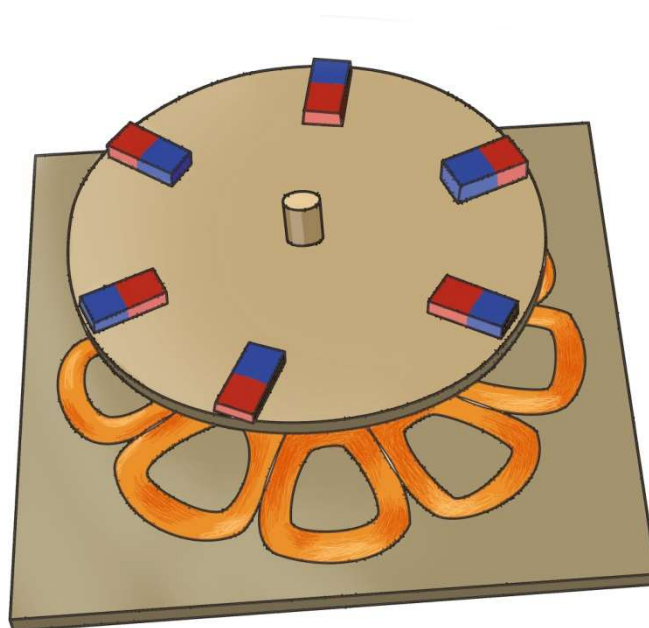
Потребитель тока - это любой прибор который работает на электричестве, например простая лампочка.

Электроны настолько малы, что их нельзя увидеть или потрогать. Из электронов состоят все атомы, электроны присутствуют в каждом предмете. Но только электроны которые движутся вместе в одном порядке могут создать электрический ток который включит электроприбор или зажжет лампочку. В проводниках электроны есть всегда, но так как они не движутся или движутся хаотично, то просто подключив лампочку к проводу - нельзя заставить ее гореть, необходимо заставить двигаться электроны в одном направлении, для этого и существуют электростанции. Но как они работают?

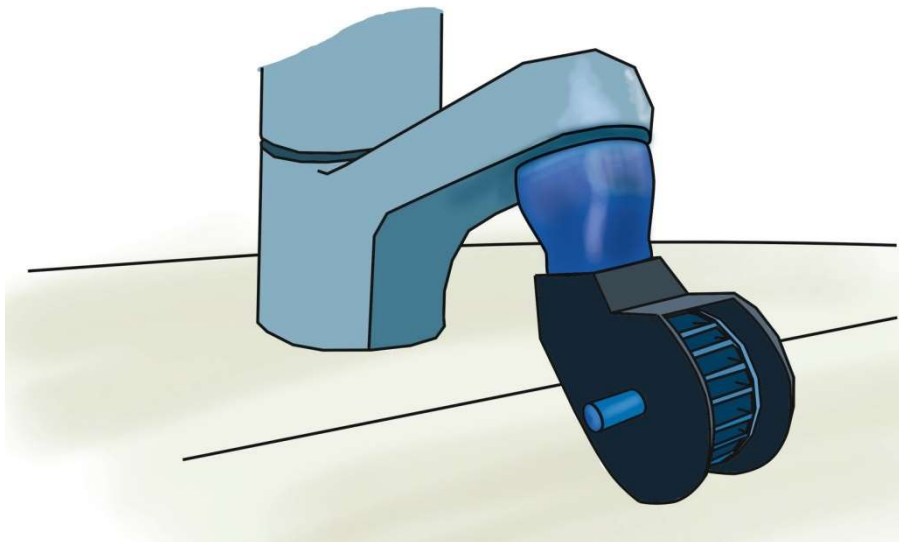
На самом деле заставить двигаться электроны очень просто. Они очень хорошо реагируют на магнитное поле, то есть достаточно быстро провести магнитом рядом с проводом, чтобы электроны начали двигаться и в проводе появился ток. Это происходит из за того, при приближении магнита к проводу - электроны смещаются со своего места, от того, что магнит на них действует. Все этого из за того, что у электронов есть свои поля, и у магнита есть поле, их не видно, но они действуют друг на друга, например если магнит поднести к другому магниту, то они оттолкнутся. Магнит создает поле которое двигает электроны или другими словами - напряжение, чем сильнее магнит действует, тем больше напряжение. В этот короткий промежуток времени, когда мы подносим магнит к проводнику - можно зажечь лампочку. Но после того как электроны сдвинулись с места, они успокаиваются, чтобы они опять начали двигаться, необходимо убрать магнит, тогда они начнут двигаться обратно. Если быстро подносить и убирать магнит, ток в проводнике будет двигаться постоянно и лампочка будет гореть почти без остановки.



На таком явлении и основываются все электростанции. На каждой электростанции установлен генератор, генератор этот очень простой, внутри него находится много катушек из проволоки и много магнитов, магниты быстро и без остановки двигаются рядом с катушками и в проводе на катушках появляется ток, этот ток по проводам передается в наши дома. Магниты приближаются и удаляются от катушек и делают они это очень быстро - 50 - 60 раз в секунду, поэтому нам кажется, что лампочки в комнате горят без остановки, на самом деле они потухают и загораются по 50-60 раз в секунду, но наши глаза не способны заметить такой скорости, поэтому мы видим плавное горение. Можно сделать чтобы магниты двигались медленней, но тогда наши глаза будут замечать мерцание света и будут уставать (все равно, что смотреть на мигающий фонарь). Но что же двигает магниты так быстро? Тут тоже всё просто. Магниты расположены по кругу и вращаются рядом с катушками. Как например на картинке снизу.



Заставить вращаться магниты тоже очень просто. Например гидроэлектростанции работают благодаря течению рек, к такому диску с магнитами просто крепится турбина, и от воды диск с магнитами вращается и вырабатывает в катушках электричество. На фото ниже изображена турбина напечатанная на 3D принтере, она подключается к обычному крану дома, если к ней подключить диск с магнитом и поднести рядом катушку с проводом, то когда мы включим воду у нас начнет вырабатываться ток.



Скачать 3D модель водной турбины для печати на 3D принтере можно по этой ссылке:
http://3dtoday.ru/3d-models/games/toys/mini_vodyanaya_turbina/

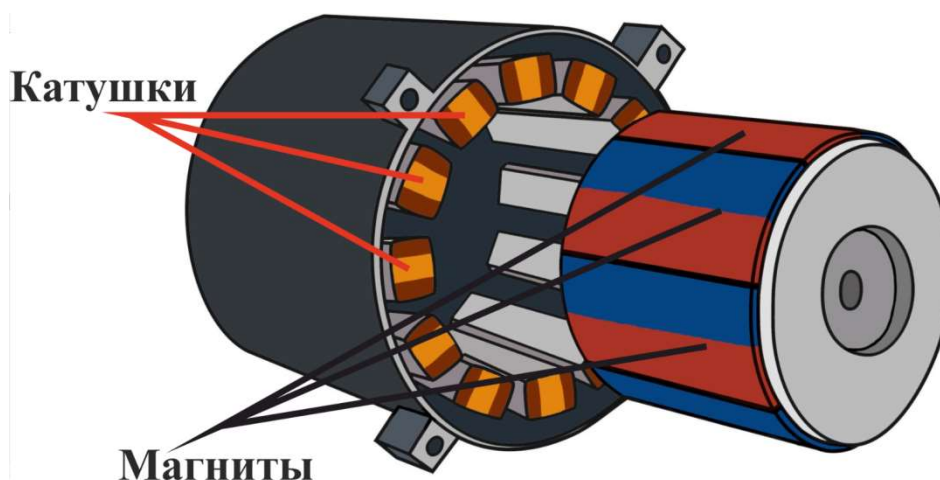
По такому же принципу работают ветряные электрогенераторы, только в этом случае к генератору крепится ветряная турбина.



Такой генератор можно сделать самому, тут либо можно намотать проволоку на катушку и сделать диск с магнитами. Либо можно использовать простой моторчик.



Дело в том что все двигатели работающие от электричества состоят из точно таких же катушек с проволокой и магнитов. На рисунке ниже, изображено внутреннее строение двигателя. По краям расположены катушки, а по центру магниты и если центр начать вращать, то можно вырабатывать электрический ток.



Получается чтобы сделать свой ветрогенератор, достаточно просто к такому двигателю прикрепить лопасти от вентилятора и если начать их вращать то подключенная к двигателю лампочка будет гореть.

Таким образом работают ветряные электростанции. В Казахстане наиболее распространены теплоэлектростанции (сокращенно ТЭЦ) они работают от тепла вырабатываемого путем горения угля. Они очень похожи на ветряные электростанции, только вместо ветра используется пар. При горении угля в больших емкостях кипятится вода и вырабатывается пар, этого пара настолько много, что он точно так же как и ветер вращает турбины и вырабатывает электричество.

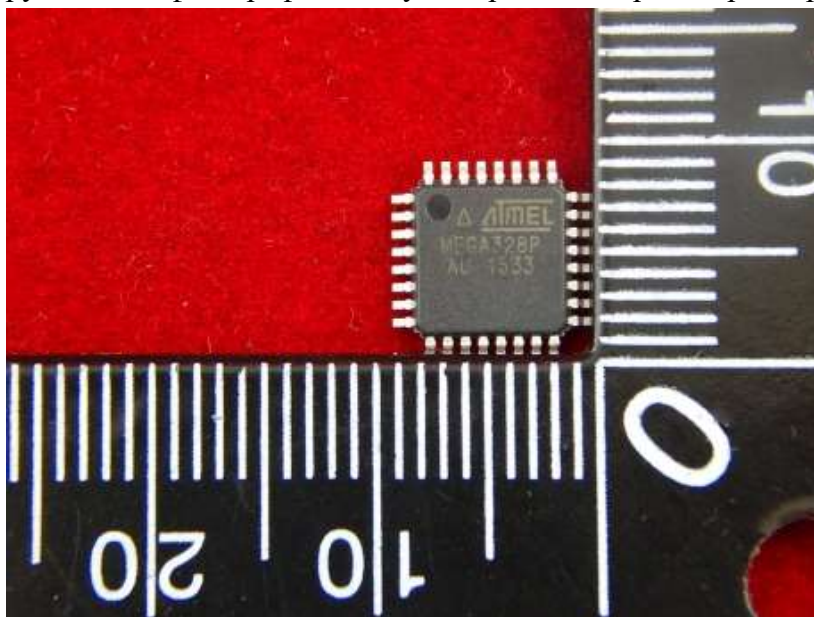
Но все эти электростанции вырабатывают переменный ток, что такое переменный ток и почему он так называется? Как мы уже говорили, что когда мы подносим магнит к катушке электроны быстро сдвигаются в одну сторону и создается электрический ток, когда мы магнит убираем от катушки, то электроны двигаются в другую сторону, и тоже создается электрический ток, и если постоянно подносить и убирать магнит, то ток будет течь сначала в одну сторону, потом в другую, всё время меняя свое направление, такой ток называется переменным. Многие электроприборы могут работать от такого тока, например чайник, лампочка или вентилятор.

Но сложные устройства, такие как электроника не любят когда направление тока в них меняется, они могут просто сгореть от такого тока, им нужно чтобы ток всегда шел в одном направлении. Например если мобильный телефон воткнуть напрямую в розетку, то он моментально сгорит, поэтому для зарядки телефона необходимо использовать блок питания (его еще называют зарядником, зарядкой и т.д.), блок питания преобразует переменный ток из розетки в постоянный ток для телефона, этот ток всегда движется в одном направлении. Все батарейки вырабатывают постоянный ток, в них электроны двигаются от положительного заряда к отрицательному (от плюса (+) к минусу (-)). Солнечные батареи тоже вырабатывают постоянный ток.

Но батарейки работают не так как электростанции, в них даже не используются магниты. Внутри батареек есть 2 химических вещества, грубо говоря одно из этих веществ реагирует с другим (разъедает, расщепляет, оно как-бы распадается) и при этом вырабатывается электрический ток, поэтому батарейки не долговечны, когда вещества внутри них полностью среагируют друг с другом, то батарейку можно выбросить, так как пользы от нее больше не будет. Аккумуляторы имеют свойство заряжаться, при их зарядке частички вещества (ионы) восстанавливаются обратно, и можно использовать аккумулятор еще раз, но дело в том что восстанавливаются они не полностью и каждый раз все меньше и меньше, поэтому в среднем после 500 - 3000 перезарядок, аккумулятор тоже становится непригодным. Солнечные батареи работают похоже, они состоят из специального материала который отдает электроны при попадании света и при этом вырабатывает электричество. Но в отличие от аккумуляторов - солнечные батареи служат намного дольше.

В наших уроках в основном будет использоваться именно постоянный ток. Как уже говорилось ранее электроны двигаются из-за напряжения. Напряжение измеряется в вольтах (V). Думаем вы не раз видели на батарейках надпись 1.5V это означает что электроны будут течь от этой батарейки с напряжением 1.5V. Так же ток может течь с определенной силой, как вода в кране, чем больше сила тока, тем толще нужен провод, как и воде чем больше напор, тем толще нужна труба. Если провод будет слишком тонким а сила тока большой, то провод начнет нагреваться и сгорит. Так же как и вода с большим напором протекая по тонкой трубе, просто разорвет ее от давления. Сила электрического тока измеряется в Амперах (A). Как мы знаем выключатели помогают управлять током, как краны в воде. Регулируя скорость вентилятора или температуру обогревателя мы на самом деле регулируем напряжение которое на них подается, подобно тому, что закрыть кран на половину например.

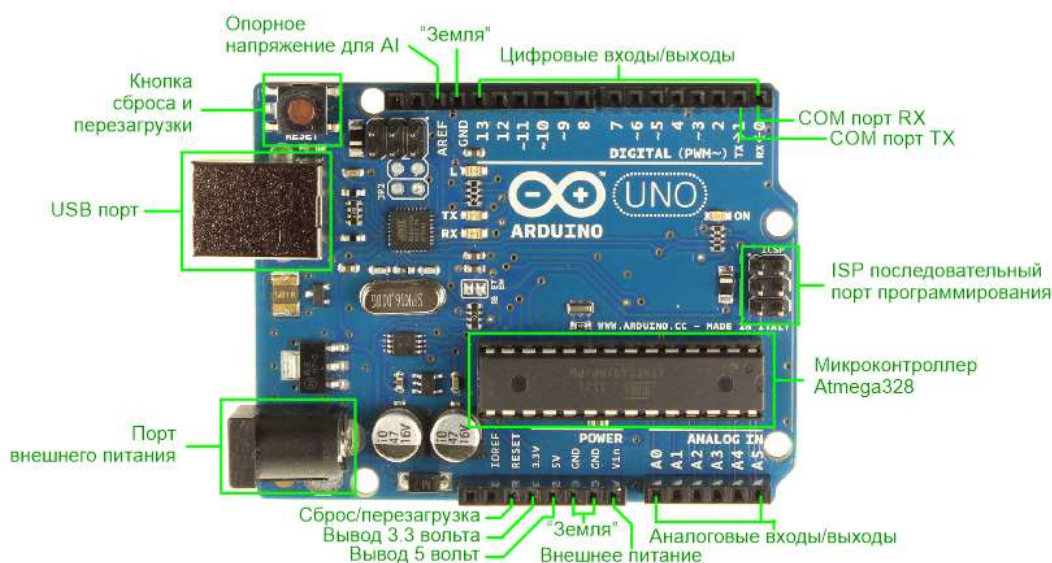
Вся робототехника управляется с помощью микроконтроллеров, микроконтроллеры это такие микросхемы которые могут управлять электричеством с помощью компьютерной программы которая в него загружена. На фотографии внизу изображен микроконтроллер



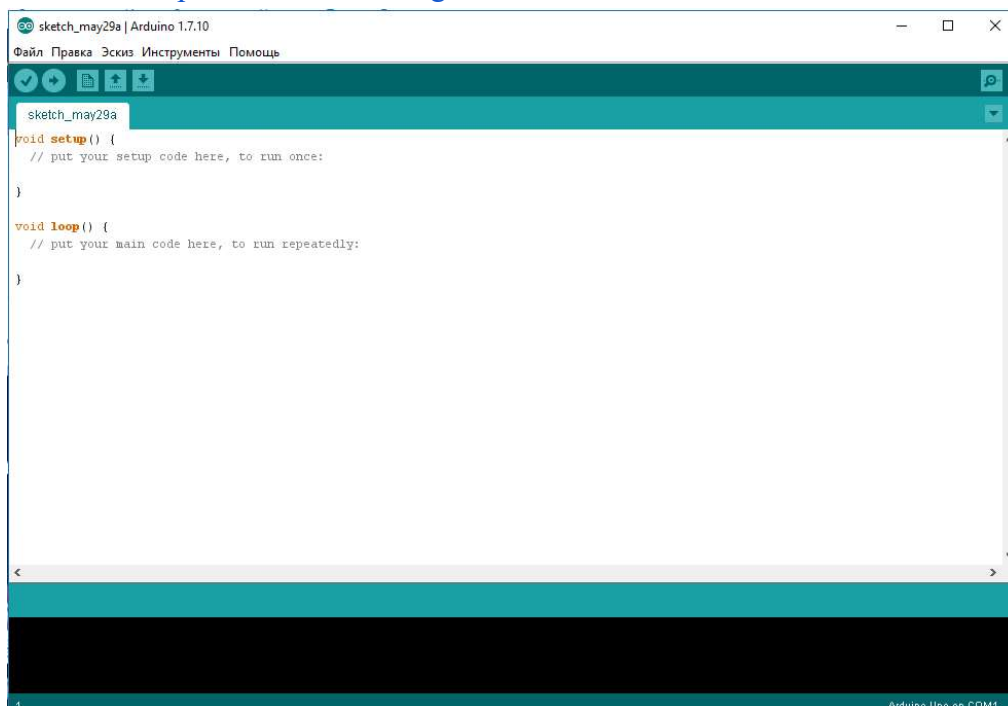
С помощью такого небольшого устройства можно сделать много интересных вещей, начиная от автоматического управления светом, пожарной сигнализации до сложных беспилотных летательных аппаратов, подслушивающих жучков, и многого другого.

Но как же к такой маленькой микросхеме подключить так много устройств и как же его запрограммировать, если его просто так не подключить к компьютеру?

Для этого была разработана отладочная плата Arduino, это тот же самый микроконтроллер, но уже с привычным для нас разъемом USB, питания, а так же выводами для подключения различных датчиков и устройств. Arduino бывают как оригинального производства из Италии так и альтернативные аналоги других фирм, но они все используют один и тот же чип, поэтому разницы в них никакой - нет, кроме цены конечно же. На рисунке ниже изображена именно такая отладочная плата.

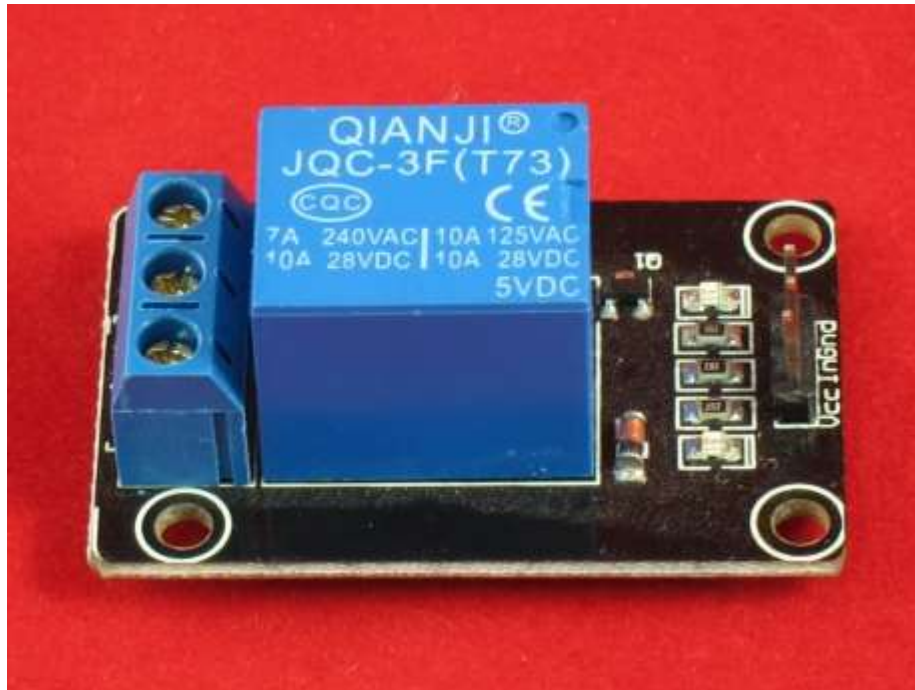


Она подключается с помощью USB разъема к компьютеру и программируется с помощью специальной программы которая называется Arduino IDE. Скачать и установить ее можно с официального сайта <http://www.arduino.org/downloads>



К Arduino можно подключать различные датчики и устройства которые управляются сигналом 5 вольт. Все датчики обычно работают по очень простому принципу. Например датчик света, если свет появляется то датчик выдает 5 вольт, а если света нету то датчик ничего не выдает, таким образом Arduino может понять темно сейчас или светло. Точно так же с датчиком движения или огня. для того чтобы считывать информацию с датчиков как и нужны Аналоговые порты и Цифровые порты изображенные на фотографии выше. На каждый из этих портов можно подключить датчик, Arduino смотрит сколько вольт подается на порт и определяет сработал ли датчик или нет.

Так же Arduino умеет управлять устройствами, например включать и выключать их. Как это происходит? Для этого Arduino подает на Цифровой порт 5 вольт, если к этому порту подключен светодиод, то он загорится. Мы можем управлять подачей 5 вольт как угодно, можем сделать чтобы 5 вольт подавалось через определенное время, или при условии что датчик света сработал, или еще по какому нибудь поводу, Делается это с помощью программ. Но как уже управлять мощными приборами а не только светодиодами, если им нужно например 220 вольт а не 5 вольт. Для этого существуют специальные выключатели называются реле, они могут включать выключать любые устройства но у них нет кнопок, как на привычных выключателях, вместо кнопки на этот выключатель подается 5 вольт и он уже включает устройство мощнее, например чайник, обогреватель, вентилятор и тд. Модуль реле изображен на фотографии ниже



Слева расположены контакты для подключения мощных электрических приборов, а справа расположены ножки для подключения Arduino.

Но как же сделать так, чтобы Arduino подала 5 вольт на какой-нибудь порт?

Для этого необходимо запрограммировать плату Arduino под Ваши нужды.

Программа представляет собой набор команд которые понятны компьютеру. На первый взгляд они кажутся сложными и непонятными на самом деле если эти команды перевести с английского на русский язык, то они очень легки для запоминания.

Например например команда `delay()` переводится с английского как “задержка” и служит для того, чтобы приостановить программу на какое-то время. например `delay(1000)` останавливает программу на 1000 миллисекунд, то-есть на 1 секунду.

Давайте подключим светодиод к Arduino к цифровому выходу №2 а загрузим в Arduino тестовую программу. Для этого откроем Arduino IDE

```

void setup() {
  pinMode(2, OUTPUT);
}

void loop() {
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}

```

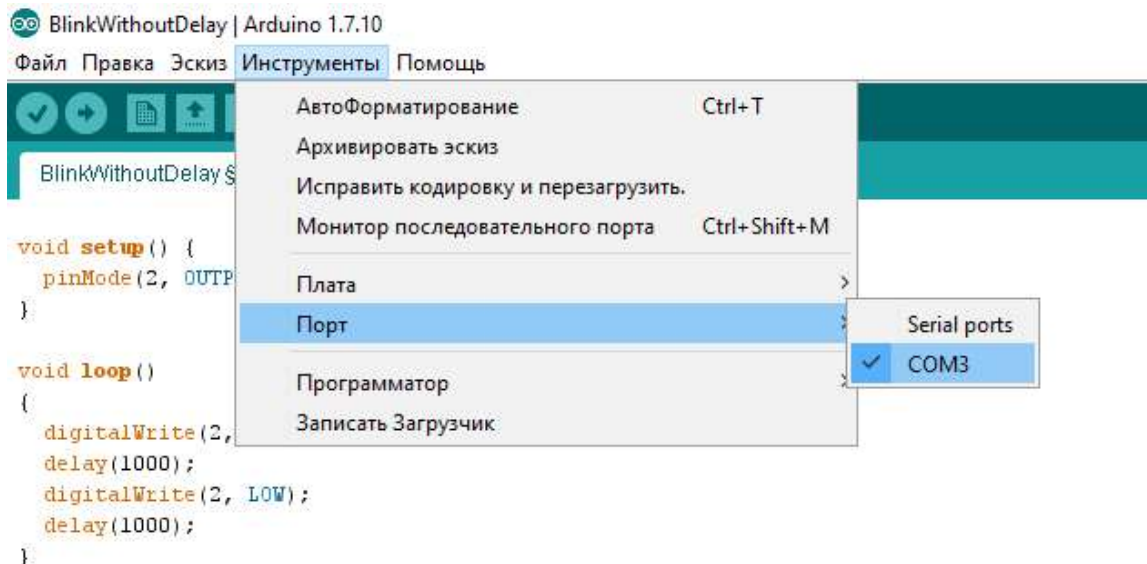
Global variables use 9 bytes (0x) of dynamic memory, leaving 2 039 bytes for local variables. Maximum is 2 048 bytes.

Arduino Uno on COM3

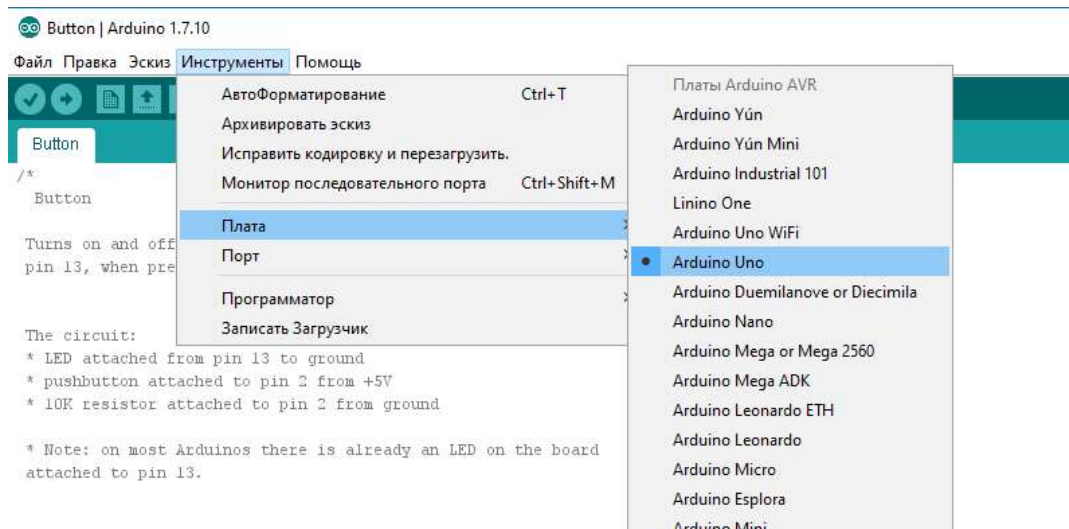
Теперь давайте сотрем, всё что здесь написано, и напомним код, указанный ниже

```
void setup() {  
  pinMode(2, OUTPUT);  
}  
  
void loop()  
{  
  digitalWrite(2, HIGH);  
  delay(1000);  
  digitalWrite(2, LOW);  
  delay(1000);  
}
```

Теперь давайте подключим Arduino к компьютеру, после этого необходимо зайти в меню **Инструменты** -> **Порт**, если Arduino подключена и драйвера установлены, то у нас появится COM порт, он может называться COM1, COM2, COM3 и т.д. (у каждого по-разному), мы выбираем его, как на изображении ниже

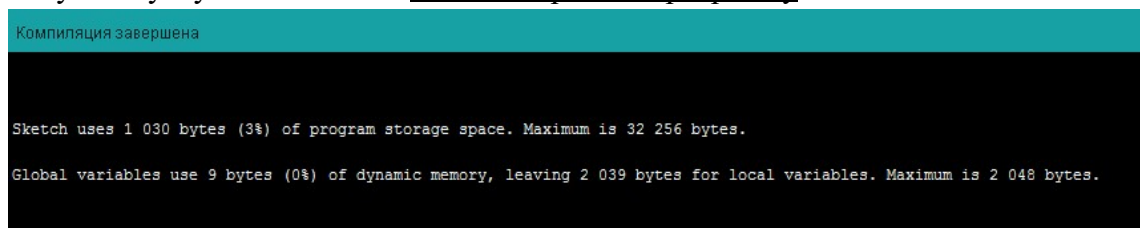


Теперь необходимо выбрать тип Arduino, в наших уроках будут использоваться платы Arduino UNO, поэтому в меню **Инструменты** -> **Плата**, выбираем **Arduino Uno** как на изображении ниже

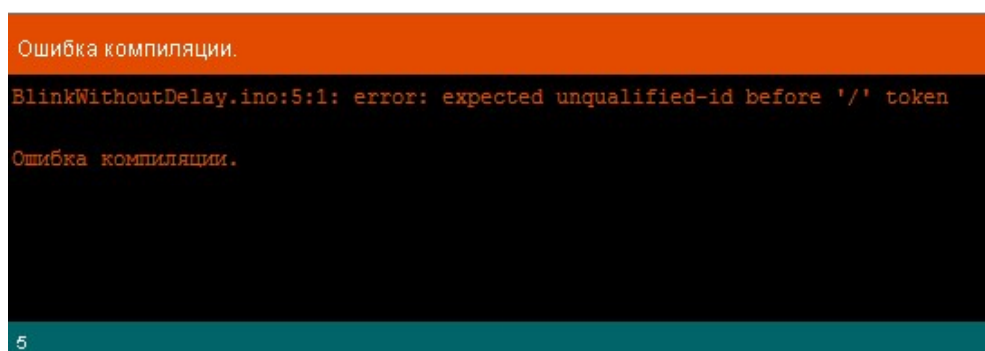


Теперь мы можем проверить наш код и загрузить его в Arduino, Для проверки кода

необходимо нажать на кнопку с изображением галочки . Теперь Arduino IDE проверит наш код на ошибки, если ошибок не будет то мы в черном поле снизу увидим такое сообщение: “Компиляция завершена”. Процесс проверки программного кода на ошибки в программировании называется “Компиляция”, проверить код на ошибки и подготовить программу к запуску - называется “Скомпилировать программу”.



Если вдруг компиляция пройдет неудачно и в коде будут ошибки, то внизу выйдет сообщение “Ошибка компиляции” и ниже оранжевым текстом будет описана ошибка и на какой строке она сделана.



В данном случае ошибка на строке №5. Вы можете преднамеренно сделать ошибку и проверить как работает компилятор (кстати компилятором называется программа которая проверяет код на ошибки, в данном случае Arduino IDE это компилятор).

Если ошибок нет, то можно загружать код в Arduino Uno. Для этого необходимо нажать на кнопку загрузки, она зеленого цвета и на ней изображена стрелка вправо .

После успешной загрузки кода внизу появится сообщение “Вгрузили”

```
Вгрузили.  
  
Sketch uses 1 030 bytes (3%) of program storage space. Maximum is 32 256 bytes.  
Global variables use 9 bytes (0%) of dynamic memory, leaving 2 039 bytes for local variables. Maximum is 2 048 bytes.
```

Поздравляем Ваш первый код загружен на плату Arduino Uno и уже работает. Если сейчас подключить светодиод в цифровому входу №2 - он начнет мигать раз секунду. Почему именно раз в секунду? Потому что мы так написали. Давайте теперь подробно разберем код который мы написали, идем по порядку:

В коде Arduino всегда есть блоки `setup()` и `loop()` эти блоки называются функции пишутся они так:

```
void setup() { ЗДЕСЬ ЛЮБОЙ }  
void loop() { ЗДЕСЬ ЛЮБОЙ КОД }
```

Сначала пишется слово `void`, что оно значит разберем с вами позже, дальше пишется слово `setup` и после него обязательно две скобки “()”, открывающая и закрывающая, внутри них ничего нет, так же пишется и функция `loop`, перед словом `loop` пишется слово `void` с пробелом, после слова `loop` ставится две пустые скобки “()”. Дальше как вы заметили идут фигурные скобки открывающие и закрывающие “{}”. Эти скобки самые важные! Мы внутри них и будем всегда писать наш код. Так зачем же эти непонятные функции **loop** и **setup**?

Из названий можно было заметить, что **loop** переводится как “петля”, а “**setup**” переводится “настройки”. Эти функции соответствуют своим названиям. Код который мы напишем внутри **setup**, то есть внутри фигурных скобок которые идут после **setup()** { ВНУТРИ ФИГУРНЫХ СКОБОК } будет выполняться 1 раз, когда мы включим Arduino. Например если мы напишем код, который включает и выключает светодиод, и поместим его в **setup()** { }, то когда мы включим Arduino светодиод загорится и погаснет.

А код который внутри **loop** то есть внутри фигурных скобок которые будут после **loop()** { ВНУТРИ ФИГУРНЫХ СКОБОК } будет все время повторяться пока Arduino включена. Например если тот же самый код который включает и выключает Arduino мы поместим в **loop()** { }, то пока Arduino включено - светодиод будет постоянно включаться и выключаться.

Рассматриваем наш код дальше. Внутри функции **setup()** { } мы написали такую программу:

```
digitalWrite(2, HIGH);  
delay(1000);  
digitalWrite(2, LOW);
```

```
delay(1000);
```

Давайте мы разберем ее по порядку. Здесь есть такая команда **digitalWrite()**, эта команда подает или отключает питание с цифровых портов Arduino, например команда **digitalWrite(2, HIGH)** подает 5 вольт на порт №2, а команда **digitalWrite(2, LOW)** отключает питание с порта №2. Если обратить внимание на закономерности, то можно заметить что команды отличаются только словом **HIGH** и **LOW**, что с английского переводится **ВЫСОКИЙ** и **НИЗКИЙ**. Значит команда **digitalWrite(2, LOW)** подает низкое напряжение на порт 2, а команда **digitalWrite(2, HIGH)** высокое напряжение (5 вольт) на порт №2. Если мы например напишем **digitalWrite(3, HIGH)** то подадим 5 вольт на цифровой порт №3 на Arduino таким образом можно подать 5 вольт на любой цифровой порт от 0 до 13. Чтобы легче запомнить название функции **digitalWrite** нужно ее просто перевести, **digital** - цифровой, **Write** - записать то есть записать на цифровой порт. Но почему записать? По сути мы подаем, записываем, выводим на цифровой 5 вольт или записываем 0 вольт. Если бы например хотели узнать подается на порт №2 5 вольт или нет, то мы бы написали **digitalRead(2)**, но это мы будем изучать позже.

Просто запомните, для того чтобы подать 5 вольт на цифровой порт, необходимо использовать функцию **digitalWrite()**.

Кстати вы заметили что команда **digitalWrite()** тоже называется функцией, но пишется без фигурных скобок “{}” и здесь нету слова **void**? Почему же так? Как различать со скобками нужно писать функцию или нет, как определить функция это или не функция? На самом деле здесь все просто все команды после которых идут круглые скобки называются функциями, например **delay()**, **serialWrite()**, **blablabla()** - всё это функции. Но разница здесь в том, что в функциях **loop()** и **setup()** мы сами придумываем свой код, который будет выполняться внутри них, поэтому нам нужно писать фигурные скобки и в прописывать внутри них команды. А функция **digitalWrite()** уже встроена в Arduino, ее писали разработчики Arduino и нам нельзя писать в них свой код, их можно только использовать. Со всеми встроенными функциями Arduino, которые нельзя изменять, вы можете познакомиться по ссылке <http://arduino.ru/Reference>

Идем дальше, как вы заметили в коде у нас после функции **digitalWrite()** - пишется функция **delay()**:

```
digitalWrite(2, HIGH);  
delay(1000);  
digitalWrite(2, LOW);  
delay(1000);
```

В переводе на русский, слово **delay** означает **задержка**. То есть функция **delay()** делает паузу в программе. Длина паузы задается в миллисекундах, напомним, что в одной секунде 1000 миллисекунд. То есть для того, чтобы сделать паузу в программе на одну секунду нужно написать **delay(1000)**, чтобы сделать паузу 2 секунды - **delay(2000)** и т.д.

Теперь зная эти команды, мы можем объяснить код который выполняется в функции **loop(){}**

```
void loop()
{
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```

Сначала мы пишем **digitalWrite(2, HIGH);** этим самым мы подаем 5 вольт на порт 2, в который подключен светодиод, светодиод при этом загорается. Далее мы делаем паузу в 1 секунду **delay(1000);** и после этого подаем низкое напряжение на 2 порт **digitalWrite(2, LOW);**, далее мы делаем паузу в 1 секунду **delay(1000);** и код повторяется заново, потому что мы его написали в функции **loop()**.

Попробуйте теперь сделать следующее:

1. Мигать светодиодом каждые пол секунды,
2. Каждые 2 секунды
3. Сделайте так, чтобы светодиод горел пол секунды, а потухал на 2 секунды
4. Сделайте, чтобы светодиод 2 раза мигнул по 500 миллисекунд, а 3 раз мигнул на 2 секунды и повторял - это действие

Ну и на конец осталась последняя строчка кода, которую мы не разобрали - **pinMode(2, OUTPUT);**, как вы заметили, она расположена в функции **setup(){ }** поэтому, как говорилось выше, она выполнится только 1 раз, в тот момент когда мы включим Arduino. Эта функция служит для того чтобы настроить цифровой порт №2 в режим записи - то есть чтобы мы могли подать на порт №2 - ВЫСОКОЕ напряжение или НИЗКОЕ, если этого не сделать, то подавать напряжение не получится. То есть по функция **pinMode()** устанавливает режим работы порта, **pin** - переводится “порт”, а **Mode** - “режим”.

Чтобы можно было выводить 5 вольт на порт №3 нужно написать **pinMode(3, OUTPUT);**, чтобы выводить на порт №4 - **pinMode(4, OUTPUT);** и т.д. То есть если посмотреть на закономерность, можно заметить, что меняется только номер порта.

Эту функцию достаточно выполнить 1 раз, чтобы порт переключился в режим записи, после этого на порт можно подавать высокое напряжение выполняя функцию **digitalWrite(2, HIGH);**.

Теперь мы знаем, что означает каждая строчка кода написанного нами, еще раз пробежитесь по каждой строчке и объясните сами себе их назначение

```
void setup() {
  pinMode(2, OUTPUT);
}

void loop()
{
  digitalWrite(2, HIGH);
  delay(1000);
}
```

```
digitalWrite(2, LOW);  
delay(1000);  
}
```

Теперь поэкспериментируйте с кодом:

1. Подключите светодиод к 3 пину и перепишите код так, чтобы светодиод мигал.
2. Попробуйте стереть строку `pinMode()`; и посмотрите что произойдет
3. Попробуйте перенести функцию `pinMode()`; из функции `setup()` в функцию `loop()`. Что изменилось? В чем разница, от того где мы написали функцию `pinMode()`;
4. Сделайте так, чтобы светодиод мигал 50 раз в секунду, вы должны почти не замечать как он мигает.

Урок №2

Тема: Введение в электронику и программирование

Электрон: На прошлом уроке мы узнали, что электрический ток это упорядоченное движение электронов в проводнике под воздействием поля (напряжения). Но что из себя представляет электрон? Мы все состоим из молекул и атомов, это маленькие-маленькие частички, которые соединены между собой, электроны еще меньше атомов, они входят в состав атома, и вращаются вокруг его ядра, вот так обычно изображают атом: по центру ядро атома и электроны, которые вокруг него вращаются

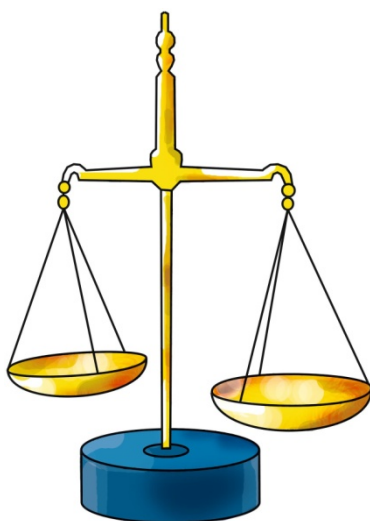


Линии вокруг ядра это пути (траектории), по этим путям двигаются электроны. Но так обычно для нас их просто рисуют на картинках, нам самом же деле этих линий нет, электроны двигаются не по кругу, они двигаются очень быстро и в разбросанном хаотичном порядке, куда они будут двигаться дальше пока еще невозможно рассчитать и предугадать. Электронов не обязательно может быть четыре как на картинке, у разных атомов их разное количество, например у атома кислорода их 8, а в атоме железа их 26, в атоме золота их аж 79! Но помимо электронов атом еще состоит из протонов, протоны находятся в ядре атома.

Также на рисунках неверно изображено расположение электронов, на самом деле, они находятся в 100 000 раз дальше от ядра, но если бы мы изобразили это на картинке, то ядро было бы так далеко, что мы бы его не увидели на таком маленьком изображении и изображение бы пришлось настолько увеличить, что оно бы не поместилось на этот листок, нужен бы был листок размером в примерно в 30 километров, чтобы увидеть как далеко они находятся, поэтому атом изображают так. Никто не знает как на самом деле выглядят электроны и протоны, потому что на данный момент развитие технологий не позволяет их увидеть.

И так, давайте теперь разберем как электроны двигаются по проводникам, ведь атомы при этом остаются на месте, вы ведь не видели никогда чтобы металл внутри кабеля протекал

вместе с электричеством? Нет, атомы всегда остаются на месте, а когда на электроны действует внешнее поле, то они просто вырываются из притяжения атома и двигаются дальше, по сути атом их теряет, но атом не теряет их навсегда, рядом с атомом есть и другие атомы они тоже теряют электроны и когда свободные электроны вырванные из них движутся и попадают под влияние нашего атома, то он притягивает к себе недостающие электроны. Давайте теперь по порядку разберем как это происходит: Когда мы включаем провод в розетку, и подключаем например к лампочке, то электроны из розетки начинают выталкивать электроны из атомов и встают на их места, вытолкнутые электроны тоже начинают двигаться дальше и выталкивают электроны из соседних атомов, вставая на их места и так далее. Когда электроны доходят до лампочки, то она начинает светиться. Но почему провод не светится а лампочка светится? Дело в том, что когда электроны начинают двигаться, то вырабатывают тепло, если электроны двигаются по обычному проводу, то тепло будет не сильным, потому что его слишком мало, чтобы нагреть толстый провод, но в лампочке установлена нить из специального металла, причем очень тонкого, который так сильно нагревается, что начинает светиться. И чем больше напряжение, тем сильнее двигаются электроны и тем ярче горит лампочка, если напряжение будет слишком большим то лампочка сгорит, если слишком маленьким, то будет гореть тускло. Напомним, что напряжение измеряется в вольтах. И так в спокойном состоянии электроны никуда не вырываются и ток по проводнику не течет, электроны не могут вырваться из под действия притяжения ядра, потому что в ядре есть протоны, они притягивают электроны как магнит, происходит это потому, что электроны имеют отрицательный заряд (-), а протоны - положительный заряд - (+). А как вы наверное пробовали экспериментировать, то противоположные стороны магнита притягиваются, так же и протоны притягивают электроны. В спокойном состоянии атом не имеет электрический заряд, но если например его нагреть или приложить трение, то можно тем самым вырвать электрон из атома и он станет положительно заряженным, потому что заряд протонов (+) станет больше заряда электронов (-), мы ведь потеряли электрон, и равновесие чуть-чуть изменилось в положительную сторону. Это равносильно что на левую и правую чашу весов положить положить много разных гирь так, чтобы весы встали ровно (спокойный атом), а потом

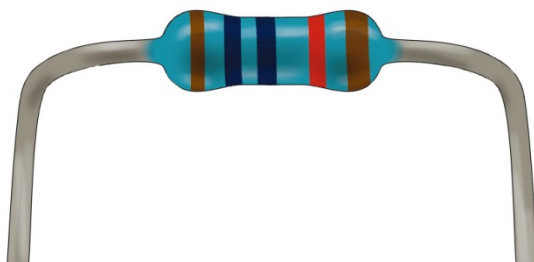


убрать одну гирю и весы начнут перевешиваться в одну сторону (положительно заряженный атом).

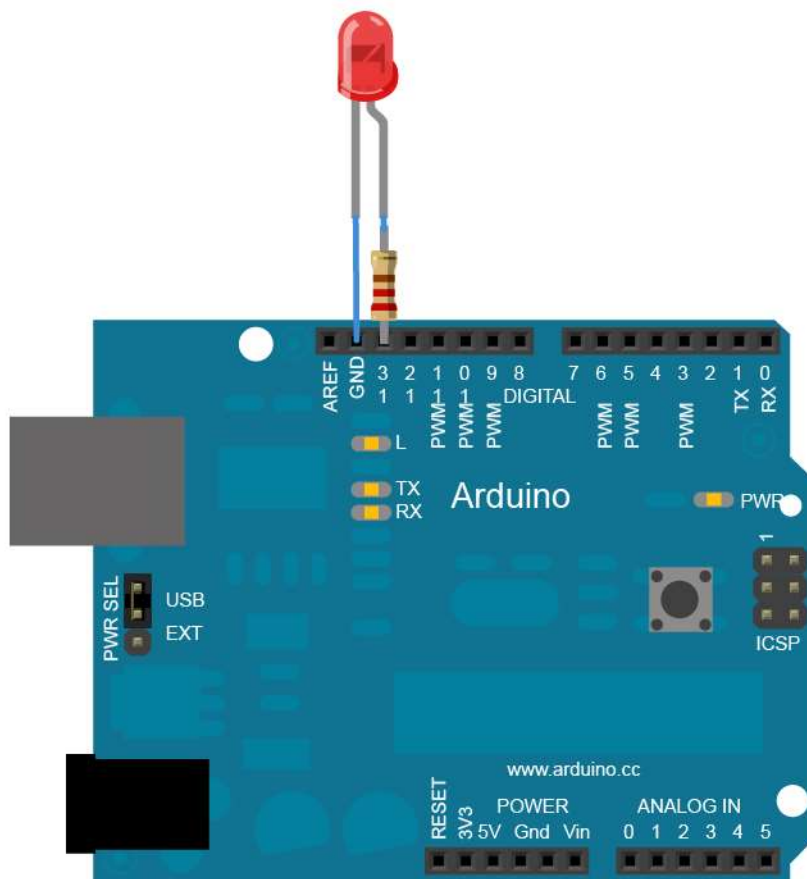
Еще мы можем добавить электронов в атом, тогда мы его опять выводим из равновесия, но уже в другую сторону, электронов в нем станет больше и атом будет отрицательно заряжен такой атом, при малейшем воздействии будет всеми силами избавиться от лишнего электрона, чтобы прийти в равновесие, точно так же и положительно заряженный атом будет всеми силами притянуть электрон, которого ему не хватает. С этим закончили, теперь давайте поговорим о том почему одни вещества проводят ток легко и быстро, например железо, медь, алюминий, золото и тд. А например дерево, пластик, резина их не проводят.

Дело здесь вот в чем: У разных веществ как мы уже сказали, разное количество электронов в атомах, а также разное количество протонов, если в атоме будет много протонов и мало электронов, то протоны будут так сильно притягивать электроны, что если мы возьмем и начнем действовать на этом и попытаемся вырвать из него электрон, то у нас этого не получится и электрический ток через такие атомы не потечет, потому что ни один электрон не сдвинется с места. У металлов, из которых сделаны проводники в атомах количество электронов которые могут вырваться - много, поэтому ток через них течет хорошо. У неметаллов же все наоборот, электронов которые бы могли вырваться при нормальных условиях у них вообще нет, поэтому воздействие на них напряжение никак не создает в них ток. По этому провод в резиновой оболочке не ударит вас током, электроны просто не вырвутся из атомов.

Иногда нам требуется понизить напряжение, например Arduino на цифровых выходах дает 5 вольт, но нам нужно подключить к нему светодиод, который горит от 3 вольт, если мы подключим его к 5 вольтам, то он сгорит. Нам нужно понизить напряжение до 3 вольт. Это можно сделать с помощью резистора. Резистор выглядит примерно так:



Если светодиод подключить не напрямую к Arduino, а через резистор, как показано ниже, то напряжение понизится и светодиод не сгорит



Напряжение после резистора изменится на 3 вольта и светодиод будет гореть долго и счастливо. Но почему так происходит, почему напряжение после резистора упало до 3 вольт? Всё дело в том, что внутри резисторов состоят из металлов, которые нагреваются при прохождении через них электрического тока, при нагревании они расходуют часть тока и превращают его в тепло, поэтому на выходе у резистора ток будет меньше чем на входе. Другие типы резисторов состоят из специального вещества, которое частично блокирует электроны, а частично пропускает, такие вещества называются полупроводники, потому что они частично проводят электричество. Поэтому блокируя часть электронов, сопротивляясь им резистор ограничивает ток и напряжение снижается. Резистор по другому еще называют сопротивлением. Но резисторы бывают разные, одни сопротивляются сильнее, другие слабее, существует сотни разных резисторов, но как узнать какой резистор нужен именно нам? Тут всё просто, например вы же знаете что батарейки бывают 1.2 вольта, 1.5 вольта, 3.7 вольта, 9 вольта, вы наверное знаете какие куда нужно подключать и никогда не путаетесь. Точно так же и с резисторами, у всех есть свое сопротивление и измеряется оно в Омах, например чтобы снизить напряжение с 5 вольт до 2.8 вольт для нашего светодиода, нужен резистор с сопротивлением 220 Ом.

Но почему резистор именно с таким сопротивлением, как узнать какой нам нужен резистор для другого светодиода? Для этого существует формула расчета сопротивления, основана она на законе Ома, сам закон пока мы не будем описывать, давайте разберем формулу. Формула очень простая: Сопротивление = Напряжение/Силу тока (или $R = U / I$).

Теперь давайте разберем как пользоваться этой формулой и рассчитывать нужное нам сопротивление для снижения тока.

Например предположим что у нас есть такой светодиод http://RadioMart.org/_904 зеленого цвета.



В описании к нему написано

Цвет свечения: зеленый
Напряжение в прямом направлении (В): DC1.9 ~ 2,5
Прямой ток (мА): 4—14
Интенсивность освещения : 100-200mcd
Длина волны (нм) : 520 - 530

Из этих характеристики нам нужно Напряжение и Ток (тоже самое, что и сила тока). Почему именно эти параметры? Потому что в формуле нужны именно они “*Сопротивление = Напряжение/Силу тока (или $R = U / I$)*”.

воНапряжение светодиода у нас пример 1.9льта -2.5 вольт, возьмем чтонибудь среднее - 2.2 вольт. Сила тока, который потребляет этот светодиод равно 4 - 14 мА (миллиампер), возьмем среднее значение 9мА. Кстати говоря в одном ампере (А) 1000 миллиампер (мА).

И так мы знаем что светодиод потребляет ток 9мА или по другому 0.009 А и напряжение необходимое для его работы - 2,2 В.

Так же мы знаем что с выхода на Arduino подается 5 вольт, поэтому нам нужно снизить напряжение до 2.2 вольт. То есть нам нужно скинуть лишние 2.8 вольт (5вольт - 2.2вольта - 2.8вольта) мы подставляем в нашу формулу значения:

$Сопротивление = Напряжение/Силу тока$	$Сопротивление = 2.8вольта/0.009А$
--	------------------------------------

Если разделить 2.8 на 0.009, то получим примерно 311 - это и есть сопротивление резистора, нам нужен резистор номиналом ровно 311 Ом, чтобы наш светодиод не сгорел.

Теперь вы можете рассчитать нужный резистор для светодиода. Но чтобы не тратить каждый раз время на расчеты, было создано множество калькуляторов, которым достаточно дать параметры светодиода и от скольких вольт они должны питаться и калькулятор всё сделает за вас, один из таких калькуляторов можно найти по ссылке

<http://cxem.net/calculc/ledcalc.php>

Напряжение источника питания (В): 5

Прямое напряжение светодиода (В): 2.2 [выбрать]

Прямой ток светодиода (мА): 9 [выбрать]

Количество светодиодов: 1

☒ Схема подключения
 ☒ Принципиальная схема

☐ Другое расположение светодиодов
☐ Сохранить вычисления на картинку
☐ Резистор 4 полосы

Расчет

Очистить

Здесь нам нужно заполнить необходимые параметры:

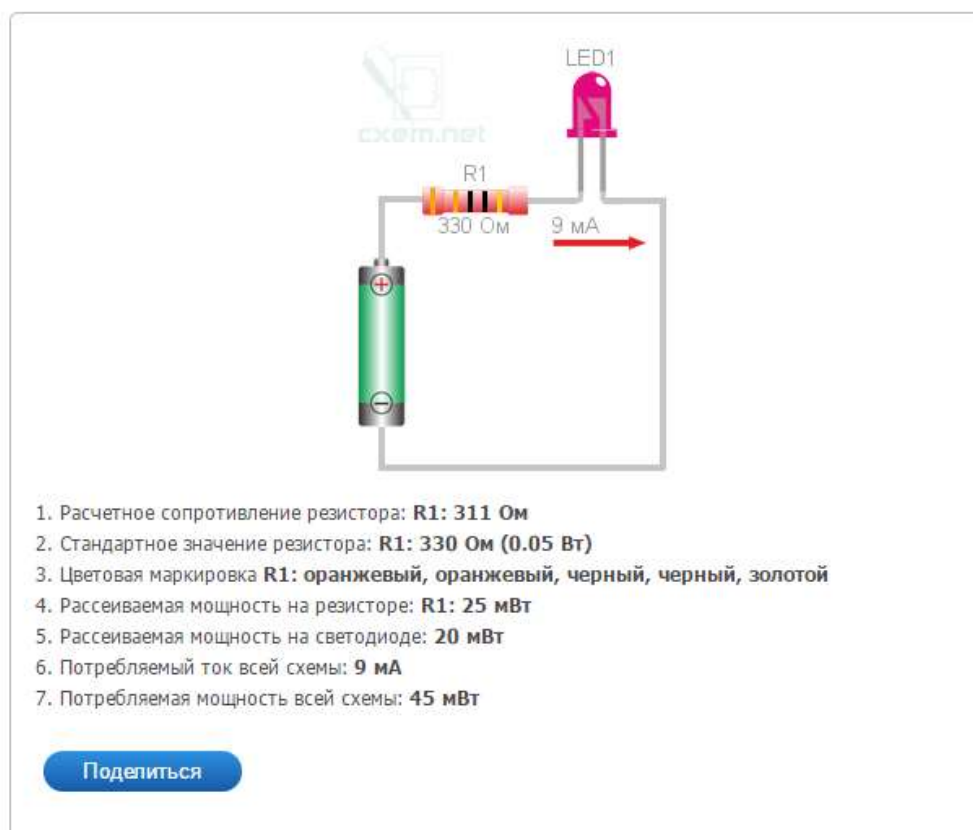
Напряжение источника питания - у нас источник питания Arduino и она выдает нам 5 вольт, значит пишем сюда цифру 5

Прямое напряжение светодиода - из характеристик на сайте RadioMart мы узнали что напряжение светодиода 2.2 вольта

Прямой ток светодиода - из тех же характеристик мы знаем что светодиода потребляем 9 миллиампер, пишем сюда цифру 9

Количество светодиодов - у нас он 1 поэтому пишем сюда единицу

Теперь можем нажимать кнопку **расчет** после чего у нас на странице будет выведен следующий результат:

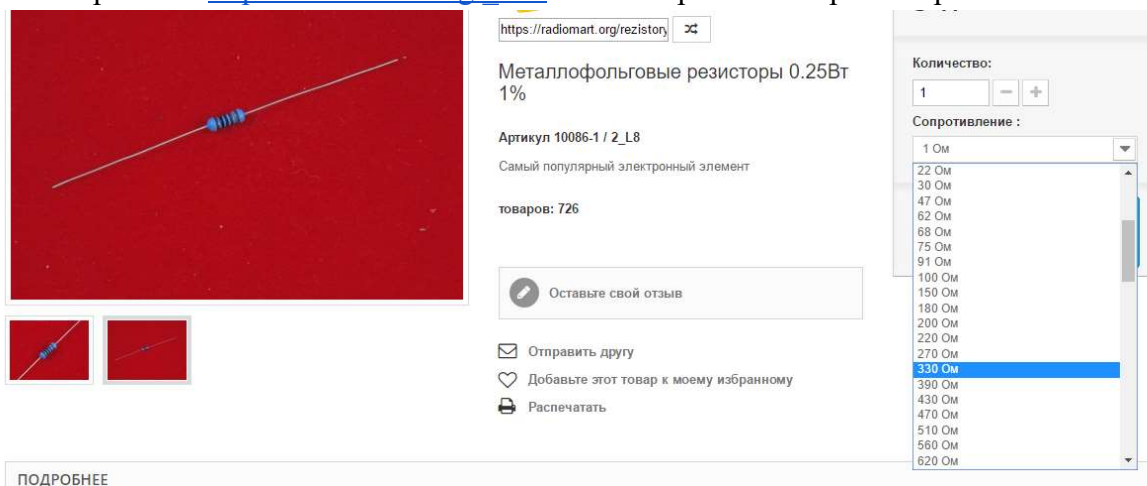


Из результата мы видим что расчетное сопротивление резистора составляет **311 Ом** точно, как мы рассчитали.

Но так же здесь написано что стандартное сопротивление резистора равняется **330 Ом**, что же это значит?

Дело в том, что на заводах выпускают не все номиналы резисторов, а только самые популярные и резистора 311Ом, 312Ом, 313Ом многих других просто не существует. Поэтому если резистора сопротивление, которого вам нужно - не существует, то можно взять сопротивление побольше, но которое существует, от этого плохо светодиоду не станет, он будет немножко тусклее светится, но так как разница между 330 Ом и 311 Ом очень мала, то эту тусклость будет почти не заметно.

Но опять же, как нам определить существует такой светодиод или нет? Да никак, вы просто идете в ближайший магазин радиодеталей и спрашиваете резистор который ближе всего подходит к тому который нужен вам. Либо чтоб никуда не ходить, можно зайти в раздел с резисторами на <http://RadioMart.org/> **905** и посмотреть какие резисторы есть:



Можно так же попробовать подключить резистор меньше по номиналу, например 270 Ом, тогда светодиод будет гореть ярче, и возможно даже не сгорит, потому что при расчетах мы брали среднее значение напряжения и среднее значение тока, значит у нас есть небольшой запас. Если вы хотите, чтобы светодиод горел максимально ярко, тогда нужно при расчетах брать его максимальные значения, например вернемся к характеристикам зеленого светодиода

Цвет свечения: **зеленый**
Напряжение в прямом направлении (В): **DC1.9 ~ 2.5**
Прямой ток (мА): **4—14**
Интенсивность освещения : **100-200mcd**
Длина волны (нм) : **520 - 530**

Отсюда следует, что для расчетов, напряжение нужно брать 2.5 вольт, а не 2.2, а ток брать 14 мА, а не 9. Теперь если мы посчитаем наше нужное сопротивление, то у нас выйдет 180 Ом. Значит даже с таким резистором наш светодиод не сгорит. А уж с резистором 270 Ом тем более. Запомните лучше брать резистор номиналом чуть больше, чтобы при мелких скачках напряжения - светодиод не сгорел.

Теперь вы знаете как подобрать сопротивление резистора, и почему для светодиода мы использовали 220 Ом.

Знакомство с мультиметром.



Сегодня мы будем знакомиться с устройством под названием: мультиметр. Возможно он вам знаком. Разные люди его называют по разному: цэшка, прибор, тестер и другое. Не важно как его называть, главное чтобы вы понимали что речь идет именно о нем. Давайте разберемся для чего он. Из названия МУЛЬТИ (разный) МЕТР (измерять) можно догадаться, что мультиметр создан для измерения разных величин. С его помощью можно измерять напряжение постоянного и переменного тока, силу постоянного тока, узнать сопротивление резистора, определить плюс и минус в цепи постоянного тока, найти короткое замыкание и многое другое. Сегодня мы будем знакомиться с мультиметром **DT-830** выглядит он примерно так:



Мультиметр представляет собой устройство с дисплеем и поворотной ручкой для смены режимов работы, так же к нему подключаются 2 щупа, с помощью которых производятся измерения. Эти щупы разных цветов, обычно черного и красного.



Разницы между ними - нет, разные цвета были сделаны для удобства. Обычно красный подключают к плюсу, а черный к минусу. Итак, в мультиметре есть 3 гнезда для подключения щупов, но щупов всего два. И подключаются они, только в 2 нижних, причем красный подключается в центральное гнездо, а черный в нижнее. Для чего же тогда верхнее гнездо. При работе с робототехникой и микроконтроллерами оно будет использоваться редко. Оно необходимо для того чтобы измерять сильный ток, до 10 Ампер, такие случаи будут редко, поэтому остановимся на нем позже. И так

по порядку, как же нам измерить напряжение постоянного тока?

Для этого существуют режимы работы мультиметра которые входят в группу под названием DCV (direct current voltage - напряжение постоянного электрического тока) на мультиметре изображается либо надписью DCV либо буквой V пунктирной и прямой линией. Здесь у нас 5 режимов (1000, 200, 20, 2000m, 200m) но все они похожи, например если нам нужно измерить напряжение на батарейке 1.5 вольт, нам подойдут для этого режимы 2000m, 20, 200, 1000, если мы переключимся на один из этих режимов, и коснемся щупами плюса и минуса батарейки, то мы увидим сколько вольт на ней. Но в этих режимах все таки есть разница. Одни режимы нужны для высокого напряжения, другие для низкого, режим **1000** необходим для измерения напряжений от 200 вольт до 1000 вольт. Режим **200** необходим для измерения напряжения от 20 вольт до 200 вольт, режим **20** измеряет напряжение от 2 вольт (2000 милливольт) до 20 вольт, режим **2000m** измеряет напряжение от 0.2 вольт (200 милливольт) до 2 вольт (2000 милливольт) и наконец режим **200m** измеряет напряжение от 0 до 0.2 вольт (200 милливольт).

У этих режимов есть свои диапазоны измерений, но почему же мы можем измерить 1.5 вольт в разных режимах, дело в том, что вся разница между режимами заключается в точности их измерений например если мы измерим 1.5 вольт в режиме **2000m**, то мультиметр нам покажет 1500 - если батарейка полностью заряжена, но почему же 1500, ведь у нас 1.5 вольт, а не 1500, дело в том что этот режим отображает всё в миллиамперах об этом соответствует надпись (2000m). Если мы эти же 1.5 вольт измерим в режиме **20**, то мультиметр нам покажет 1.50, потому что точность здесь измеряется в сотых долях, больше символов просто не влезет на мультиметр, в режиме **200** у нас отобразится 1.5 вольт (тут влезает только одна цифра после точки), а в режиме **1000** отобразится 2 вольт, почему 2, а не 1.5, дело в том что в этом режиме цифры после запятой совсем нет, и мультиметр округляет значение 1.5 до 2.

В этом и есть разница - чем выше стоит режим тем лучше он измеряет высокое напряжение и тем хуже низкое, поэтому для полтора вольт самый оптимальный режим будет **2000m**, для 5 вольт самый оптимальный режим будет **20** потому что режим **2000m** не может замерить напряжение больше 2 вольт, так же как и режим **200m** не сможет замерить 1.5 вольт, потому

что максимальное напряжение которое он может измерить - это 0.2 вольт, что намного меньше чем 1.5.



Ну собственно больше здесь никаких особенностей нет, подключаете 2 щупа к плюсу минусу и измеряете напряжение, но напоминаем, что эти режимы только для постоянного тока (постоянный ток у нас в аккумуляторах, в батарейках, в компьютере, в блоке питания от телефона).

В этих режимах так же можно определить полярность, то есть узнать где плюс, а где минус. Проще всего это объяснить на примере: если подключить красный щуп к плюсу, а черный щуп к минусу ну например к той же батарейке, то мультиметр отобразит 1.5 вольт, а если сделать наоборот и подключить красный щуп к минусу, а черный к плюсу то мультиметр отобразит -1.5 вольт.

Выходит что из этого можно сделать правило - если на мультиметре отображается знак минуса, то красный щуп подключен к минусу, если же на мультиметре не отображается знак минуса, то красный щуп подключен к плюсу, вот и всё.

Точно так же измеряется напряжение переменного тока, но для этого существует другая группа режимов ACV (alternating current voltage - напряжение переменного тока), на мультиметрах она обычно обозначена либо тремя буквами ACV либо буквой V с волнистой линией. Здесь у нас есть два режима 200 и 750, думаю как вы уже догадались, 200 измеряет напряжение до 200 вольт, а 750 - до 750 вольт, соответственно так как у нас в розетках более 200 вольт (по крайней мере в хорошо населенных пунктах), то режим нам подойдет только 750. Переставляем режим на 750 и касаемся щупами за источник переменного тока, например вставляем щупы в розетку и тут уже не важно каким щупом куда, у нас в любом случае покажется 220 вольт (может меняться и быть от 200 до 235). С этим тоже всё.



Что еще можно измерить мультиметром? Например у нас дома где-то валяется резистор и мы не знаем его сопротивление, с помощью мультиметра мы можем измерить его

сопротивление. Для этого существует группа режимов со значком Ома - Ω . Ну здесь у

нас 5 режимов:

200 - измеряет сопротивление до 200 Ом

2000 - измеряет сопротивление до 2000 Ом (2кОм - 2 килоома)

20k (20 000) - измеряет сопротивление до 20000 Ом (20кОм)

200k (200 000) - измеряет сопротивление (200кОм)



2000k (2 000 000) - измеряет сопротивление до 2 000 000 Ом (2МОм - 2 мегаома)

Для того, что бы определить какое сопротивление у резистора, нам нужно приложить концы щупов к левой и правой ножке резистора, после чего мультиметр отобразит сопротивление резистора, если вы выбрали правильный режим, если же вы выбрали неправильный режим, то мультиметр ничего не отобразит либо отобразит цифру 1 в самом левом краю, это означает что нужно увеличить режим и увеличивать до тех пор пока не покажется наше сопротивление. Очень важно не касаться металлических концов щупов, потому что наше тело тоже имеет сопротивление, при измерении сопротивления резистора - значения могут быть не верными.

Еще одна важная вещь которую можно делать с помощью мультиметра, это звонить, что

же это значит? Если переключить режим мультиметра, на значок звуковой волны, и коснуться щупами друг друга, то мультиметр запищит.



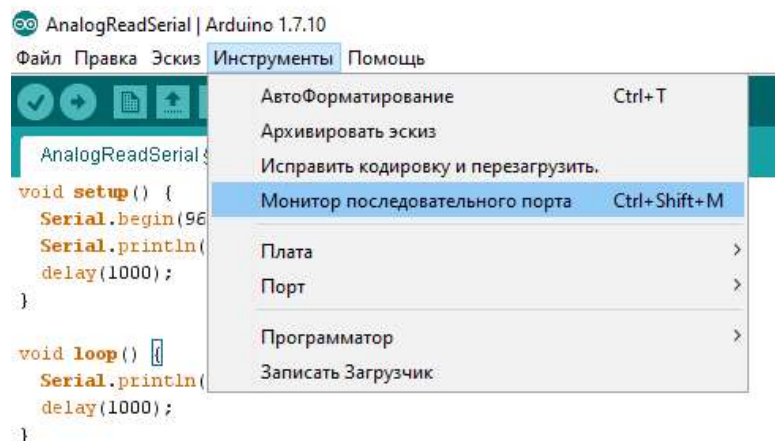
Это означает что мы замкнули электрическую цепь, в данном случае в электрической цепи у нас только щупы.

Этим способом можно проверить замыкание или проверить куда идет провод и какой-то контакт, допустим у нас 10 запутанных проводов, мы нашли один конец и нам нужно найти другой конец, для этого к одному щупу мы примотаем один конец провода, а вторым щупом мы будем касаться все концов проводов по очереди, пока не мультиметр не издаст звук.

Serial.read и Serial.print

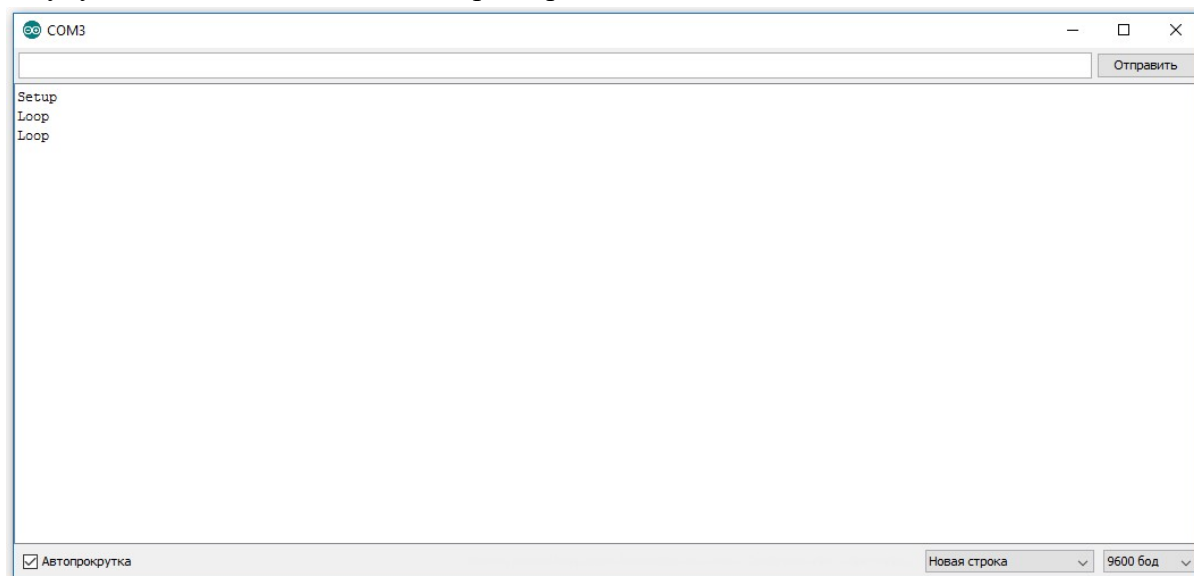
Достаточно на сегодня теории, давайте перейдем к программированию. На прошлом уроке мы научились мигать светодиодом, давайте рассмотрим как можно давать команды на Arduino напрямую с компьютера и получать на них ответ. Давайте напишем простой код и разберем его:

```
void setup() {  
  Serial.begin(9600);  
  Serial.println("Setup");  
  delay(1000);  
}  
  
void loop() {  
  Serial.println("Loop");  
  delay(1000);  
}
```



Теперь давайте загрузим данный код в нашу плату Arduino, как это сделать - обсуждалось на прошлом уроке. После успешной загрузки программы зайдите в меню **Инструменты -> Монитор последовательного порта** либо нажмите горячую клавишу **Ctrl+Shift+M**. После этого перед вами откроется окно в которое будут выводиться слова Setup и бесконечно повторяющееся слово Loop.

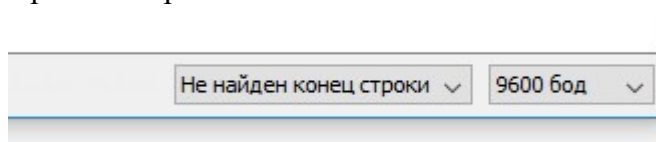
Это те самые слова, которые мы написали в коде, если мы напишем другие слова то и они тоже будут выводиться. И так давайте разберем, что мы написали.



Думаю как вы поняли то команда **Serial.println("Setup");** как раз выводит надпись "Setup" а команда **Serial.println("Loop");** выводит "Loop". Если рассматривать глубже, то эти команды не просто выводят надписи на экран компьютера, они могут отправить на компьютер любую информацию через порт USB к которому Arduino подключено, а компьютер эту информацию может уже обработать и сделать какое либо действие. Но сейчас мы пока ограничимся на том, что будем просто программировать надписи и читать, что выводит Arduino на компьютер. И так надпись Setup вывелась 1 раз, потому что мы ее вывели в блоке setup() { }, а надпись Loop выводится все время потому, что находится в блоке loop() { }. Функция delay() делает просто паузу, ее мы проходили на прошлом уроке. Еще одна новая функция которую мы не изучали

это **Serial.begin(9600);** Она вызывается в блоке `setup() { }` и как вы наверное могли догадаться - может служить для настройки. Так и есть - эта функция открывает порт передачи данных, без нее команда **Serial.print()** работать не будет. Параметр 9600 означает скорость порта с которой будут отправляться данные, эта скорость измеряется в бодах, значит мы открыли серийный порт передачи данных на скорости 9600 бод.

Что такое бод? По сути **бод** это количество символов отправляемых в секунду. То есть со скоростью 9600 бод мы можем отправить за секунду 9600 символов (букв, знаков препинания и т.д.). Для наших целей этого вполне достаточно, но бывают скорости и выше, например 19200, 38400, 57600, 115200 и другие. Если вы заметили, то в мониторе последовательного порта, так же есть настройка скорости:



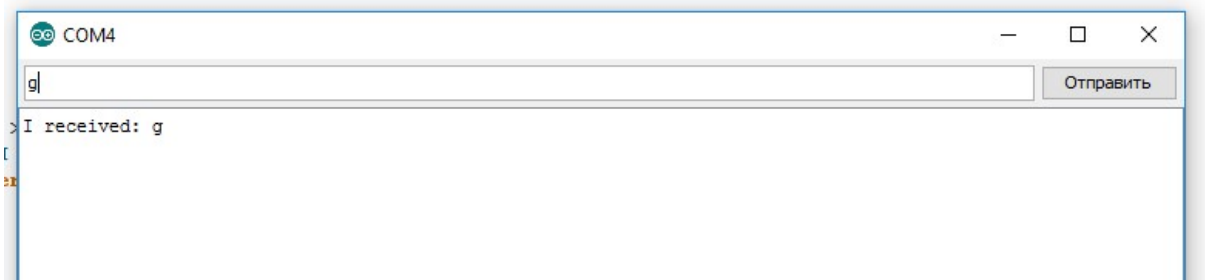
Для чего она нужна если мы в программе уже установили нужную скорость? Дело в том, что в программе мы устанавливаем скорость передачи данных с Arduino, а в мониторе порта, мы устанавливаем скорость приема данных компьютером. Эта скорость должна совпадать со скоростью передачи данных от Arduino, иначе данные будут искажаться и вы либо вообще ничего не получите, либо получите “кракозябры”. Это происходит потому что, если скорости будут разные, то компьютер будет просто не в нужный момент считывать данные и не успевать их полностью принимать. Чтобы понять как это работает представьте, что каждую минуту в другой комнате у вас появляется записка с буквой, вы берете ее, уносите в другую комнату на стол и так раз за разом складываете слова. Но представьте что в другой комнате буквы стали появляться чаще чем раз в минуту и вы об этом не зная продолжаете каждую минуту приходить в ту комнату и забирать одну букву, а за эту минуту уже появлялось и исчезало много других букв, которые вы не взяли. В итоге у вас начнут собираться непонятные слова. Точно так же, компьютер должен знать как часто ему нужно считывать символ.

С функцией **Serial.println();** мы разобрались. Помимо того, что мы можем выводить данные на компьютер, мы так же можем и считывать их с компьютера. Для этого существует команда **Serial.read();** давайте напишем код, загрузим в Arduino и протестируем его:

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    if (Serial.available() > 0) {  
        Serial.print("I received: ");  
        Serial.write(Serial.read());  
    }  
}
```

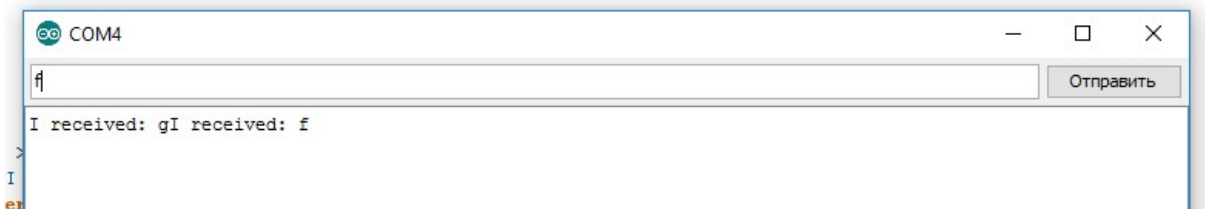
После загрузки кода на плату, откройте монитор последовательного порта, попробуйте написать в поле какойнибудь символ, букву или цифру и нажмите на кнопку **Отправить**,

ниже у вас отобразится строка "I received: " после которой отобразится тот символ который мы отправили. Это означает что наш код выводит нам то, что мы ему напишем.



Давайте разберем код построчно, **Serial.begin(9600);** как мы знаем инициализирует на последовательный порт. Далее идет строка которая проверяет передается ли какая нибудь информация на порт **if (Serial.available() > 0) {**, мы будем разбирать этот код на следующем уроке, поэтому просто запомните, что функция **Serial.available()** проверяет поступила какая нибудь информация или нет. Если какие-то данные передаются на последовательный порт, тогда программа читает код дальше, если нет, то пропускает мимо и опять проверяет передаются ли данные.

Функция **Serial.print("I received: ");** как вы уже наверное догадались эта функция выводит надпись "I received: ", эта функция очень похожа на **Serial.println()**, но все же немного отличается от нее. Функция **Serial.println()** после того как выводила строку, то переводила курсор на следующую строку, а функция **Serial.print();** выводит все одну длинную строку и перенос на следующую строку не делает, по сути надпись **ln** это сокращение **line**. Если мы попробуем отправить еще один символ, то надпись "I received: " отобразится на этой же строке:



Если мы немного переделаем код и добавим в еще строку **Serial.println("");**, то в том месте, где она будет добавлена произойдет перенос строки:

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

void loop() {
    if (Serial.available() > 0) {
        Serial.print("I received: ");
        Serial.write(Serial.read());
        Serial.println("");
    }
}
```

Загрузим код на плату, теперь если мы пошлем несколько символов, то надпись будет выводиться с новой строки каждый раз.



Функция **Serial.read()** считывает данные которые мы передаем из компьютера, при этом функция считывает каждый раз по одному символу, то есть если мы посылаем слово из 10 букв, то функция **Serial.read()** сработает 10 раз, прежде чем получит слово полностью. Ну и наконец функция **Serial.write()**; - эта функция делает почти тоже самое, что и функции **Serial.println()**; и **Serial.print()**; - отправляет данные через последовательный порт, но в отличии от них, **Serial.write()**; не может отправлять целые слова, она может отправлять только по одному символу.

Теперь становится понятно что делает комбинация функций **Serial.write(Serial.read());** функция **Serial.read()** получает один символ и тут же передает его в функцию **Serial.write()**, которая отправляет обратно на компьютер.

Собственно так и работает последовательный порт, эти знания нам понадобятся когда мы будем изучать передачу данных через bluetooth, там тоже будет использоваться **Serial**, это поможет нам управлять с помощью bluetooth на телефоне - любыми устройствами.

Урок №3

Тема: Математические операторы, переменные, знакомство с условными операторами

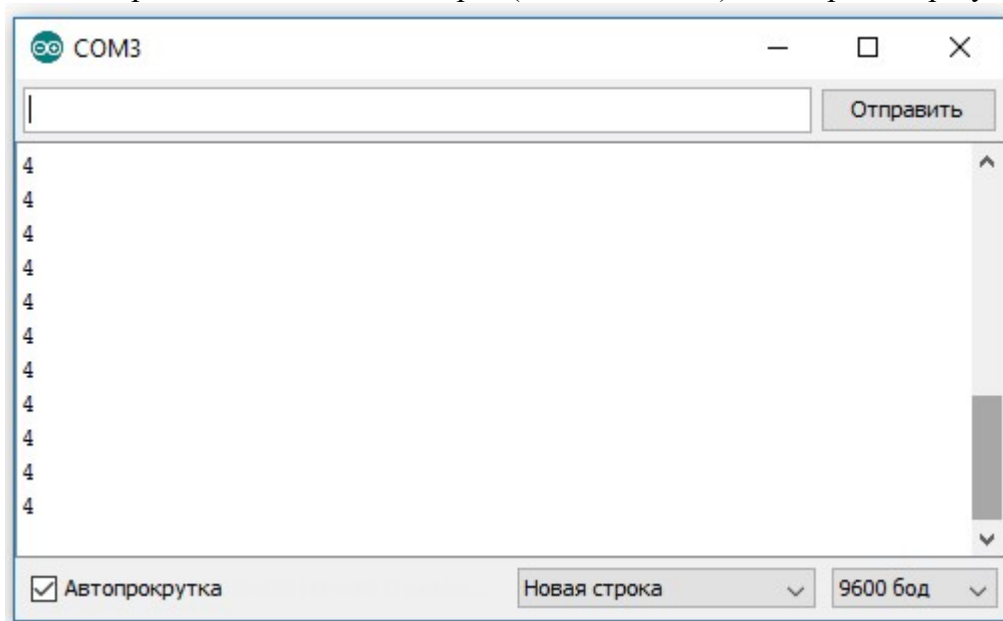
На этом занятии немного углубимся в изучение программирования микроконтроллеров. Сегодня мы узнаем, что в программировании есть математические функции, а так же познакомимся с условиями и переменными.

Давайте начнем с простого, с математических операторов: в программировании как и в математике можно выполнять математические операции с числами: Делить, умножать, складывать и вычитать. На прошлом занятии мы научились передавать информацию с Arduino на компьютер через Serial порт. Мы выводили простые текстовые строки строки, помимо строк можно выводить и числа, например `Serial.println(2);` передаст на последовательный порт число 2. А давайте попробуем передать на последовательный порт `Serial.println(2+2);` и посмотрим что отобразится на экране, думаю вы догадываетесь, загрузим

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(2+2);  
  delay(1000);  
}
```

Теперь открываем этот код в Arduino:

Открываем Монитор последовательного порта (**ctrl+ shift + M**) и смотрим на результат:

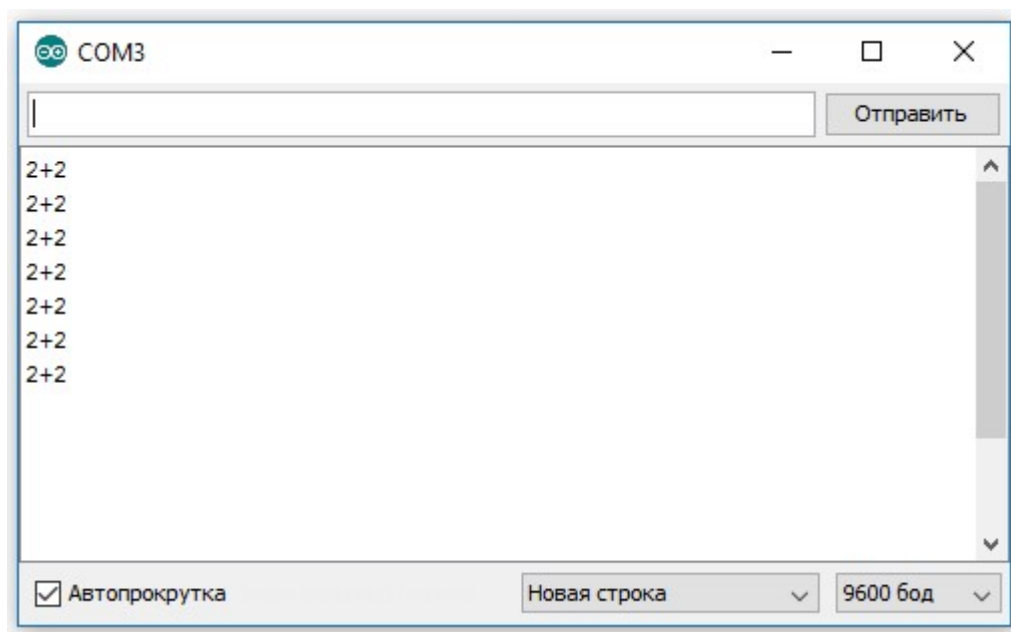


В результате у нас выводится 4 - решение сложения двух чисел 2+2, то есть программа сама посчитала сумму и вывела нам уже готовый ответ. Но обратите внимание что мы писали сумму чисел 2+2 без кавычек. Если вы помните то на прошлом занятии строки мы выводили в кавычках `Serial.println("Loop");`.

Давайте попробуем заключить сумму “2+2” в кавычки и посмотреть, что из этого получится:

```
void setup() {  
  Serial.begin(9600);  
}  
void loop() {  
  Serial.println("2+2");  
  
  delay(1000);  
}
```

Теперь загружаем код на плату, запускаем монитор порта и наблюдаем:



Из результата можно сделать вывод, что когда математическое выражение заключено в кавычки то программа выводит его таким, какое оно есть. Для программы всё, что заключено в кавычки является обычной строкой и она даже не пытается складывать, вычитать и выполнять любые другие действия. Это сделано специально, вдруг вам не нужно чтобы программа считала, а вы хотите просто вывести то, что написали, тогда заключение в кавычки будет очень полезным.

Помимо сложения программа может так же выполнять и другие математические операции. Для этого есть свои операторы, основные из них мы привели в таблице ниже:

+	сложение
-	вычитание
/	деление
*	умножение

Попробуйте с каждым оператором выполнить действие и вывести результат, например:

Serial.println(2); - здесь у нас на экран отобразится число 2

Serial.println(6*2); - результатом умножения отобразится число 12

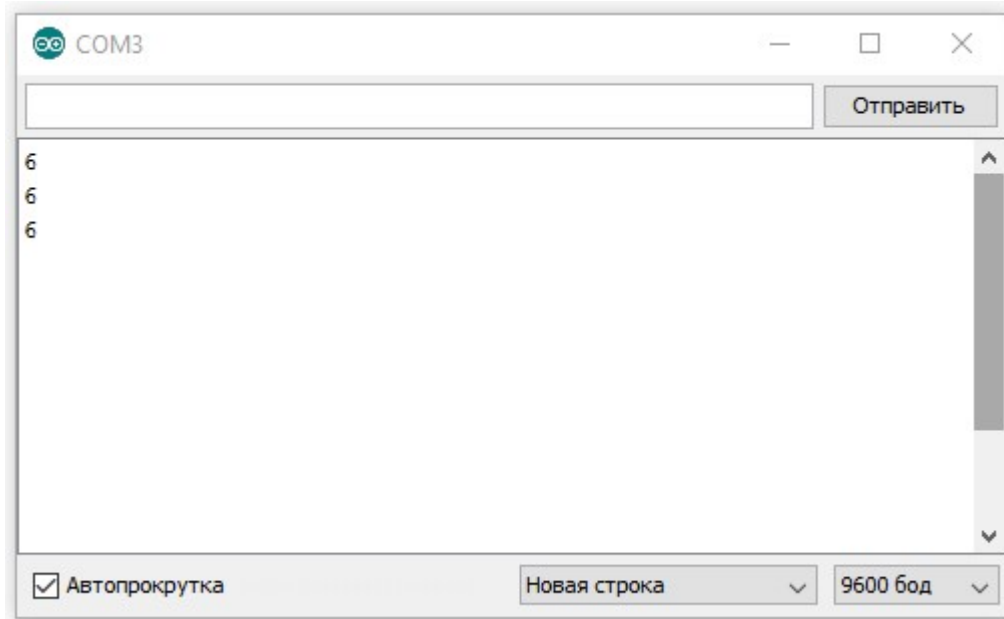
Serial.println(6-2); - ну и здесь тоже отобразится цифра 4

В программировании так же как и математике у каждой операции есть свой приоритет, например, например умножение и деление всегда выполняется раньше, чем сложение и вычитание. Думаю многие из вас слышали математическую задачку с подвохом: “Сколько будет два плюс два умножить на два”, многие ошибаются и говорят ответ: 8, потому что они сначала выполняют действие $2 + 2$ следовательно получается 4, после этого мы еще раз умножаем на 2 и выходит, что $4 * 2 = 8$. Но это неверно! Потому что сначала нужно производить умножение, а после этого производить сложение. Соответственно правильный ответ будет 6, потому что $2*2 = 4$ и $4 + 2 = 6$.

Так вот, к чему это говорится - программа всегда правильно определяет порядок действий, чтобы это проверить давайте выведем в последовательный порт это выражение: $2+2*2$ напомним, что если вы хотите чтобы программа посчитала математические действия, то нужно писать их без кавычек:

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(2+2*2);  
  delay(1000);  
}
```

В итоге у нас должен выводиться ответ:



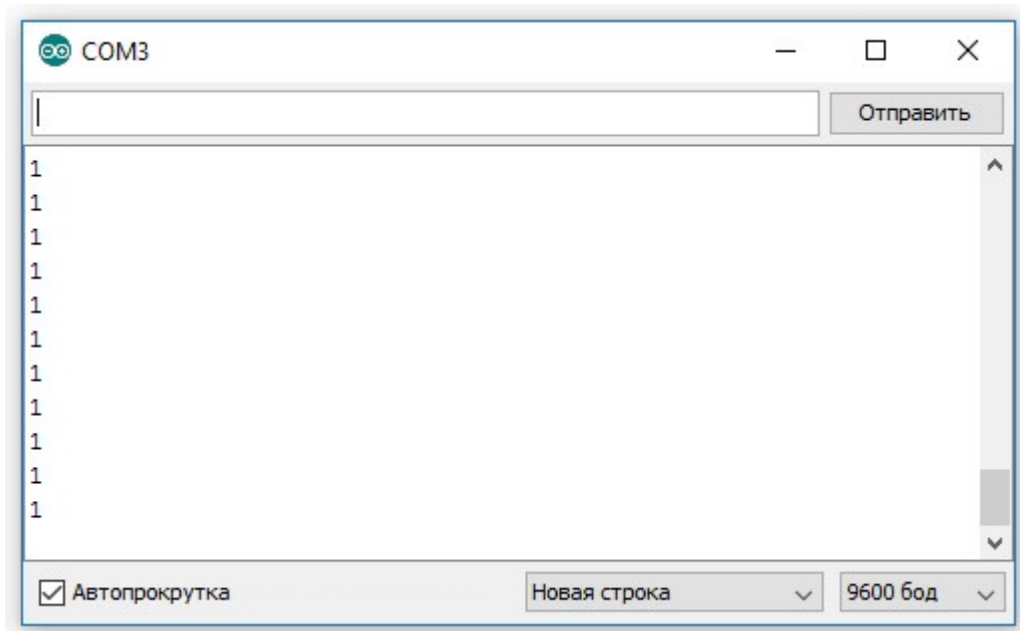
Программа так же понимает когда математические операторы заключаются в круглые скобки, например если заключить в скобки наше выражение $(2+2) * 2$, то программа учтет это и выдаст от 8, попробуйте это сделать сами:

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println((2+2)*2);  
  delay(1000);  
}
```

Давайте рассмотрим еще такой пример, попробуем разделить $5/3$, на сколько нам известно то получится бесконечно длинное число 1,66666666666666... и т.д. Но если мы заставим нашу программу эту посчитать и выдать ответ, то ответ будет другим:

```
Serial.println(5/3);
```

Тогда открыв монитор последовательного порта мы увидим, что ответ будет 1:

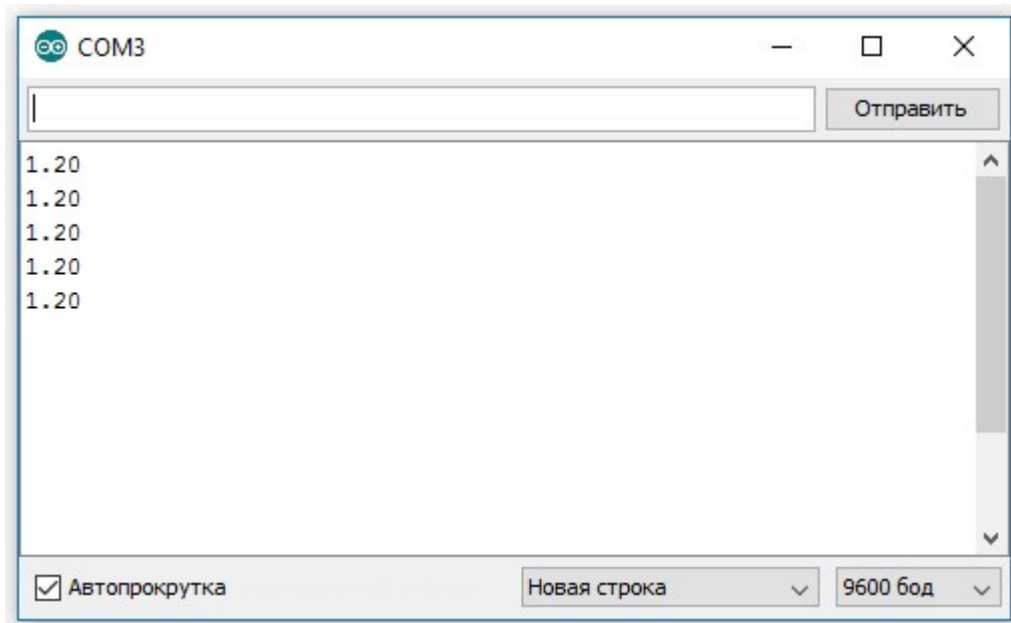


Программа выдала такой ответ округлив 1,6 до 1 (просто отрезала дробную часть) потому что подумала, что мы работаем с целыми числами и нам не нужна дробная часть.

Чтобы дать программе понять, что нам нужен ответ поточнее включая дробную часть, нам нужно в выражении писать дробные числа, например если мы разделим $2.4/2$:

```
Serial.println(2.4/2);
```

то программа выдаст ответ 1.20

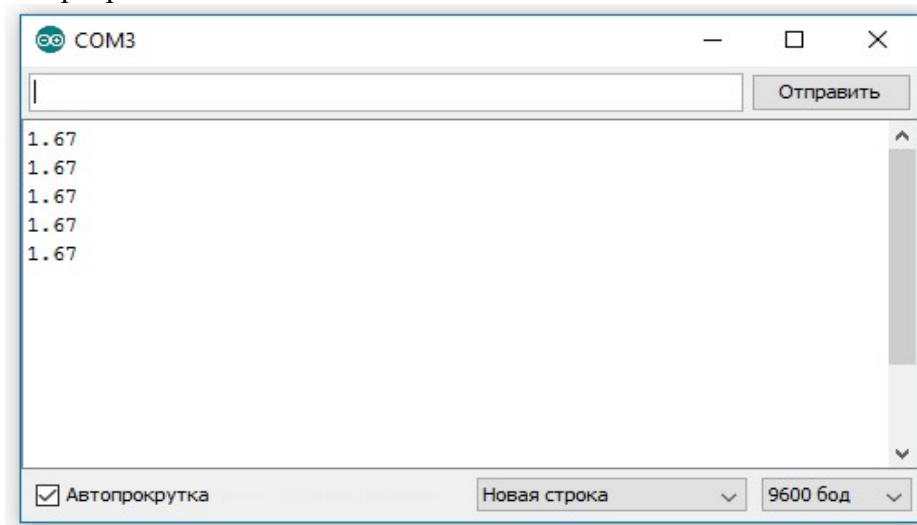


Получается что если одно из чисел участвующих в математических операциях будет дробным, то программа выдаст дробный ответ. Но когда мы делили $5/3$ у нас оба числа были целыми, а ответ нужен дробный. Чтобы получить дробный ответ достаточно около любой из цифр поставить точку и ноль, например $5.0/3$ или $5/3.0$ или $5.0/3.0$.

Давайте попробуем наш старый пример по новому

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(5.0/3.0);  
  delay(1000);  
}
```

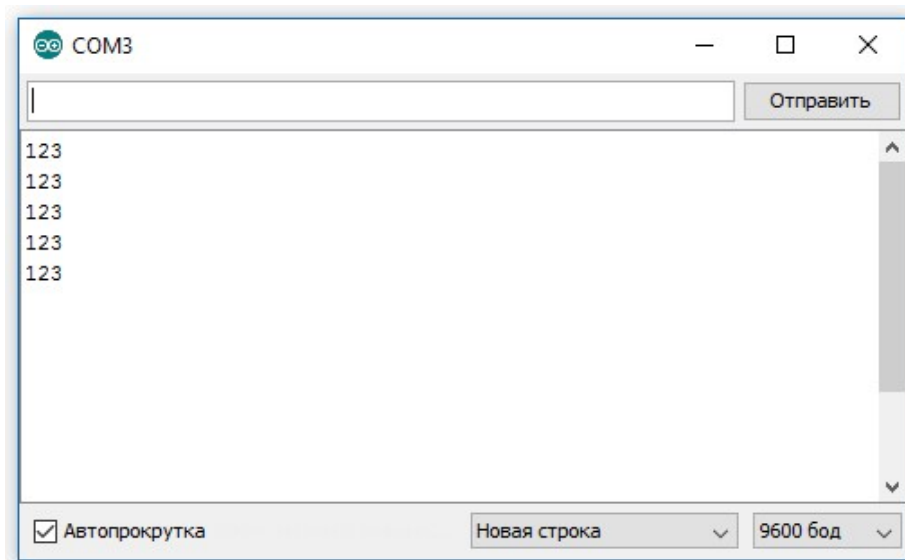
И мы видим что программа выдает более точный ответ:



На этом мы пока остановим изучение операторов, и перейдем к изучению переменных. Что такое переменная и с чем ее едят? Самый лучший способ это увидеть самому, давайте напишем следующий код.

```
int korobka;  
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  korobka = 123;  
  Serial.println(korobka);  
  delay(1000);  
}
```

Теперь загружаем код в Arduino и смотрим результат:



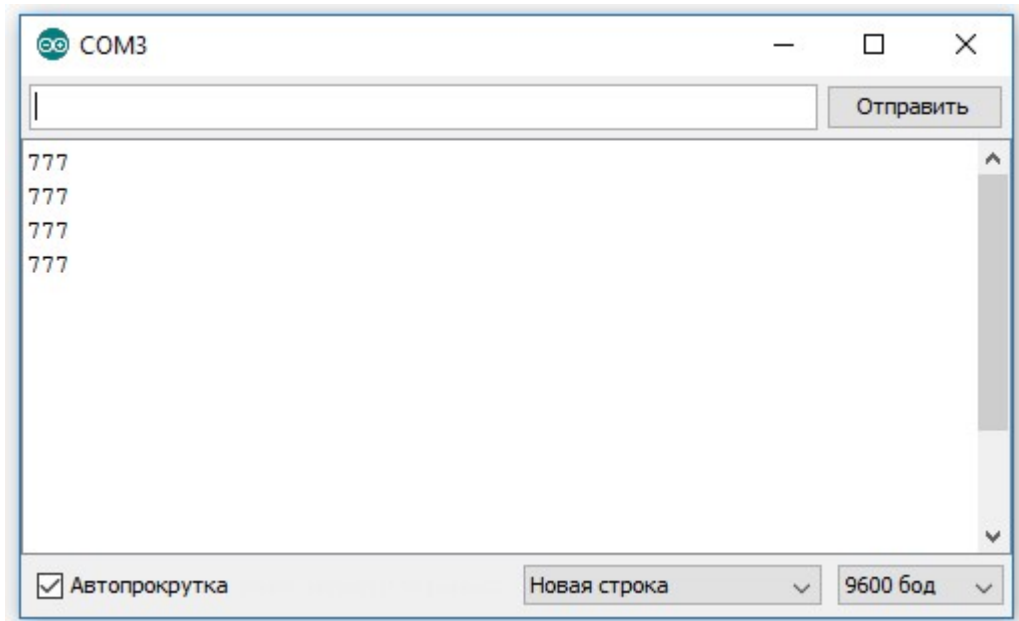
Как мы видим - в монитор порта выводится число 123, если мы изменим код и вместо числа 123 любое другое, то в монитор порта будет выводиться, то число которое мы напишем, например изменим строку

```
korobka = 123;
```

на такой код:

```
korobka = 777;
```

то на в монитор порта у нас будет выводиться три семерки:

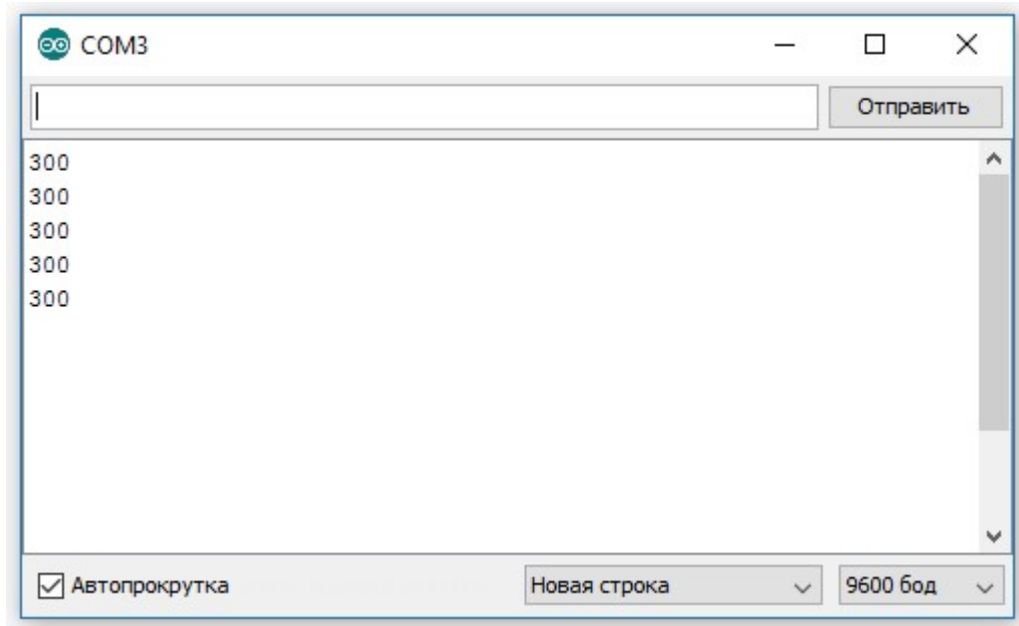


Так вот, теперь можно лучше объяснить что такое переменная, *korobka* - это переменная. Переменная в программировании представляет собой некий ящик в который можно сложить информацию, потом ее использовать, выходит так что программа вытаскивает из переменной ту информацию которую мы в нее положили и использует, в нашем случае программа выводит это значение в сериал монитор. Грубо говоря программа если в переменную положить число **777** (***korobka* = 777;**), то программа будет относиться к переменной ***korobka*** как будто она и есть число **777**. Если мы например напомним ***korobka* + 1**, то программа выдаст нам ответ **778**, то есть программа прибавляет к числу которое лежит в переменной - цифру **1** и выдает нам ответ, давайте попробуем поэкспериментировать, напомним следующий код:

```
int korobka;
void setup() {
  Serial.begin(9600);
}

void loop() {
  korobka = 600;
  Serial.println(korobka/2);
  delay(1000);
}
```

Загрузим в Arduino и посмотрим, что произойдет:



Программа разделила число сохраненное в переменной - 600 на 2 и выдала нам ответ. Думаю вы уже поняли, что для того чтобы сохранить в переменную число, нам необходимо написать название нашей переменной, поставить после нее знак равенства “=” и после этого написать число, которое мы хотим в нее сохранить.

Возможно вы так же догадались что, слово `korobka` было “взято с потолка”, потому что называть переменные можно любыми именами, это означает что вместо слово `korobka`, можно написать любое другое слово, например `box`, `container`, `blablabla`, `peremennaya1`, `peremennaya2`, `a`, `b`, `c`... и так далее, в названии переменных вы можете использовать любые латинские (английские) буквы, цифры и даже знак подчеркивания “_”. но первым символом в названии обязательно должна быть буква, например можно назвать переменную `box2`, но нельзя назвать переменную `2box`. Чтобы создать новую переменную, с новым именем, вы должны ее написать (объявить) до того как как сохранять и считывать из нее данные, Создается переменная очень просто, для этого, как вы могли заметить, в самом начале кода, перед функцией `setup()` { } мы написали

`int korobka;` - эта строчка и создала переменную `korobka`. Если бы мы указали здесь другое название то создали бы переменную с другим названием, все логично и просто.

Теперь по подробнее о команде **`int`**, **`int`** - это сокращение слова **`integer`** (в переводе - *целое число, целочисленный*), на самом деле это даже не команда, написав **`int`** мы задали целочисленный тип нашей переменной. Что это значит? Это означает что мы можем сохранять в нашу переменную только целые числа, если мы попытаемся сохранить в нее дробное число, то программа округлит это дробное число и все равно сохранит целое число. А если мы захотим сохранить в переменную дробное число? Для этого есть тип данных **`float`**, то есть если мы создадим переменную **`float korobka;`**, то сможем сохранить в нее дробное число, давайте попробуем это сделать:

```
float korobka;
void setup() {
  Serial.begin(9600);
}

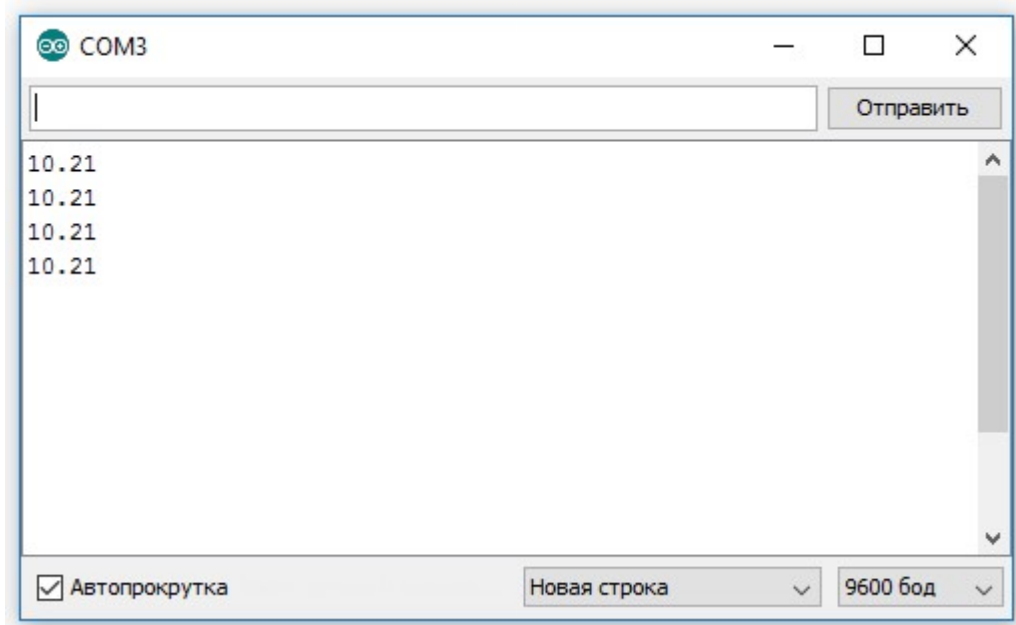
void loop() {
```

```

korobka = 20.42;
Serial.println(korobka/2);
delay(1000);
}

```

Программа выдаст дробный ответ, так как одно из чисел у нас является дробным:



Есть еще одно ограничение связанное с переменными, в переменную типа **int** нельзя сохранить число меньше чем -32768 и больше чем 32767. Вы должны это всегда учитывать при написании кода, сохранение чисел размером больше чем ограничение может привести к неожиданным результатам.

int и **float** это не единственные типы данных, в таблице ниже мы привели еще несколько типов, которые можно использовать в Arduino:

Тип данных	Диапазон значений
int	от -32768 до 32767
unsigned int	от 0 до 65535
long	от -2 147 483 648 до 2 147 483 647
unsigned long	от 0 до 4 294 967 295
float	от -2 147 483 648.0 до 2 147 483 647.0

Если вы заметили, то в некоторых названиях присутствует слово **unsigned** - в переводе означает “без знака”, это означает что в таких типах данных нельзя использовать такой знак минуса. Это позволяет увеличить максимальное значение для такой переменной.

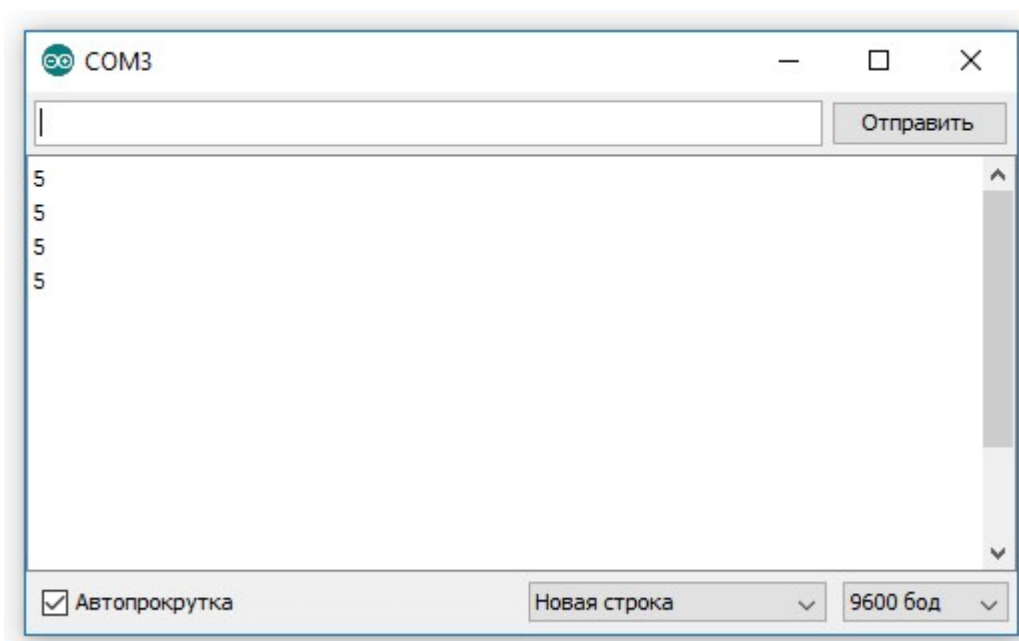
Попробуйте создать переменную с каждым из этих типов данных, сохранить и вывести максимальные значения.

Давайте рассмотрим еще одно свойство переменных. Что например будет если мы напишем в коде так:

```
int korobka;
void setup() {
  Serial.begin(9600);
}

void loop() {
  korobka = 20;
  korobka = 5;
  Serial.println(korobka);
  delay(1000);
}
```

Как вы думаете какое число будет показано программой в мониторе последовательного порта?



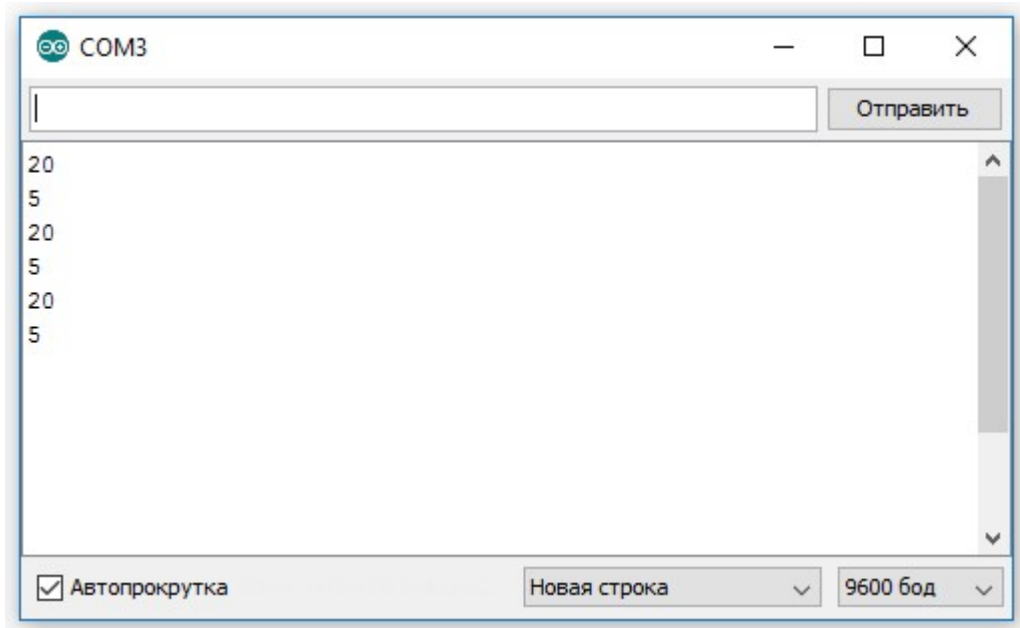
Как мы видим отобразилось число 5, но мы ведь еще и сохраняли в переменную число 20, куда же оно делось? Дело в том, что если мы сохраняем в переменную число, то предыдущие данные из нее сразу удаляются. На самом деле число 20 было в переменной korobka, но было это очень не долго. Если бы мы попробовали сразу после сохранения числа 20 в переменную korobka вывести ее в последовательный порт, то увидели бы, что число есть в переменной, давайте попробуем:

```
int korobka;
void setup() {
  Serial.begin(9600);
}

void loop() {
```

```
korobka = 20;
Serial.println(korobka);
korobka = 5;
Serial.println(korobka);
delay(1000);
}
```

Как видите, на экран выводится и число 20 и число 5, несмотря на то что мы выводим одну и ту же переменную, в разное время у нее разные значение.



Поэтому переменные и называются и называются “Переменные”, потому что значения в них могут изменяться, переменяться.

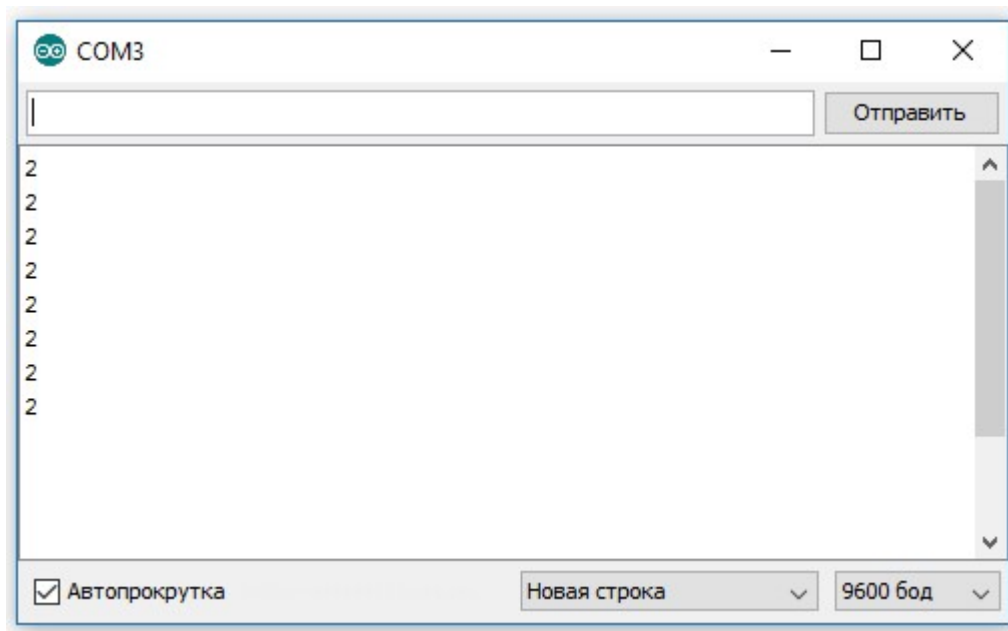
Как уже говорилось, с переменными мы можем выполнять все те же математические действия, что и с числами, например давайте создадим 2 переменные, сохраним им 2 числа и попробуем разделить одну переменную на другую и посмотрим на результат:

```
int box1;
int box2;
int x;
void setup() {
  Serial.begin(9600);
}

void loop() {
  box1 = 20;
  box2 = 10;
  x=box1+box2;
  Serial.print(box1);
  Serial.print("+");
  serial.print(box2);
}
```

```
serial.print("=");  
serial.println(x);  
Serial.println(box1 / box2);  
delay(1000);  
}
```

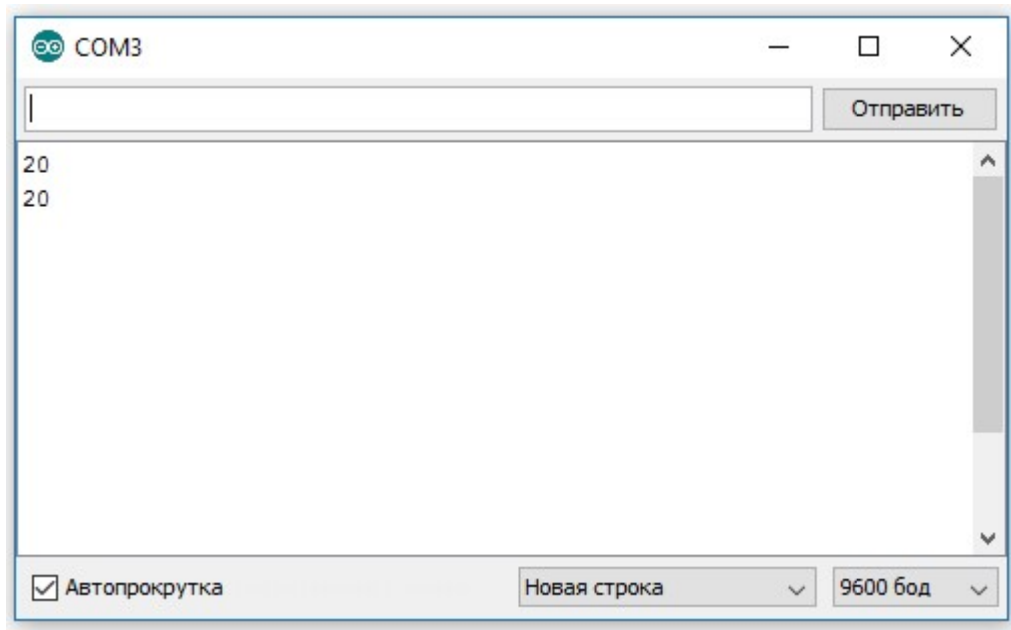
Как мы видим программа прочла из переменной box1 значение 20, а из переменной box2 значение 10, выполнила деление и вывела ответ 2:



В переменные так же можно сохранять другие переменные, например

```
box1 = 20;  
box2 = box1;  
Serial.println(box2);
```

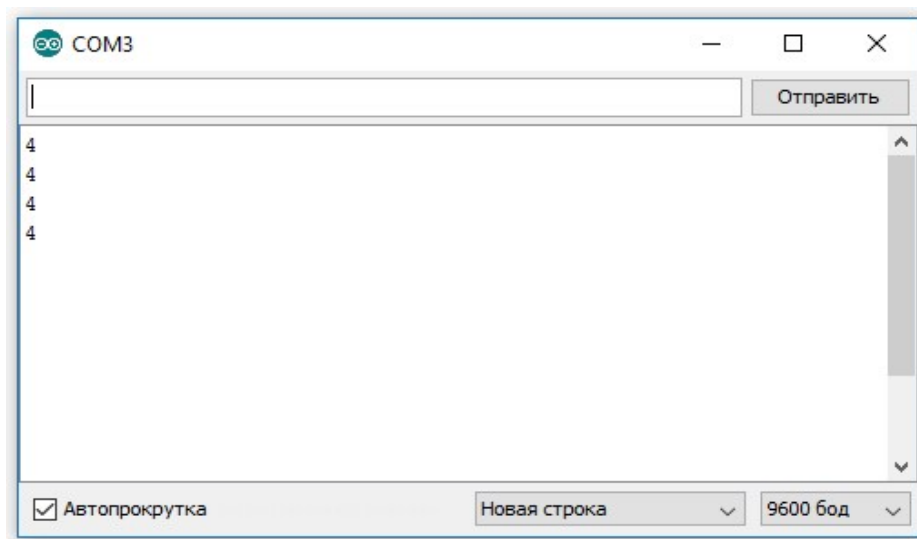
В этом примере мы сохранили в переменную box1 число 20, после этого в переменную box2 мы сохранили переменную box1, в итоге у в переменной box2 у нас тоже будет число 20, потому что когда мы написали box2 = box1, то программа посмотрела, что хранится в переменной box1, считала это число и записала его в переменную box2, теперь в обоих переменных у нас число 20 и при выводе в монитор последовательного порта Serial.println(box2); у нас отобразится число 20.



В переменную так же можно сохранять самих себя, например

```
box1 = 2;  
box1 = box1 + box1;  
Serial.println(box1);
```

В этом примере мы сначала сохранили в переменную box1 число 2, а после этого в переменную box1 мы сохраняем box1 + box1, что при этом делает программа? Сначала у нас в переменной box1 лежит число 2, дальше когда мы пишем box1 = box1 + box1 то программа сначала выполняет код который находится справа от скобок, то есть она выполняет box1 + box1, мы знаем что в них у нас сейчас лежит число 2, в итоге когда мы пишем box1 + box1 - это все равно, если бы написали 2 + 2. Программа вычисляет что box1 + box1 будет 4 и эту четверку сохраняет обратно в переменную box1, теперь в переменной box1 у нас лежит число 4, команда Serial.println(box1); выведет в монитор порта число 4

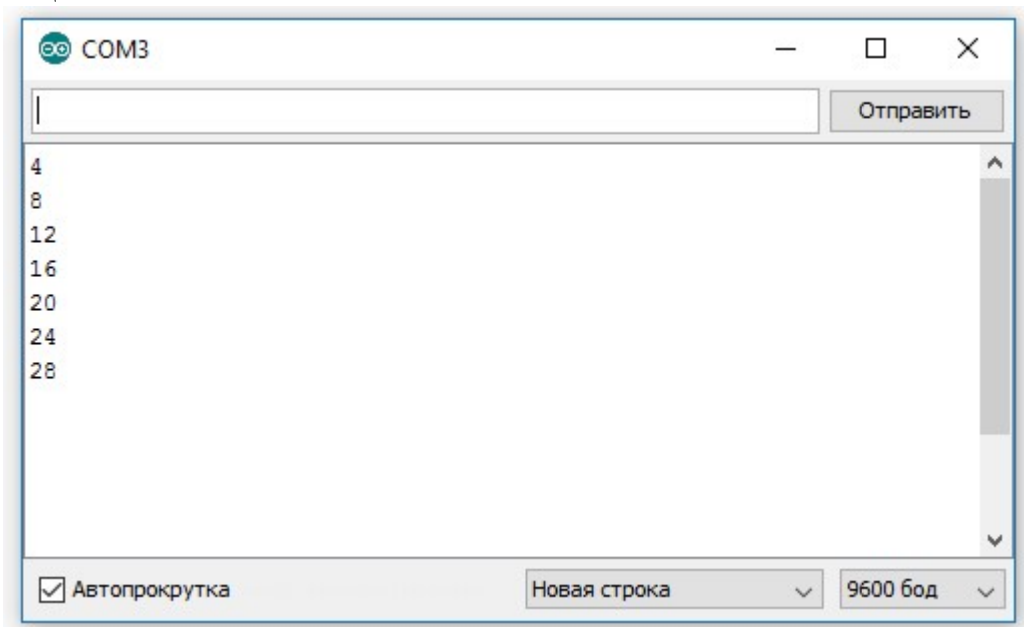


Давайте рассмотрим еще 1 пример, напомним такой код и посмотрим, что он будет делать:

```
int box1;
void setup() {
  Serial.begin(9600);
  box1 = 0;
}

void loop() {
  box1 = box1 + 4 ;
  Serial.println(box1);
  delay(1000);
}
```

После загрузки программы на плату в монитор порта у нас будет выводиться числа каждый раз увеличивающиеся на 4



Если вы не поняли почему так происходит то давайте по порядку разберем, в самом начале в `setup() {}` у мы сохраняем в переменную `box1` число 0. Далее мы присваиваем `box1 = box1 + 4;`, как и в предыдущем примере программа сначала считает всё выражение после знака равно, то есть `box1 + 4`, соответственно раз у нас в переменной сейчас число 0, то `box1 + 4` будет равняться 4, это число мы пересохраняем обратно в переменную `box1`, теперь в переменной `box1` число 4. Далее мы просто выводим это число в монитор порта и делаем паузу 1 секунду. Далее цикл `loop` снова повторяется, снова высчитывается `box1 = box1 + 4;`, но так как сейчас у нас в переменной `box1` уже не 0 а 4, то `box1 + 4` будет равняться 8 и число 8 сохраняется обратно в переменную `box1`, теперь в переменной `box1` у нас число 8 и в следующий раз `box1 + 4` будет равен 12 и т.д

Точно так же можно делать с вычитанием у умножением или делением, например:

```
box1 = box1 - 1;
box1 = box1 * 2;
box1 = box1 / 3;
```

Попробуйте это сделать эти примеры сами.

В программировании такие приемы используются очень часто, когда к одной переменной прибавляют все время какое-то число, поэтому для этого была придумана более короткая запись, в предыдущем примере мы писали так:

```
box1 = box1 + 4;
```

То же самое действие в программировании можно написать намного короче:

```
box1 += 4;
```

Обе записи правильные и делают абсолютно одно и то же, они прибавляют к переменной box1 число 4;

Точно так же можно выполнять и деление, вычитание, умножение, например такая запись:

```
box1 = box1 - 1;  
box1 = box1 * 2;  
box1 = box1 / 3;
```

и такая запись делают одно и то же действие:

```
box1 -= 1;  
box1 *= 2;  
box1 /= 3;
```

Попробуйте сами повторить эти примеры.

Ну и напоследок, существуют еще 2 оператора в программировании которые позволяют прибавлять и отнимать это: ++ и --; Оператор ++ прибавляет единицу, а оператор -- отнимает единицу, например:

```
box1 = 6;  
box1++;
```

В переменную box1 мы сначала сохранили число 6, после этого мы написали box1++, теперь у нас в переменной box1 будет находится число 7, потому что оператор ++ прибавляет единицу. Еще 1 пример:

```
box1 = 6;  
box1--;
```

В переменную box1 мы сначала сохраняем 6, после этого вызываем команду box1-- и от нашей переменной отнимается единица, теперь в переменной box1 находится 5. Попробуйте сами сделать эти примеры.

На этом на сегодня достаточно разговора о переменных. У нас есть еще одна важная тема - условные операторы.

Условные операторы.

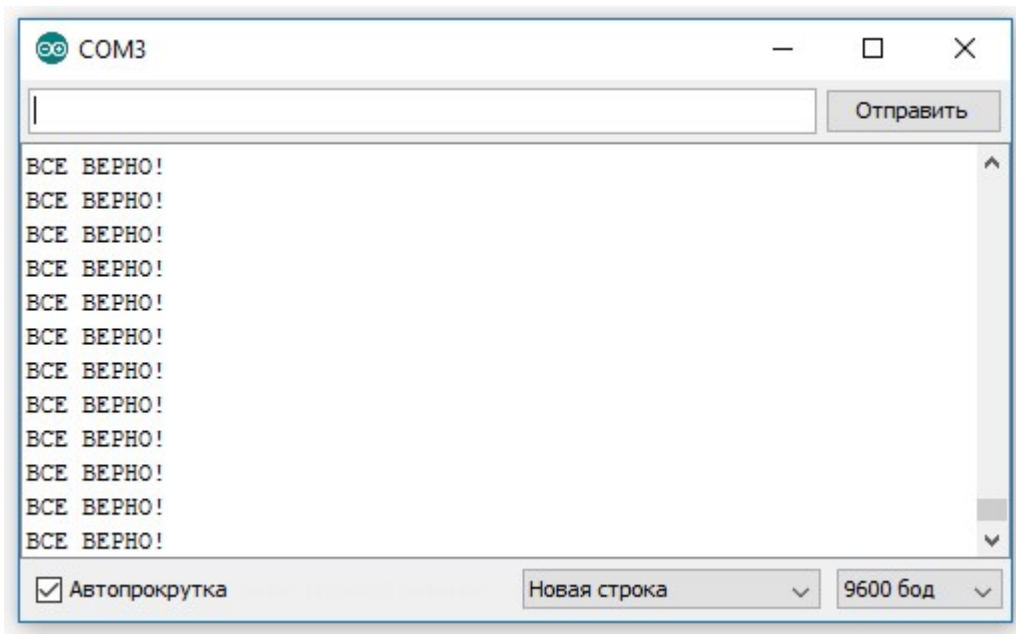
Что такое условные операторы? Что опять за новое и не ясное определение?

Думаю в реальной жизни вы знаете, что такое условия? Например раньше вам наверное родители говорили: “Если не сделаешь уроки, то не пойдешь играть!” или учителя в школе “Если не выполните домашнее задание, то получите двойку!” и т.д.

В программировании тоже есть условия, без них бы программирование было абсолютно бесполезным, в нем так же как и в жизни вы можете задавать программе условия, например “Если датчик движения сработал, включить сигнализацию!” или “Если сейчас 22.00, то зажечь свет во дворе!”, но условия конечно же пишется не простыми предложениями как вы поняли, а для этого есть специальный условный оператор() **if{ }**. Слово **if** с английского означает “если”. Внутри круглых скобок мы пишем условия которые мы хотим проверять, а внутри фигурных скобок мы пишем код, который должна выполнить программа если условие будет выполнено. например

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  if(5 > 0){  
    Serial.println("ВСЕ ВЕРНО!");  
  }  
}
```

В этом коде мы выводим в монитор последовательного порта надпись ВСЕ ВЕРНО! в том случае если 5 больше 0, ну мы то знаем с вами что 5 по любому больше нуля (*внимание все буквы здесь написаны на английском - все верно!, только в верхнем регистре (зажата CAPS LOCK)*)



А если же мы бы написали что 5 меньше 0 то условие бы не выполнилось, давайте проверим

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  if(5 < 0){  
    Serial.println("BCE BEPHO!");  
  }  
}
```

Теперь если мы запустим программу и откроем монитор порта, то не увидим в нем ничего, потому что условие не верно, 5 никогда не станет меньше 0 и программа никогда не выведет не выполнит действие **Serial.println("BCE BEPHO!");**.

Как вы заметили чтобы сравнить два числа, как и в математике нужно поставить между ними знак сравнения больше или меньше, как и в математике в программировании есть и другие символы сравнения:

Символ сравнения	Что он означает
>	больше - условие верно, когда левая цифра больше правой
<	меньше - условие верно, если левая цифра меньше правой
==	равно - условие верно, когда обе цифры равны
!=	не равно - условие верно когда цифры не равны
<=	меньше либо равно - условие верно когда левая цифра меньше либо равно правой цифре
>=	больше либо равно - условие верно когда левая цифра больше либо равно правой цифре

Например если напишем код:

```
if(5 == 5){  
  Serial.println("BEPHO!");  
}
```

Тогда на экран у нас будет выведена надпись BEPHO!, потому что 5 равно 5, если мы напишем код:

```
if(5 != 5){  
  Serial.println("BEPHO!");  
}
```

Тогда на экран ничего не будет выведено, потому что 5 равно 5, а мы даем команду вывести

надпись только в том случае если $5 \neq 5$, так как такого быть не может, то на экран ничего не будет выведено.

В программировании так же можно задать программе выполнять код, если условие не было выполнено. Например, возможно вам когда-то говорили “Если будешь хорошо кушать то вырастишь большим и сильным, а иначе останешься маленьким”.

Так и в программировании можно задать код, который выполнится если условие не верное, для этого существует оператор **ELSE{ }**, в переводе с английского означает - иначе. Давайте рассмотрим как он работает:

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

void loop() {
  if(5 < 0){
    Serial.println("ВЕРНО!");
  } else {
    Serial.println("НЕ ВЕРНО!");
  }
}
```

Как вы наверное поняли то, то код который написан в фигурных скобках после оператора **else** будет выполнен в том случае если условие которое прописано в **if()** не верно, то есть сейчас у нас на экран должна вывестись надпись “НЕ ВЕРНО!”, потому что в условии у нас стоит 5 меньше 0, а так как 5 у нас на самом деле больше 5, значит условие не верное и программа выполнит код, который написан в **else { Serial.println(“НЕ ВЕРНО!”); }**

У оператора **if** есть еще очень удобное свойство - **else if(){ }**. Оно как и **else{ }** тоже выполняется в том случае если условие прописанное в **if()** не верно. Но в отличие от **else{ }** в **else if(){ }** можно прописывать свои дополнительные условия.

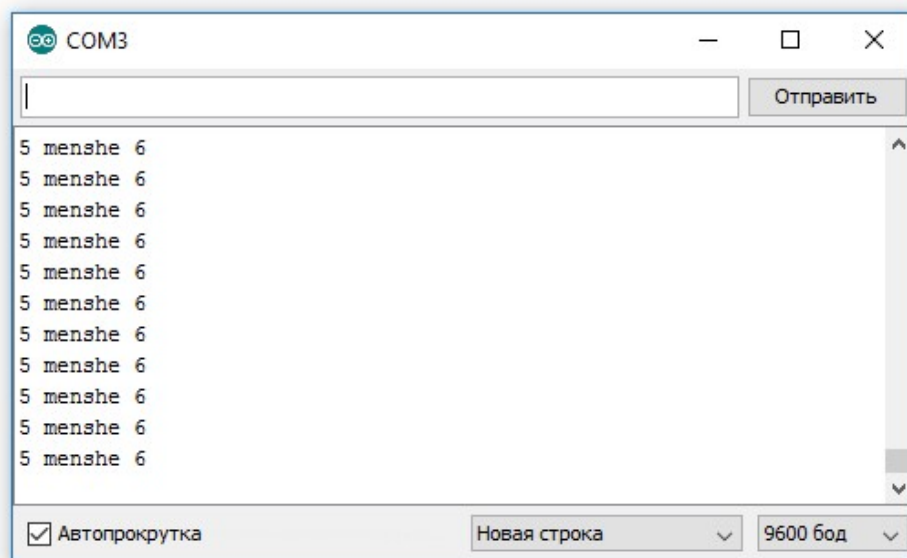
Приведем пример:

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

void loop() {
  if(5 < 0){
    Serial.println("5 menshe 0");
  } else if( 5 < 3) {
    Serial.println("5 menshe 3");
  } else if( 5 < 6) {
    Serial.println("5 menshe 6");
  } else if( 5 < 10) {
    Serial.println("5 menshe 10");
  }
}
```

Теперь если загрузить этот код, мы видим, что выводится только последняя надпись "5 menshe

6" это потому, что выполнилось условие $5 < 6$, даже не смотря на то, что условие $5 < 10$ верное, надпись "5 menshe 10" все равно не выведется в serial потому, что программа перестает проверять условия, как только первое из них выполнилось.



А что если нам нужно будет проверить 2 условия

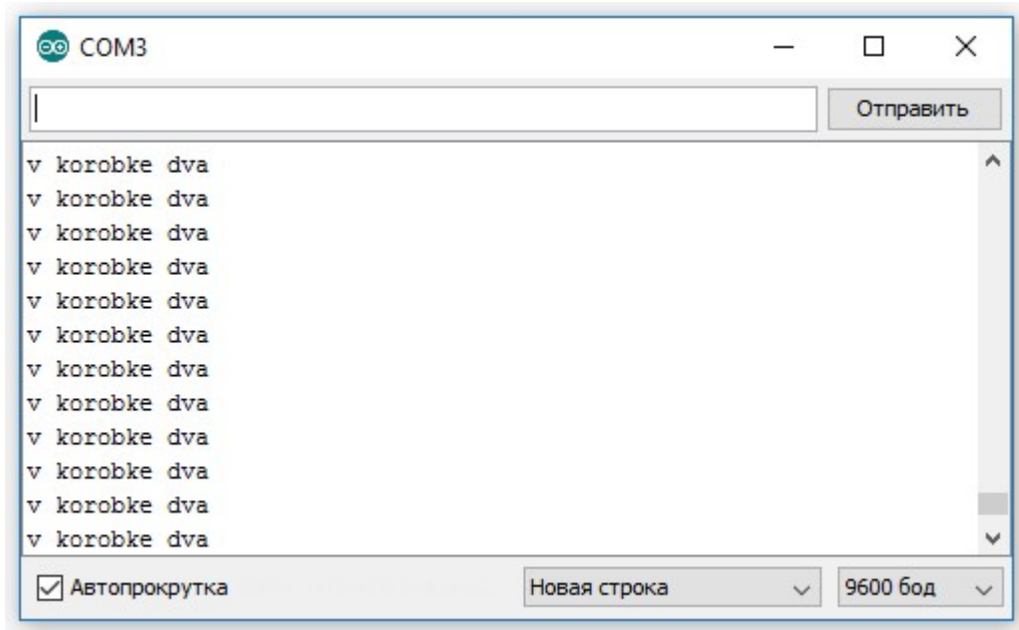
Кроме условного оператора **if** существует так же оператор **switch() { case: break;}**

Делает он тоже самое, что и оператор **if**, но немного по другому, как всегда давайте напишем пример кода, выполним его и нам станет понятно как работает **switch()**.

```
int korobka;
void setup() {
  Serial.begin(9600);
}

void loop() {
  korobka = 2;
  switch(korobka){
    case 1:
      Serial.println("v korobke odin");
      break;
    case 2:
      Serial.println("v korobke dva");
      break;
    case 3:
      Serial.println("v korobke tri");
      break;
  }
}
```

Загружаем программу в Arduino, открываем монитор порта и смотрим что у нас получилось:



Мы видим, что на экран вывелась надпись “v korobke dva”, она вывелась программой потому что в переменной **korobka** было сохранено число 2. Мы поставили в условие **switch** переменную **korobka**, и оператор **switch** сравнивал нашу переменную с каждым значением из блоков **case**, так как в первом блоке **case** была цифра 1, то программа проверила, равно ли значение в переменной цифре 1, оно оказалось не равно и программа пошла дальше. В следующем **case** у нас находится цифра 2, тут программа опять проверяет равно ли число, которое лежит в нашей переменной цифре 2, и так как в переменной **korobka** лежит как раз цифра 2, то проверка прошла успешно и выполнялся код который находится во втором кейсе. Команда **break;** делает тоже самое что и закрывающая фигурная скобка “}” в условиях **if**, она закрывает блок **case**, то есть “**case...break;**” это тоже самое что и “{...}”, только пишется длиннее.

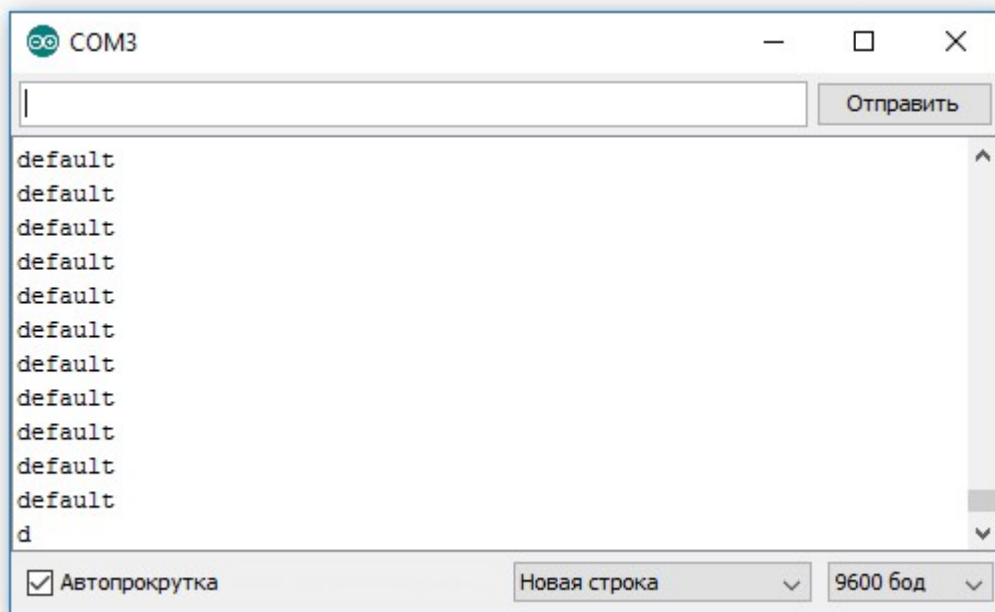
Как в условии **if** у нас есть **else{...}**, точно так же и **switch** у нас есть условие **default:...break;**, оно выполняется в том случае, если ни один **case:** не сработал, например если мы возьмем и переменной **korobka** присвоим например 5, то не сработает ни один **case**, и если добавим **default**, то сработает именно он:

```
int korobka;
void setup() {
  Serial.begin(9600);
}

void loop() {
  korobka = 5;
  switch(korobka){
    case 1:
      Serial.println("v korobke odin");
      break;
    case 2:
      Serial.println("v korobke dva");
      break;
    case 3:
```

```
Serial.println("v korobke tri");  
break;  
default:  
    Serial.println("default");  
break;  
}  
}
```

Теперь если загрузить прошивку, то на экране у нас отобразится надпись “default”:



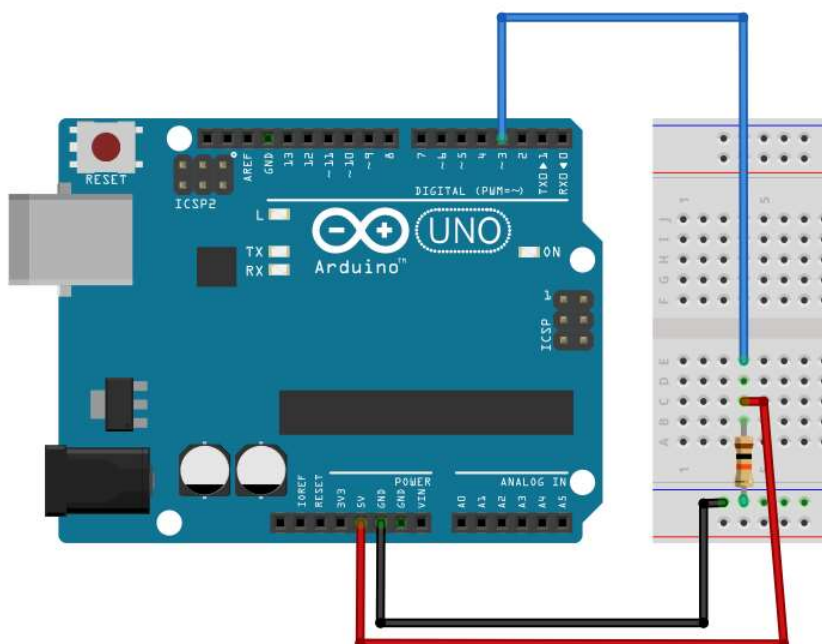
Условный оператор **switch** используется не так часто как **if**, поэтому мы не будем на нем сильно заострять внимание.

Урок №4

Тема: Работа с портами ввода и вывода, константы, булевы переменные

Сегодня мы будем учиться считывать данные с цифровых портов Arduino. Подключим кнопку и будем с помощью нее управлять различными действиями. Для того, чтобы считывать данные с цифровых портов, в Arduino есть специальная функция **digitalRead()**. Давайте подключим Arduino к макетной плате как показано на рисунке ниже:

Порт Arduino	Макетная плата
+5V	Резистор1 10кОм
GND	Резистор1 10кОм
3 цифровой порт	+5V Arduino



ina

В итоге получается, что мы подключили 5V от Arduino напрямую к 3 цифровому порту, а также соединили 3 цифровой порт с землей через резистор, зачем мы это сделали я объясню позже. Теперь напишем следующий код:

```
int pin;  
void setup() {  
  Serial.begin(9600);  
  pinMode(3, INPUT);  
}  
void loop() {
```

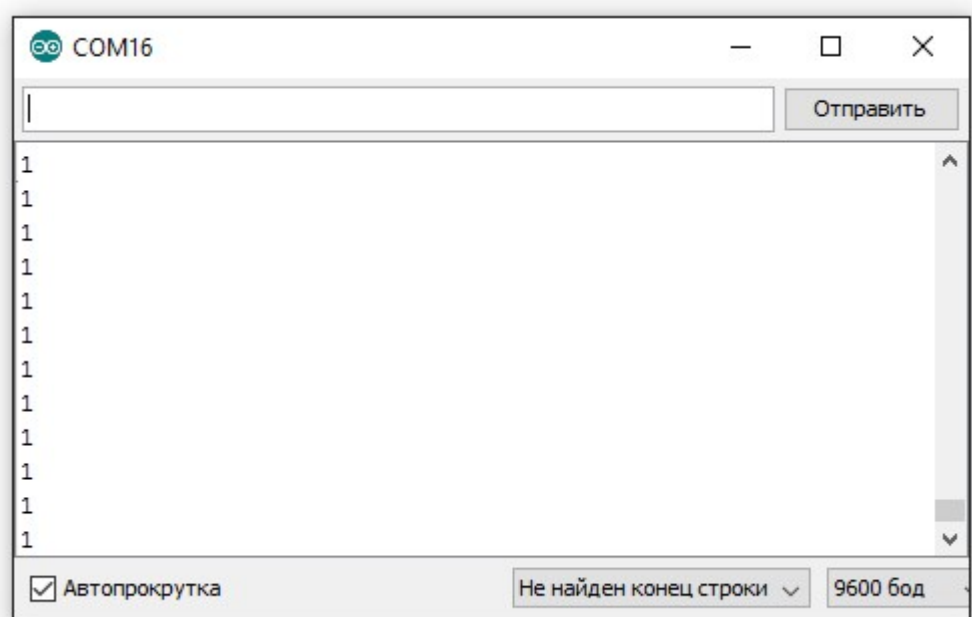
```
pin = digitalRead(3);  
Serial.println(pin);  
}
```

Давайте, теперь построчно разберем код.

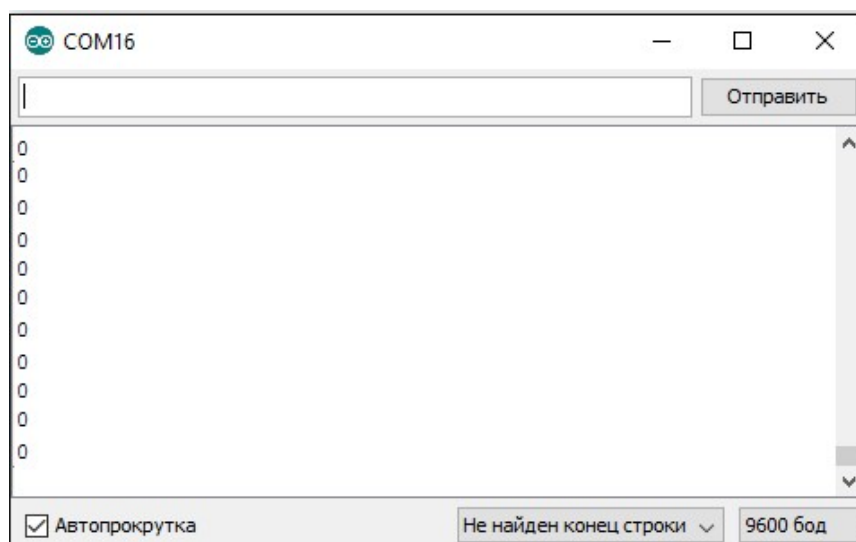
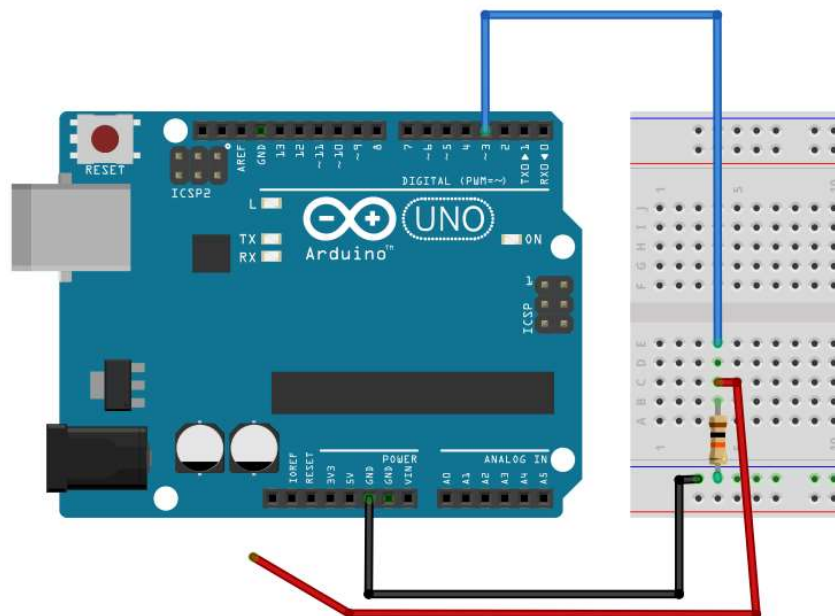
В **setup()** мы написали **pinMode(3, INPUT);**, как мы помним команда **pinMode** устанавливает режим работы порта на ввод или на вывод, когда мы включали светодиод. прописывали режим **OUTPUT** - вывод. Теперь мы наоборот хотим считывать напряжение с порта и нам нужно установить режим **INPUT**. Выходит что команда **pinMode(3, INPUT);** устанавливает 3 порт в режим считывания, мы можем подавать на третий порт 5 или 0 вольт и считывать данные.

В блоке **loop()** есть новая не знакомая нам команда - **digitalRead(3);**, в предыдущих занятиях мы использовали похожую команду **digitalWrite(3, HIGH);** если вы помните, то с помощью нее мы подаем на порт Arduino 5 вольт или 0 вольт. Функция **digitalRead();** наоборот считывает напряжение с порта. Если напряжение на порт подается то функция выдает 1, а если напряжение не подается то 0. В качестве параметра мы указываем номер порта с которого считываем напряжение.

Это означает, что если сейчас мы будем подавать на 3 порт 5 вольт, то функция **digitalRead(3)** будет выдавать 1. Соответственно строка **pin = digitalRead(3);** сохранить в переменную **pin** число 1. И если мы выведем в монитор порта переменную **Serial.println(pin);**, то увидим значение которое в ней хранится. Загружаем наш код на плату и смотрим что у нас получилось:

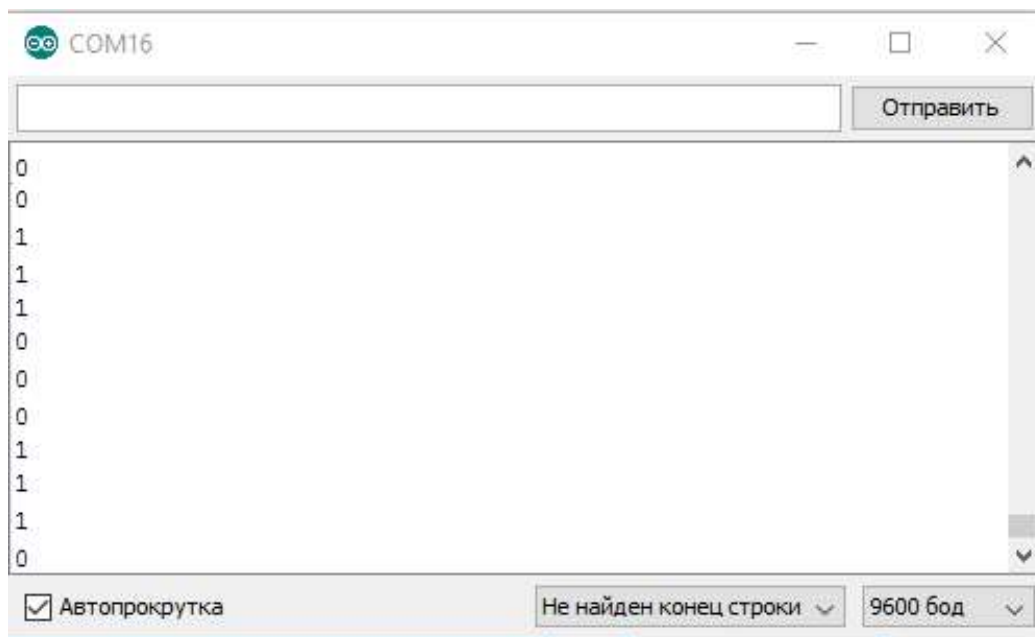


В монитор последовательного вывода выводится цифра один, потому что мы подключили 3 порт Arduino к 5V. Если мы сейчас попробуем отключить наш провод от 5 вольт, то монитор порта выведется 0:



Теперь когда на 3 порт ничего не подается, то функция **digitalRead(3)** сохраняет в переменную **pin** значение 0;

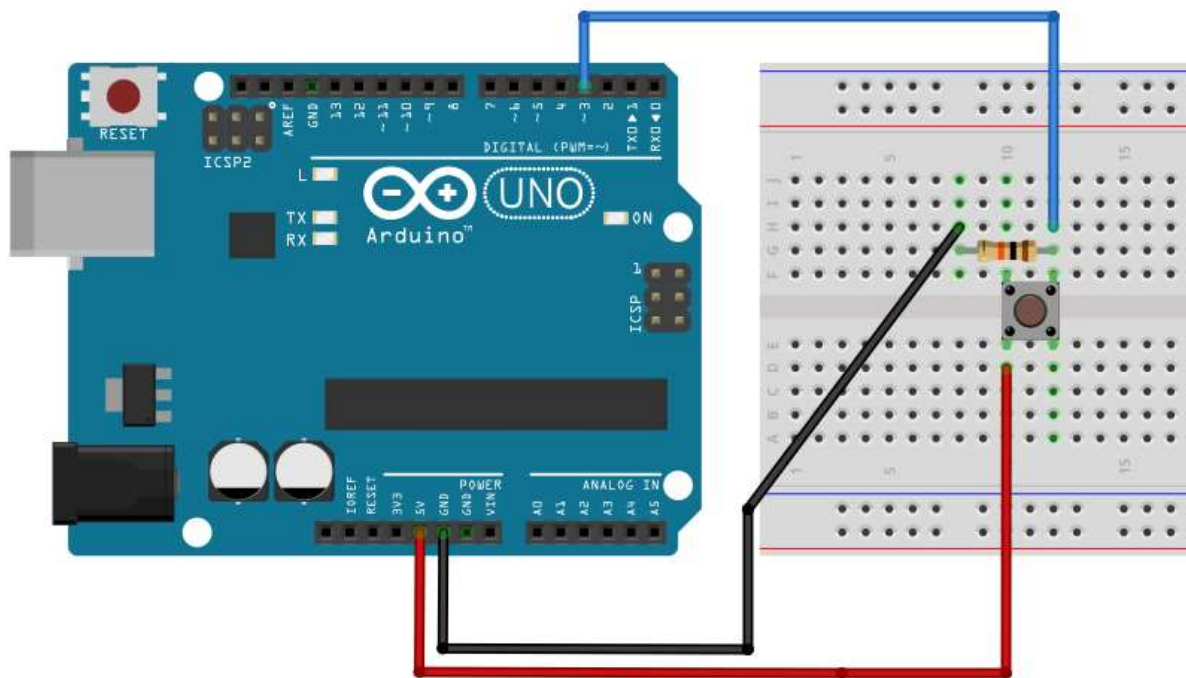
Теперь давайте вернемся к резистору, он здесь не просто так, попробуйте отключить провод от 5V, в монитор порта у нас будет подаваться ноль, а теперь попробуем вытащить резистор из макетной платы, и вы увидите как монитор последовательного порта будут хаотично подаваться единицы и нули:



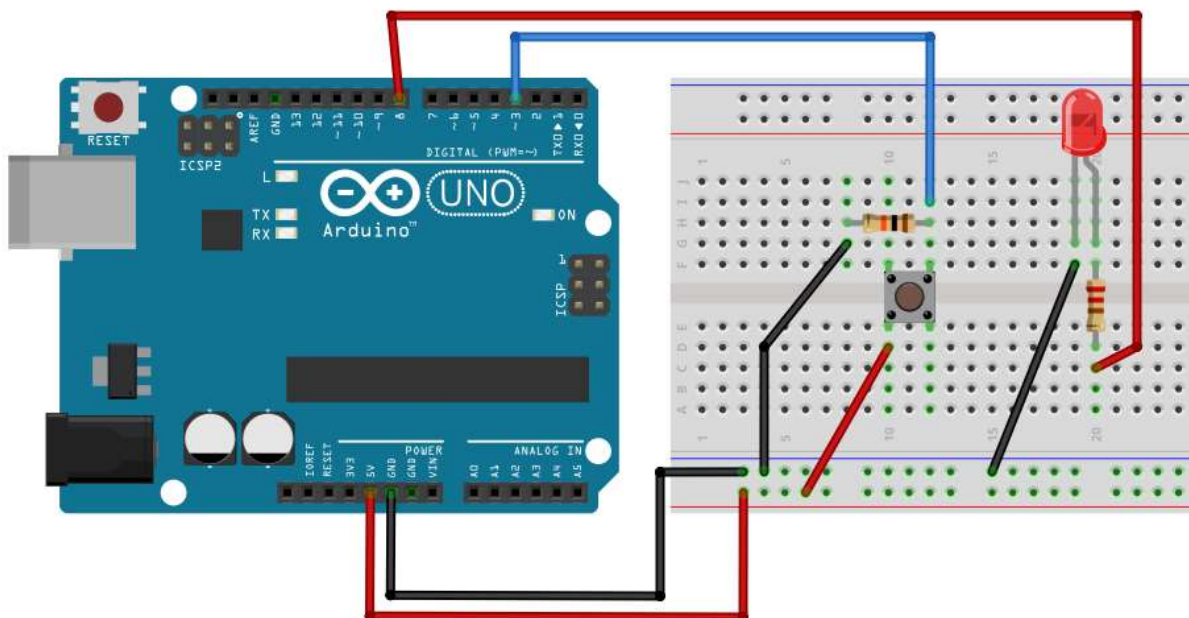
Сейчас мы не подаем 5 вольт на 3 порт arduino, но единицы все равно появляются на экране, это происходит потому что, вокруг нас много электрических помех, таких как радиоволны, наводка от электропроводки, статическое электричество, и Arduino их ловит через 3 порт как радио ловит радиоволны, и выдает все это в наш монитор, чтоб Arduino игнорировало такие помехи, необходимо обязательно подключить к нашему порту землю через резистор, тогда все электроны получаемые от помех, будут уходить через резистор. Поэтому если когда нибудь вы увидите такое хаотичное поведение порта, вспомните что необходимо подключить резистор к земле. Какое сопротивление должно быть у резистора? Сопротивление должно быть больше 5кОм. Потому что, сопротивление резистора должно превышать сопротивление порта Arduino, обычно используют резисторы номиналом 10кОм.

Такой резистор называется **стягивающий** - он стягивает электроны к минусу. Если мы подключили резистор к плюсу, то он бы назывался **подтягивающий**.

Теперь мы можем узнавать когда на цифровой порт подается питание, это позволяет подключать нам различные датчики и кнопки, давайте начнем с простого и подключим тактовую кнопку по следующей схеме.



Теперь если нажать на кнопку то цепь замыкается и у нас в мониторе порта, точно так же выводится цифра 1 - мы можем определять нажатие кнопки. Как вы могли заметить кнопка тоже подключается через резистор. Давайте сделаем чтобы светодиод включался по нажатию кнопки, для этого добавим в нашу схему светодиод и подключим его к 8 порту Arduino, обязательно подключаем светодиод через резистор 220 Ом, а кнопку через резистор 10кОм:



Теперь мы можем написать код для включения светодиода кнопкой. Мы знаем, что при нажатии на кнопку функция **digitalRead()** возвращает единицу, зная это мы можем поставить условие, что если **digitalRead()** больше единицы, то включаем светодиод, иначе - гасим светодиод:

```

int knopka;
void setup() {
  pinMode(3, INPUT); // устанавливаем 3 порт в режим считывания (ввода)
  pinMode(8, OUTPUT); // устанавливаем 8 порт в режим вывода (чтобы зажечь свет)
}

void loop() {
  knopka = digitalRead(3); //считываем нажатие кнопки с 3 порта в переменную
  if(knopka == 1){          //Если кнопка нажата
    digitalWrite(8, HIGH); //Зажигаем светодиод
  } else {                  //Если кнопка не нажата
    digitalWrite(8, LOW);  //Гасим светодиод
  }
}

```

Все строки в этом коде нам знакомы, и ничего сложно в них нет, если Вам что-то не понятно из них, то повторите предыдущие уроки. Теперь когда кнопка нажата светодиод будет гореть. Теперь попробуйте выполнить следующие задания:

1. Сделайте наоборот, чтобы когда кнопка не нажата - светодиод горел, а когда нажата был потушен.
2. Сделайте чтоб при коротком нажатии светодиод горел 3 секунды а не сразу выключался.
3. Сделайте чтобы светодиод включался не сразу после короткого нажатия на кнопку, а через 5 секунд, и горел 3 секунды.
4. Сделайте чтобы при коротком нажатии на кнопку светодиод мигал 2 раза по 1 секунде.

Давайте рассмотрим еще 1 пример и сделаем так, при коротком нажатии кнопки, светодиод включался и не выключался, для этого нам нужно будет при нажатии сохранить в переменную например цифру 1. А светодиод зажигать только тогда, когда в переменной цифра 1. При отпускании кнопки мы не будем менять значение переменной, и светодиод продолжит гореть, давайте попробуем:

```

int knopka;
void setup() {
  pinMode(3, INPUT); // устанавливаем 3 порт в режим считывания (ввода)
  pinMode(8, OUTPUT); // устанавливаем 8 порт в режим вывода (чтобы зажечь свет)
}

void loop() {
  if(digitalRead(3) == 1){ //Если кнопка сейчас нажата
    knopka = 1;           //сохраняем в нашу переменную цифру 1
  }
  if(knopka == 1){        //Если в переменной цифра 1
    digitalWrite(8, HIGH); //Зажигаем светодиод
  } else {                //Если в переменной другая цифра

```

```
digitalWrite(8, LOW); //Гасим светодиод
}
}
```

Теперь при нажатии на кнопку и ее отпускании светодиод не потухнет и продолжит гореть. Давайте теперь попробуем сделать так, чтобы при втором нажатии светодиод выключался. Как это сделать? Как определить первое сейчас нажатие или нет? Мы знаем что если мы один раз кнопку уже нажимали то в переменной `knopka` будет число 1, потому что мы его туда сохранили. Значит нужно переделать условие, если мы нажимаем кнопку и в переменной уже есть единица, то мы эту единицу меняем на 0 например и условие

```
if(knopka == 1){
    digitalWrite(8, HIGH);
}
```

не работает, а работает код, который находится в `else`

```
else {
    digitalWrite(8, LOW);
}
```

А если мы нажимаем кнопку и в переменной нет единицы, то мы включаем светодиод, давайте попробуем:

```
int knopka;
#define pin =7
void setup() {
    pinMode(3, INPUT); // устанавливаем 3 порт в режим считывания (ввода)
    pinMode(pin, OUTPUT); // устанавливаем 8 порт в режим вывода (чтобы зажечь свет)
}

void loop() {
    if(knopka != 1 && digitalRead(3) == 1){ //Если это первое нажатие

        knopka = 1; //сохраняем в нашу переменную цифру 1
        delay(1000);
    } else if(knopka == 1 && digitalRead(3) == 1){ //Если это второе нажатие

        knopka = 0; //сохраняем в нашу переменную цифру 0
        delay(1000);
    }
    if(knopka == 1){ //Если в переменной цифра 1
        digitalWrite(pin, HIGH); //Зажигаем светодиод
    } else { //Если в переменной другая цифра
        digitalWrite(8, LOW); //Гасим светодиод
    }
}
```

Теперь у нас светодиод включается и выключается при кратковременном нажатии, у нас получился настоящий выключатель света, кнопка которая запоминает действие называется кнопка с фиксацией, у нас получилась кнопка с фиксацией, потому что при коротком нажатии светодиод остается всегда включенным и при втором коротком нажатии остается всегда выключенным. Условие `if(knopka != 1 && digitalRead(3)==1)` проверяет “Если в переменной еще нет единицы и кнопка нажата”, так как в переменной нет единицы, то означает что мы нажимаем кнопку первый раз. Потому, что после того как только мы первый раз нажимаем на кнопку, то сохраняем в переменную число 1: `knopka = 1;`. Больше это условие срабатывать не будет потому что в переменной у нас единица, а если это условие не срабатывает, то программа идет проверять второе условие `else if(knopka == 1 && digitalRead(3) == 1)`, здесь мы проверяем равно ли переменная `knopka` единице, мы знаем, что если мы 1 раз нажали на кнопку то, в переменной точно будет единица, а если не нажали то не будет. Значит это условие сработает только при втором нажатии на кнопку.

Константы

Давайте теперь отвлекаемся от кнопки на другую тему, разберем что такое константы. Мы с вами проходили переменные, и знаем что это нечто похожее на коробку, только в программе, в переменную можно положить значение, можно взять, можно поменять значение.

А что же такое константа? Константа очень похожа на переменную, в нее тоже можно сохранить значение, можно взять, но в отличии от переменной значение в константе нельзя менять, если мы 1 раз в программе положили какое-то значение в константу, то больше мы это значение изменить на другое не можем. Зачем же нужна, константа, если у нее такой большой недостаток и мы не можем изменять ее значение второй раз?

Дело в том, что константа не занимает оперативную. память в Arduino. Если нам нужно всего лишь 1 раз сохранить значение в переменную, то лучше сохранить его в константу. Как же выглядят эти константы и как их использовать?

Есть 2 способа создать константу:

```
const int BOX = 2;
```

и

```
#define BOX 2;
```

обе команды делают одно и то же, они создают константу с именем `BOX` и сохраняют в нее значение 2. После этого мы уже не можем изменить это значение. Если вы заметили то когда мы объявляем константу через **const**, то необходимо указывать тип переменной **int**, **float** и т.д. Если мы объявляем константу с помощью **#define**, то тип переменной указывать не нужно. Вы можете использовать любой из этих вариантов.

Название у констант как и у переменных можно задавать любые, можно использовать латинские буквы, цифры и знак подчеркивания, цифры не должны идти первые.

Но обычно при названии констант использую только заглавные (большие) буквы, что в коде было понятно где у нас константа, а где переменная, чтобы не путаться в какой из них можно изменить значение, а в какой нельзя.

У Arduino есть свои константы, если вы заметили, например **HIGH** и **LOW**, в константе **HIGH** хранится число 1, а в константе **LOW** - число 0. Например если мы напишем

```
digitalWrite(8, HIGH);
```

и напишем

```
digitalWrite(8, 1);
```

то эти 2 команды сделают одно и то же, они подадут высокое напряжение на порт 8. Можете попробовать изменить в предыдущем примере все константы HIGH на 1, а константы LOW на 0 и проверить, будет ли код работать точно так же как и раньше.

Для чего могут использоваться константы, если их можно установить только 1 раз? Например в предыдущем примере мы часто используем порт 8 для включения светодиода, если мы захотим подключить светодиод к 9 порту, то нужно будет изменять код во всех местах, а если мы будем использовать константу в которую сохраним число 8, то код нужно будет изменить только 1 раз, давайте попробуем изменить наш код и использовать константу.

```
int knopka;
#define PIN 8;
void setup() {
  pinMode(3, INPUT); // устанавливаем 3 порт в режим считывания (ввода)
  pinMode(PIN, OUTPUT); // устанавливаем 8 порт в режим вывода (чтобы зажечь свет)
}

void loop() {
  if(knopka != 1 && digitalRead(3) == 1){ //Если это первое нажатие

    knopka = 1; //сохраняем в нашу переменную цифру 1
    delay(1000);
  } else if(knopka == 1 && digitalRead(3) == 1){ //Если это второе нажатие

    knopka = 0; //сохраняем в нашу переменную цифру 0
    delay(1000);
  }
  if(knopka == 1){ //Если в переменной цифра 1
    digitalWrite(PIN, HIGH); //Зажигаем светодиод
  } else { //Если в переменной другая цифра
    digitalWrite(PIN, LOW); //Гасим светодиод
  }
}
```

Мы создали константу PIN, сохранили в нее значение 8. И везде где нужно использовать 8 порт, вместо цифры 8 мы пишем константу PIN. Теперь если мы захотим быстро подключить светодиод в другой порт, то просто изменим код **#define PIN 8;** на код **#define PIN 9;** и везде где используется константа PIN будет выводиться число 9, а не 8, это намного быстрее, чем если бы мы везде меняли число 8 на 9.

Булевы переменные

На константах мы закончили, давайте теперь рассмотрим булевы переменные.

На 3 уроке мы изучали разные типы переменных: **int**, **float**, **long**, **unsigned long** и **unsigned int**. В Arduino, а точнее в языке C++ (потому, что Arduino программируется именно на этом языке) есть еще один тип данных, и называется он **boolean** - булевый тип данных. Переменные этого

типа создаются с помощью слова **bool**. Переменные этого типа могут хранить только 2 значения **true** и **false** (истина и ложь).

Например:

```
bool box;  
box = true;  
box = false;
```

true и **false** - по сути те же самые константы, в **true** хранится 1, а в **false** хранится 0; Поэтому если мы напишем код:

```
bool box;  
box = 1;  
box = 0;
```

То сделаем то же самое, что и выше. Зачем нужны переменные в которые можно сохранить только 2 значения, когда у нас есть например переменная **int** в которую можно сохранить аж целых 65536 разных значений.

Дело в том, что переменные типа **int** занимают в оперативной памяти 2 байта, а переменные типа **bool** - 1 байт, поэтому используя переменные **bool** там где это возможно, вы экономите оперативную память.

Где можно использовать переменные **bool**? Например в опять в нашем примере мы сохраняли в переменную **korobka** только 2 значения 1 и 0. Мы могли использовать вместо переменной **int** переменную **bool** и сэкономили бы немного памяти. Давайте переделаем наш код на использование переменной **bool**:

```
bool knopka;  
#define PIN 8;  
void setup() {  
  pinMode(3, INPUT); // устанавливаем 3 порт в режим считывания (ввода)  
  pinMode(PIN, OUTPUT); // устанавливаем 8 порт в режим вывода (чтобы зажечь свет)  
}  
  
void loop() {  
  if(knopka != 1 && digitalRead(3) == 1){ //Если это первое нажатие  
  
    knopka = 1; //сохраняем в нашу переменную цифру 1  
    delay(1000);  
  } else if(knopka == 1 && digitalRead(3) == 1){ //Если это второе нажатие  
  
    knopka = 0; //сохраняем в нашу переменную цифру 0  
    delay(1000);  
  }  
  if(knopka == 1){ //Если в переменной цифра 1  
    digitalWrite(PIN, HIGH); //Зажигаем светодиод  
  } else { //Если в переменной другая цифра  
    digitalWrite(PIN, LOW); //Гасим светодиод  
  }  
}
```

```
}  
}
```

Здесь мы изменили тип переменной `korobka`, если запустить программу то ничего не изменится, но свободной оперативной памяти стало больше. Теперь давайте во всех местах где в переменную `korobka` сохраняются или считываются значения заменим 1 и 0 на **true** и **false**.

```
bool knopka;  
#define PIN 8;  
void setup() {  
  pinMode(3, INPUT); // устанавливаем 3 порт в режим считывания (ввода)  
  pinMode(PIN, OUTPUT); // устанавливаем 8 порт в режим вывода (чтобы зажечь свет)  
}  
  
void loop() {  
  if(knopka != true && digitalRead(3) == 1){ //Если это первое нажатие  
  
    knopka = true ; //сохраняем в нашу переменную цифру 1  
    delay(1000);  
  } else if(knopka == true && digitalRead(3) == 1){ //Если это второе нажатие  
  
    knopka = false;  
    //сохраняем в нашу переменную цифру 0  
    delay(1000);  
  }  
  if(knopka == true){ //Если в переменной цифра 1  
    digitalWrite(PIN, HIGH); //Зажигаем светодиод  
  } else { //Если в переменной другая цифра  
    digitalWrite(PIN, LOW); //Гасим светодиод  
  }  
}
```

Если запустить программу, то опять же мы увидим, что ничего не поменялось, поэтому не важно что вы сохраняете в переменную типа `bool`, можно сохранять 1 или 0, а можно сохранять `true` или `false`, здесь нет никакой разницы, просто если в будущем вы в коде увидите `true` или `false`, то знайте, что это тоже самое, что и 1 или 0.

Урок №5

Тема: Работа с аналоговыми портами, считывание данных с датчика света

Наконец-то мы подобрались к работе с датчиками, сегодня мы будем подключать фоторезистор для определения уровня освещенности. Но для этого нам необходимо научиться считывать данные с аналоговых портов на Arduino.

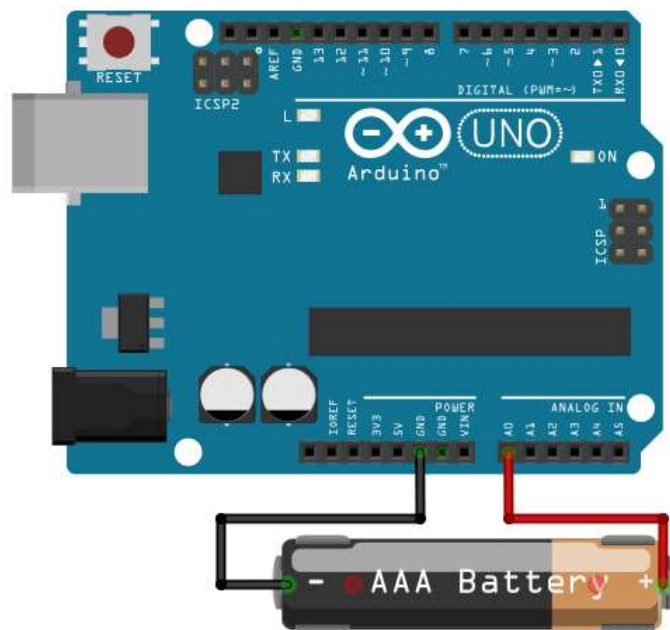
На первом уроке мы говорили, что аналоговые порты на Arduino могут считывать уровень напряжения от 0 - 5 в более точном диапазоне чем цифровые.

Чтобы считать уровень напряжения которое подается на аналоговый порт в Arduino существует функция **analogRead()**; В параметрах функции мы пишем номер аналогового порта, от A0 до A5 (на Arduino UNO). Эта функция возвращает числовое значение от 0 до 1023, в зависимости от того, сколько вольт подается на порт. Если 0 вольт, то функция вернет 0, если 5 вольт, то функция вернет 1023, все остальные числа она возвращает когда напряжение находится в промежутке между 0 и 5 вольт.

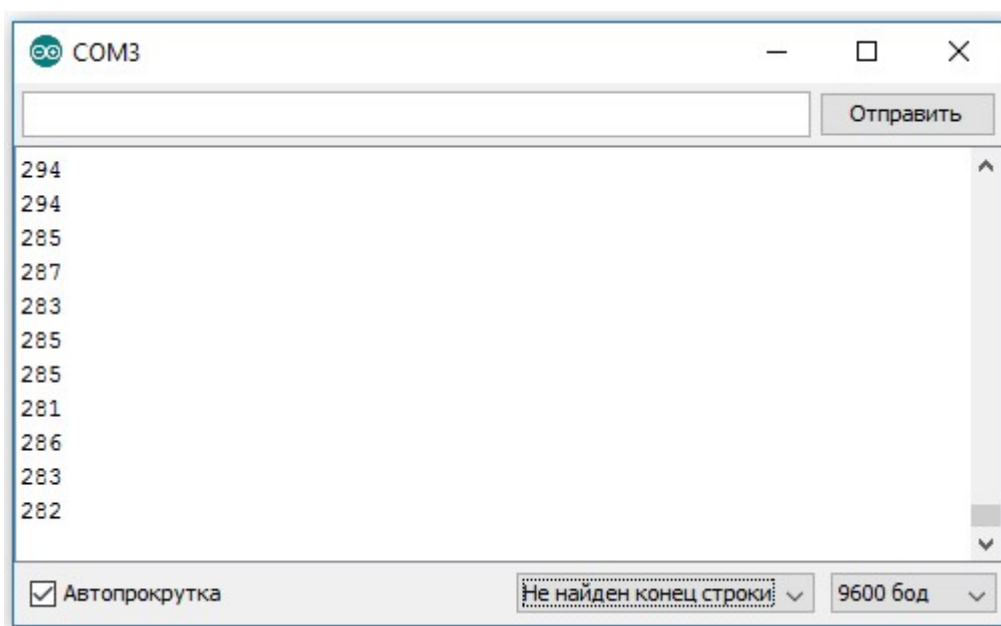
Давайте напишем следующий код, который будет считывать напряжение с порта A0 и выводить в монитор последовательного порта:

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(analogRead(A0));  
  delay(100);  
}
```

Теперь загрузим прошивку на Arduino и подключим к ней батарейку 1.5 вольт, плюс от батарейки вы подключите в аналоговый порт A0, а минус от батарейки объедините с минусом Arduino.



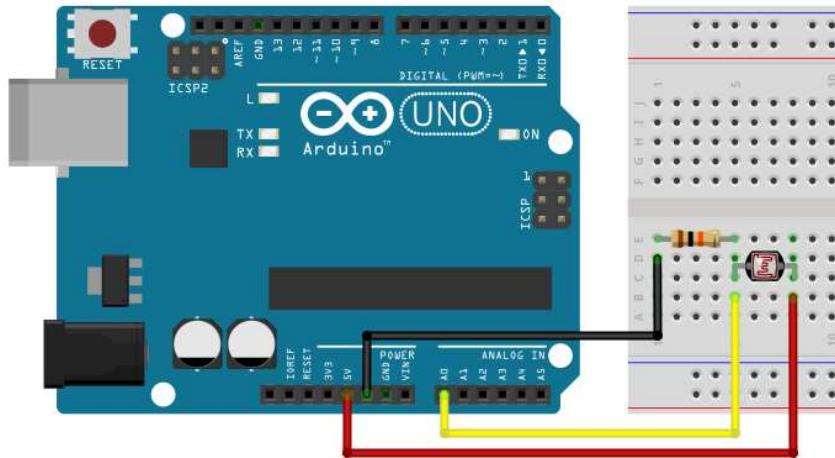
Теперь если открыть монитор последовательного порта, тогда мы увидим какое значение подается на аналоговый порт:



У вас может быть другое значение в зависимости от того на сколько заряжена ваша батарейка. Теперь если оставить так программу на долгое время мы увидим как значение постепенно уменьшается, это означает что батарейка садится, поэтому напряжение на ней падает поэтому функция **analogRead()** выдает значение меньше чем раньше.

Давайте подключим фоторезистор к Arduino. Фоторезистор необходим для того, чтобы определять насколько светло сейчас. Фоторезистор очень похож на обычный резистор, он как резистор сопротивляется току, но в отличие от простого резистора, он меняет свое сопротивление в зависимости от того, сколько света попадает на него, чем больше света попадает на фоторезистор, тем меньше его сопротивление и если например мы подключим его к +5V на Arduino, то на выходе резистора будет не 5 вольт а меньше и чем темнее будет вокруг и меньше света, на него будет попадать, тем меньше будет напряжение на выходе, а с

помощью функции `analogRead()` мы сможем считать эти данные и определить степень освещения в комнате. Давайте подключим его к нашей Arduino, например к аналоговому порту A0:



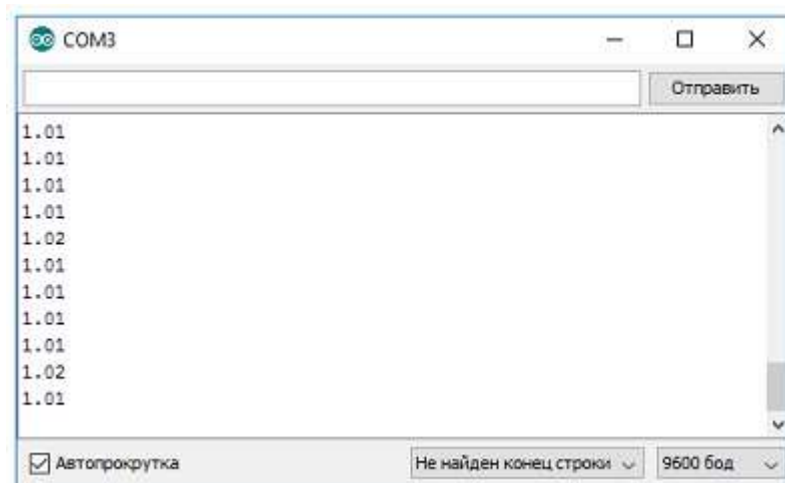
Теперь если с той же самой прошивкой мы откроем монитор последовательного порта, то увидим, что у нас точно так же как и с батарейкой выводятся числа в диапазоне от 0 до 1023, если мы попробуем прикрыть фоторезистор чем нибудь, чтобы на него не попадал свет, то увидим, что значение выводимое в монитор порта уменьшилось, это потому что фоторезистор увеличил свое сопротивление, и напряжение подаваемое на порт снизилось. Если же наоборот мы посветим чем нибудь ярким на фоторезистор, то выводимое значение увеличится, потому, что сопротивление фоторезистора уменьшилось и напряжение подаваемое на порт выросло.

Кстати говоря мы можем посчитать сколько именно вольт подается на порт.

Мы знаем что если подать на порт 5 вольт, **`analogRead()`** выдаст нам число 1023. Зная это мы можем посчитать какая цифра соответствует одному вольту, для этого разделим $1023 / 5$ и получим 204.6. Теперь если в коде выводить на экран не, напрямую считанное число, а разделенное на 204.6, то мы получим напряжение подаваемое на порт, например:

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(analogRead(A0)/204.6);  
  delay(100);  
}
```

Теперь на порт у нас будет выводиться напряжение, от 0 до 5 вольт, в зависимости от того, как сейчас темно, обычно значение никогда не достигает ровно 5 вольт или ровно 0 вольт, должно быть очень темно или очень светло чтобы это произошло.



Теперь если подключить обратно нашу батарейку, то мы увидим какое на ней напряжение. Есть еще один способ подсчитать напряжение подаваемое на порт, для таких целей в Arduino существует функция **map()**, если мы вызовем функцию **map()** с такими параметрами: **map(analogRead(A0), 0, 1023, 0, 5)**, то она автоматически преобразует получаемые с порта значения в число от 0 до 5. Мы видим что у функции есть 5 параметров, первый параметр, то число которое нам необходимо преобразовать, второй параметр это минимально возможное значение этого числа, для нас это 0, потому что функция **analogRead()** вернет 0, если на порт ничего не подать. Второй параметр функции это максимально возможное значение для число которое мы будем преобразовывать, у нас это 1023. Третий параметр это минимально возможно значение числа в которое мы хотим преобразовать, то есть 0 вольт, и наконец четвертый параметр это максимально допустимое значение в которое мы хотим преобразовать, для нас это 5 вольт, давайте попробуем применить эту функцию:

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

void loop() {
  Serial.println(map(analogRead(A0), 0, 1023, 0, 5));
  delay(100);
}
```

Результат будет аналогичный предыдущему примеру.

Давайте теперь выполним небольшое задание:

1. Подключите к Arduino светодиод и сделайте так, чтобы он включался когда становится темно, и выключался, когда становится светло.

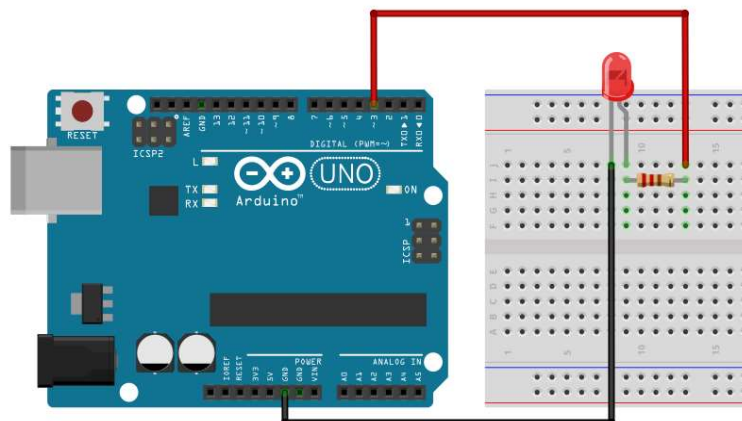
помощью функции **analogRead()** мы считывали значения с аналоговых портов, то с помощью функции **analogWrite()** мы можем плавно менять напряжение на некоторых. Сделайте так, чтобы если на порт подается напряжение меньше 4 вольт, то в монитор порта писалось предупреждение: "Low power", что будет означать, о том что у нас низкий заряд батарей.

1. Давайте добавим к схеме кнопку и сделаем так, чтобы от нее включался светодиод, но только в темное время суток.

Функция **analogWrite()**;

Теперь давайте рассмотрим функцию **analogWrite()**, как вы могли уже догадаться, то эта функция противоположна функции **analogRead()**, но это не совсем так. Если с портах Arduino.

Функция **analogWrite()** принимает значения от 0, до 255, если мы вызовем **analogWrite(3, 0)** то на порт 3 будет подано 0 вольт, а если мы вызовем **analogWrite(3, 255)**, то на порт 3 будет подано 5 вольт, передавая значения от 0 до 255 мы будем плавно изменять напряжение от 0 до 5 вольт. Давайте попробуем реализовать следующий пример, для начала подключим к 3 порту светодиод:



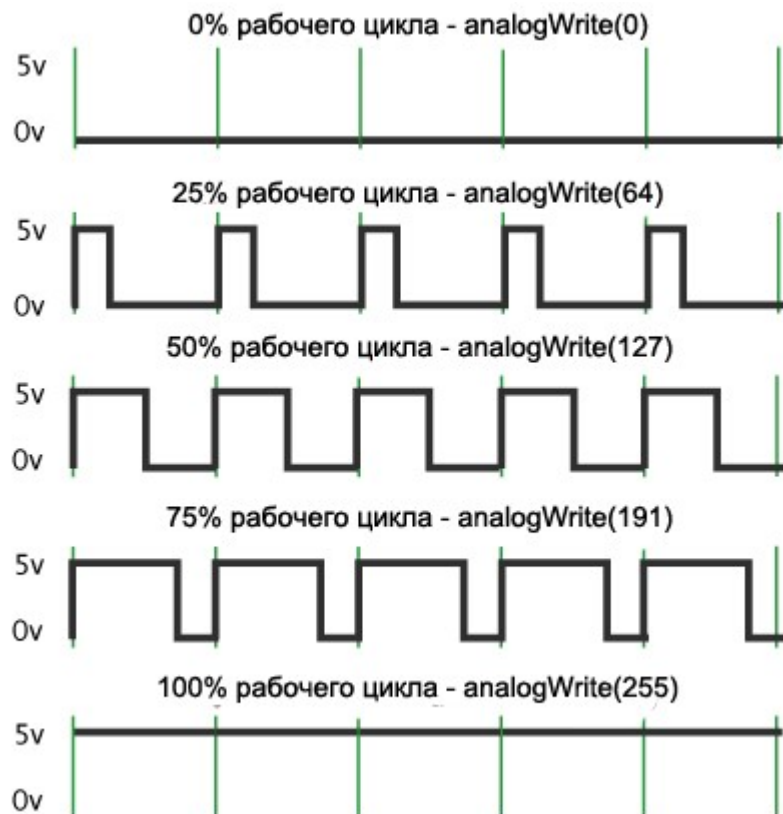
Теперь напомним следующий код и загрузим его в Arduino:

```
void setup() {  
  
}  
  
void loop() {  
  analogWrite(3, 0);  
  delay(500);  
  analogWrite(3, 127);  
  delay(500);  
  analogWrite(3, 255);  
  delay(500);  
}
```

Загрузим код вы увидите что светодиод, сначала не будет гореть, потом будет гореть тускло, потом на полную яркость.

Если вдаваться в подробности, то на самом деле, функцией **analogWrite()** мы изменяем не напряжение на 3 порту, а быстро подаем на него 5 и 0 вольт по очереди, Это происходит очень быстро, 490 раз в секунду (490 герц), когда одинаковое время подается и высокое напряжение и низкое то светодиод светит на половину, если 5 вольт подается дольше по времени, то светодиод светит ярче и так до максимума, когда мы пишем **analogWrite(3, 127)** то на порт подается 5 и 0 вольт одинаково по времени. Если мы пишем **analogWrite(3, 255)** то на порт подается только 5 вольт, если мы пишем **analogWrite(3, 0)** то на порт подается только 0 вольт. Это очень наглядно можно увидеть смотря на изображение взятое из

<http://arduino.ru/Tutorial/PWM>



На этом графике видно как при разных значениях функции **analogWrite()** у нас изменяется продолжительность высокого и низкого сигнала. Поэтому чем длиннее по времени промежуток высокого напряжения (5 вольт) тем ярче горит светодиода, а чем короче этот промежуток, тем тусклее, вплоть до полного выключения - **analogWrite(3,0)**. Но так управлять яркостью мы можем не на всех портах, а на специальных, ШИМ портах, они помечены на Arduino знаком волнистой линии “~”. Что означает ШИМ? ШИМ - это широтно импульсная модуляция. Если быстро по-очереди подается низкое и высокое напряжение, это называется импульсный ток, потому что подается короткий сигнал (импульс) высокого напряжения, а потом низкого.

По английски ШИМ переводится как PWM (pulse-width modulation). На Arduino обычно есть такая надпись:



Теперь мы можем сделать вывод что у функций **analogWrite()** и **analogRead()** нет ничего общего, **analogWrite()** даже не работает с аналоговыми портами.

Давайте теперь выполним несколько заданий:

1. *Сделайте так чтобы у светодиода плавно увеличивалась яркость и как только она дойдет до максимального значения, то уменьшалась. В общем необходимо чтобы светодиод плавно мигал.*
2. *Подключите к Arduino кнопку с делайте так, чтобы при нажатии у светодиода менялась яркость свечения, всего должно быть 3 уровня яркости: светодиод выключен, светодиод горит в половину яркости и светодиод горит на полную яркость.*

Урок №6

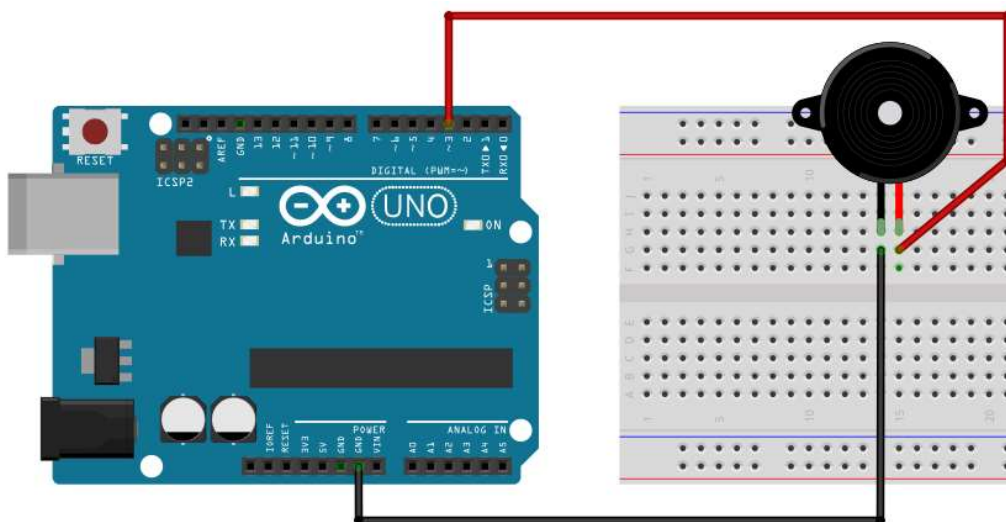
Тема: Работа с аналоговыми портами, подключение тонального динамика, работа с датчиком температуры

На прошлом уроке мы познакомились с аналоговыми портами и широтно-импульсной модуляцией (ШИМ), на этом уроке мы продолжим выполнять практические занятия на эту тему, мы подключим к Arduino тональный динамик и датчик температуры, рассмотрим как они работают.

Тональный динамик или по другому его еще называют пьезо-пищалкой, это динамик который воспроизводит звуки на частотах 2-4кГц, работает он от 4-8 вольт. Наверное вы уже сталкивались в жизни с такими динамиками например в музыкальных открытках, в популярная игра 00-х годов - тетрис, в каких нибудь китайских игрушках издающих звуки. Выглядят они по разному, но чаще всего встречаются такие:



Сегодня мы будем работать с первым вариантом. Тональный динамик начинает издавать звук, когда на него подается импульсный ток, на прошлом уроке мы узнали, что функция `analogWrite()` может подавать импульсный ток через ШИМ (PWM) порты, в параметрах задаются значения от 0 до 255, чем выше значение тем длиннее по времени импульс высокого напряжения и короче по времени импульс низкого напряжения. Это означает что если мы подключим пищалку к любому ШИМ порту, например к 3, и напишем команду `analogWrite(0)`, то динамик пищать не будет совсем, а если напишем `analogWrite(255)`, то динамик будет пищать максимально громко. Давайте подключим динамик к Arduino, наш динамик имеет 2 ножки короткую и длинную, они подключаются соответственно к + и - . Мы подключим плюс к 3 цифровому порту, а минус к GND.



Теперь напишем такой код:

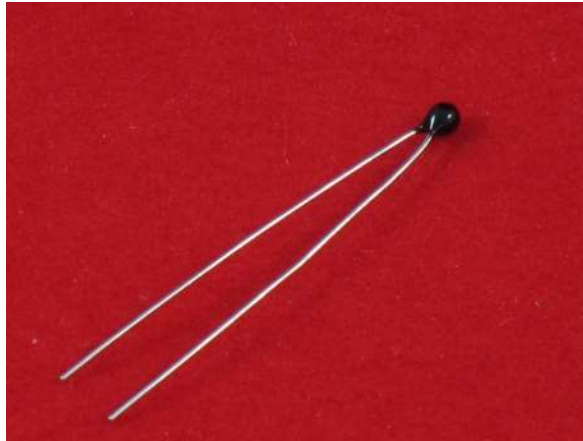
```
void setup() {  
  
}  
  
void loop() {  
  analogWrite(3, 0);  
  delay(500);  
  analogWrite(3, 100);  
  delay(500);  
  analogWrite(3, 150);  
  delay(500);  
  analogWrite(3, 200);  
  delay(500);  
  analogWrite(3, 255);  
  delay(500);  
}
```

Если вы помните, то похожую программу мы использовали на прошлом уроке, чтобы светодиод у нас с каждым разом горел все ярче и ярче. Этот же код будет заставлять пищать наш динамик всё громче и громче. Как видим, издавать звуки с помощью Arduino не сложно, давайте выполним несколько заданий:

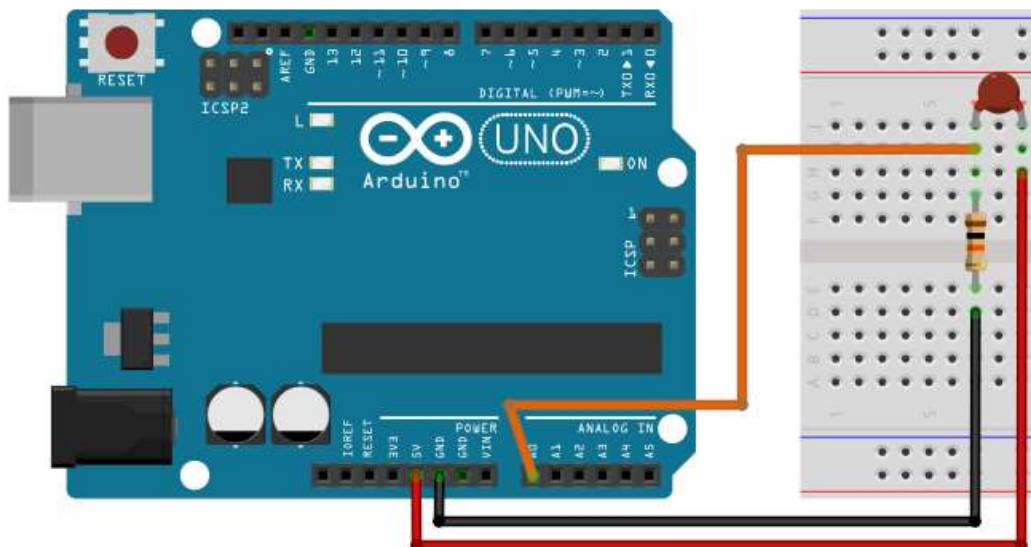
1. Вспомним пример прошлого урока и сделаем так чтобы динамик плавно пищал все громче и громче, доходил до максимальной громкости а потом плавно начинал пищать все тише и тише и так далее. Делаем это с помощью функции `analogWrite()`
2. Давайте сделаем терменвокс, терменвокс это музыкальный инструмент который издает звуки в зависимости от положения руки над ним <https://ru.wikipedia.org/wiki/Терменвокс> , мы это сделаем с помощью датчика освещения, сделайте так, что чем больше попадает света на датчик освещения тем громче звучит динамик и чем меньше света попадает на датчик освещения тем тише звучит динамик, это можно сделать с помощью функции `map()`, `analogRead()` и `analogWrite()`.
3. Давайте сделаем свой синтезатор, подключите к Arduino 3 кнопки и сделайте чтобы при нажатии на каждую из них издавался разный звук.

Датчик температуры, термистор

Теперь давайте познакомимся с термистором. на прошлом уроке мы изучали фоторезистор, он измерял свое сопротивление в зависимости от яркости света, который на него попадает. Термистор работает точно так же в отношении температуры - он меняет свое сопротивление, причем чем выше температура тем ниже его сопротивление, а чем ниже температура, тем выше сопротивление. Выглядит термистор в виде небольшой черной застывшей капельки от которой отходят 2 ножки, у термистора как и резисторов нет плюса или минуса. Внутри этой черной капельки находится маленький кусочек вещества, которое при изменении температуры изменяет свое сопротивление. Мы будем использовать термистор MF52, его сопротивление меняется от 0 до 10кОм. Он изображен на фото ниже:



Давайте теперь подключим его к Arduino, схема подключения у него такая же как и датчика освещенности, одна ножка подключается к плюсу, вторая подключается к аналоговому порту и подтягивается к земле резистором 10кОм.

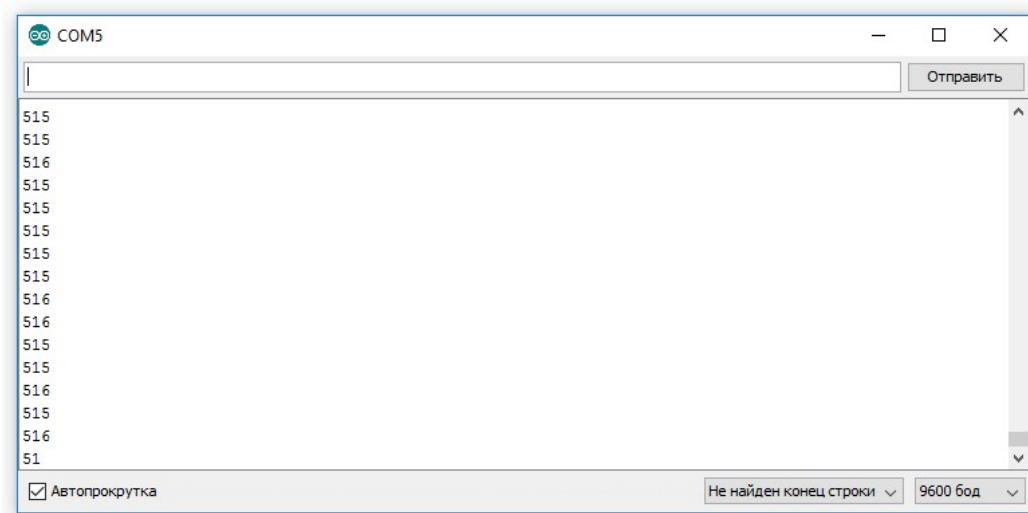


Давайте теперь выведем на монитор последовательного порта, значение которое нас на выводится с аналогового порта:

```
int temp;
void setup() {
  Serial.begin(9600);
}

void loop() {
  temp = analogRead(A0);
  Serial.println(temp);
}
```

Теперь если загрузить наш код и открыть монитор последовательного порта, то мы увидим, как у нас в зависимости от температуры меняются значения считываемые с порта



Теперь давайте коснемся пальцем руки до термистора и увидим как на порту стало расти напряжение, но коснитесь именно до черной капельки, контактов касаться нельзя иначе измерения будут не точными. Можете попробовать налить в стакан горячей воды и поднести поближе к термистору, может даже коснуться.

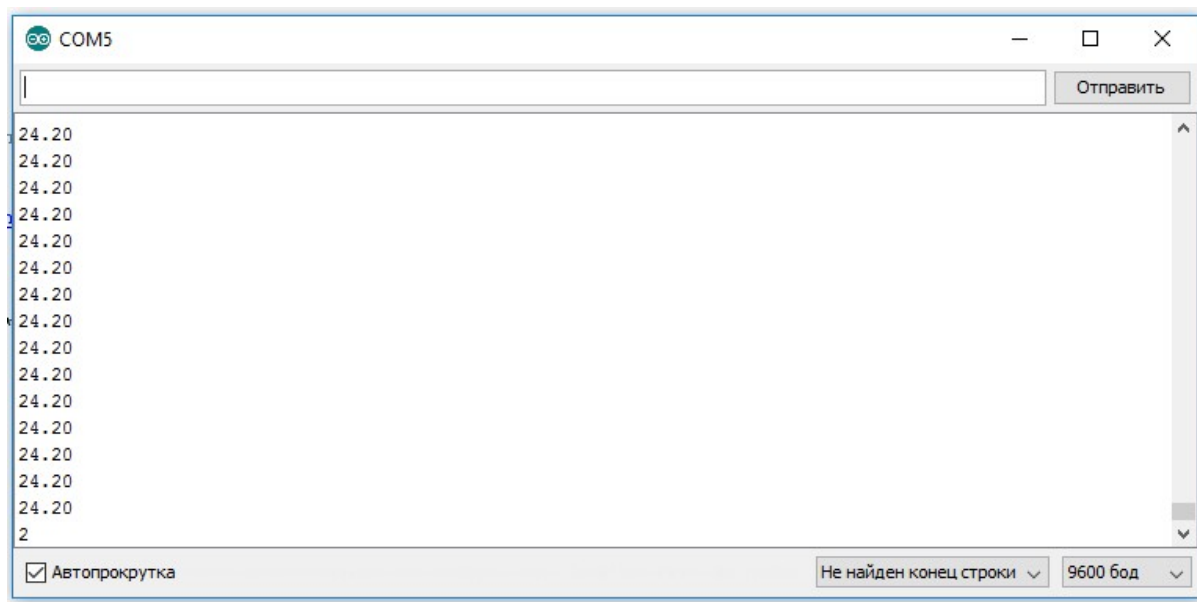
На википедии есть формула расчета температуры данного термистора в зависимости от сопротивления посмотреть ее можно здесь <http://en.wikipedia.org/wiki/Thermistor> , мы не будем вдаваться в ее подробности, иначе это может занять весь урок, потому что урок рассчитан и на читателей которые возможно еще не знакомы с логарифмами и возведением в степень. Поэтому мы написали сразу готовую функцию которая по этой формуле рассчитывает температуру нашего термистора и назвали ее getTemp(), не вдавайтесь в подробности ее содержимого, просто перепишите ее и запустите программу. Эта функция получает значение с аналогового порта Arduino и сразу выдает температуру.

```
float temp;
void setup() {
  Serial.begin(9600);
}

void loop() {
  temp = getTemp(analogRead(A0)); //Рассчитываем температуру
  Serial.println(temp);           //Выводим температуру на экран
}

// Функция написана по формуле http://en.wikipedia.org/wiki/Thermistor
double getTemp(int analogVal) {
  double t;
  t = log(((10240000/analogVal) - 10000));
  t = 1 / (0.001129148 + (0.000234125 * t) + (0.0000000876741 * t * t * t));
  t = t - 273.15; // Конвертируем Кельвины в Цельсии
  return t;
}
```

Теперь загрузим программу, запустим монитор последовательного порта и увидим, что у нас сразу отображается текущая температура:



Функция `getTemp(analogRead(A0))`, считывает значение с аналогового порта, преобразует его в температуру в градусах Цельсия и сохраняет в переменную `temp`, далее мы просто выводим эту переменную на последовательный порт. Попробуйте теперь коснуться пальцем до датчика, вы увидите как быстро изменяется его температура. Эта температура сейчас не совсем точная, потому что мы подключаем термистор через макетную плату и провода, они тоже имеют свое сопротивление, для получения более точных значений необходимо было бы припаять термистор прямо к Arduino.

Ничего сложного в работе с термистором нет, теперь давайте выполним несколько заданий:

1. *Сделайте так чтобы при превышении температуры в 30 градусов Цельсия, начинал пищать динамик, а при понижении прекращал.*
2. *Добавьте кнопку, и сделайте так чтобы при нажатии на кнопку в какую нибудь переменную сохранялась температура, и эта переменная выводилась на монитор последовательного порта*

Урок №7

Тема: Знакомство с массивами данных, циклы, перебор массивов, работа с потенциометром

Сегодня у нас будет довольно сложная тема, мы будем изучать массивы. Что же такое массивы? Простыми словами это куча переменных объединенных в одну переменную. Например, представим что нам нужно создать 10 переменных и потом сохранить в них числа. Тогда бы мы писали это так:

```
int box1;  
int box2;  
int box3;  
int box4;  
int box5;  
int box6;  
int box7;  
int box8;  
int box9;  
int box10;
```

Согласитесь, что это всё занимает уйму места и времени, а если мы захотим сохранить в каждую переменную какое-то число, то нам будет необходимо перечислять каждую переменную:

```
box1 = 5;  
box2 = 7;  
box3 = 6;  
box4 = 1;  
box5 = 4;  
box6 = 80;  
box7 = 5;  
box8 = 21;  
box9 = 40;  
box10 = 0;
```

С массивами все намного, проще мы можем создать массив переменных простой командой, например если мы хотим создать массив из 10 переменных, то нам достаточно написать так:

```
int box[10];
```

Теперь у нас есть массив из 10 переменных, или по другому это называется массив состоит из 10 элементов, в каждую переменную (элемент) мы как и в предыдущем примере можем сохранить значения:

```
box[0] = 5;
box[1] = 7;
box[2] = 6;
box[3] = 1;
box[4] = 4;
box[5] = 80;
box[6] = 5;
box[7] = 21;
box[8] = 40;
box[9] = 0;
```

То-есть в квадратных скобках мы указываем номер нашей переменной, хотим напомнить, что в программировании нумерация начинается с нуля, поэтому когда мы хотим сохранить значение в первый элемент массива box, то мы пишем `box[0] = 5`.

Но так сохранять данные тоже не удобно, это занимает много места, массивы имеют еще один большой плюс, мы можем сохранить все значения сразу при создании массива, вот как это будет выглядеть:

```
int box[10] = {5,7,6,1,4,80,5,21,40,0};
```

Мы ставим пустые квадратные скобки не указывая номер элемента, и присваиваем массиву все нужные числа, через запятую заключенные в фигурные скобки. Теперь наш код занимает гораздо меньше места, 2 строчки вместо 20.

При этом мы как и раньше можем обращаться к каждому элементу массива по отдельности, например если мы напишем код:

```
int box[10] = {5,7,6,1,4,80,5,21,40,0}; //Создаем массив размером в 10 элементов
//И сразу сохраняем в него элементы

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

void loop() {
  Serial.println(box[0]); //Выводим на экран нулевой элемент массива, т.е. число 5
}
```

В итоге на экран у нас будет выводиться число 5, так как оно является нулевым элементом массива.

Мы можем так же не указывать явно номер элемента, мы можем сохранить номер элемента в переменную и потом вместо номера элемента подставлять переменную, например если мы сохраним переменную `int i=0`; и потом выведем на экран `box[i]`, то на экран так же выведется число 5. Мы можем по очереди вывести все элемента массива увеличивая переменную, например:

```
int box[10] = {5,7,6,1,4,80,5,21,40,0}; //Создаем массив размером в 10 элементов
//И сразу сохраняем в него элемент

int i=0;
void setup() {
```

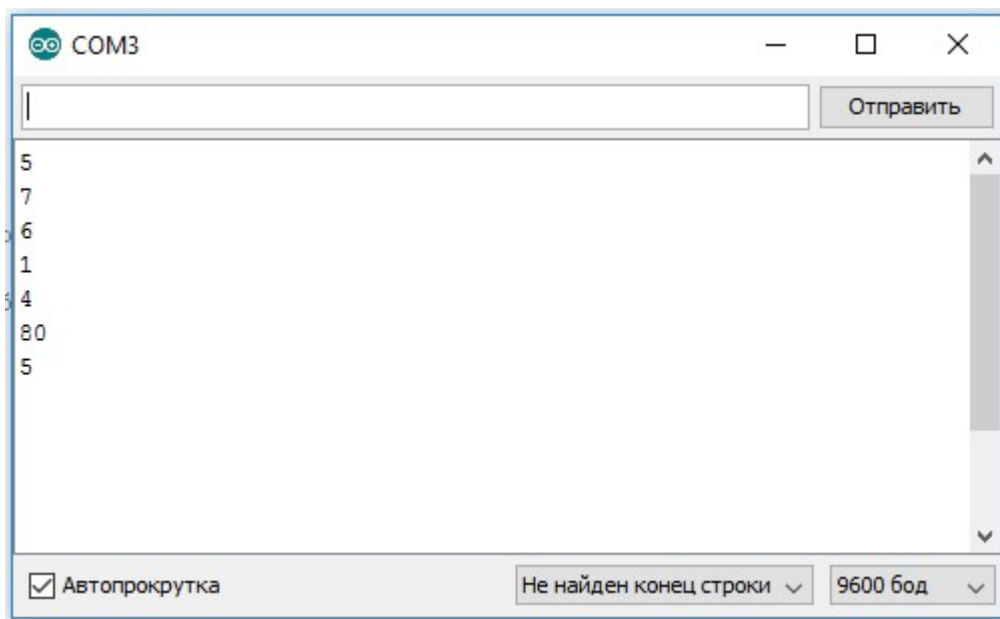
```

Serial.begin(9600);
}

void loop() {
  Serial.println(box[i]); //Выводим на экран нулевой элемент массива, т.е. число 5
  i++; //Увеличиваем переменную i на 1
  if(i==10){ //Если i равна 10, то обнуляем ее, чтобы не превысить размер массива
    i=0;
  }
  delay(1000);
}

```

Теперь на экран будут выводиться все элементы массива по очереди:



Обратите внимание, что мы обнуляем переменную *i*, когда она превышает 9, потому что у нас нет десятого элемента в массиве и если мы попытаемся получить его значение, то будут выдаваться ошибочные данные, наш массив имеет всего 10 элементов, с 0 по 9, и мы не можем сохранять в него больше 10 чисел, для этого необходимо увеличить при создании размер массива.

Вместо того, чтобы перебирать все элементы массива и выводить их на экран мы можем выводить их на тональный динамик например, если пропишем уровни звучаний по очереди, то сможем сделать свою мелодию. Или можем мигать светодиодом в определенной последовательности.

Для закрепления выполните задания:

1. Сделайте так, чтобы динамик воспроизводил какуюнибудь мелодию, например сигнал SOS <https://ru.wikipedia.org/wiki/SOS>, он звучит как 3 коротких сигнала, 3 длинных сигнала и опять 3 коротких сигнала. Для этого используйте массив в котором будете указывать тональности динамика от 0 до 255 и еще один массив в котором будете указывать длину звучания в миллисекундах, с помощью команды `delay()` задавайте длину звучания, пауза между звуками одинаковая
2. Сделайте 3 режима мигания светодиода, например: SOS, нормальный и быстрый и

меняйте режимы по нажатию кнопки. Для этого используйте 3 массива в которых задавайте паузы для светодиодов, и при нажатии на кнопку меняйте массив.

Для примера внизу приведен код, который воспроизводит жалкое подобие вступления песни “КаЛинка-малинка”

```
int box[11] = {150,125, 75,100,125, 75,100,125, 100,75,50}; //Тональности динамика
int pause[11] = {200,300, 100,100,200, 100,100,200, 100,100,100}; //Длины звуков
int pause2[11] = {200,100, 150,150,200, 150,150,200, 150,150,1000}; //Паузы
int i=0; //Здесь мы будем хранить номер текущей тональности
void setup() {

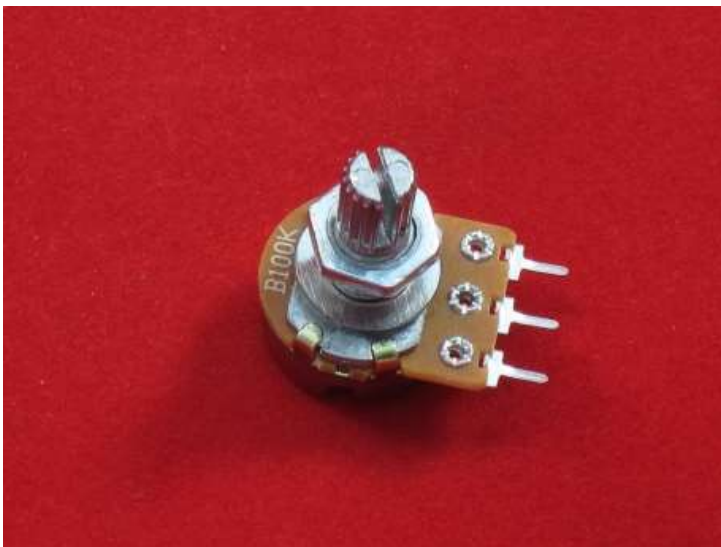
}

void loop() {

  analogWrite(3,box[i]); //Воспроизводим звук из массива
  delay(pause[i]);       //Воспроизводим звук с длиной из массива pause
  analogWrite(3,0);
  delay(pause2[i]);      //Делаем паузу указанную в массиве pause2

  i++; //Увеличиваем переменную i на 1
  if(i==11){ //Если переменная i равна 11, то обнуляем переменную
    i=0;
  }
}
```

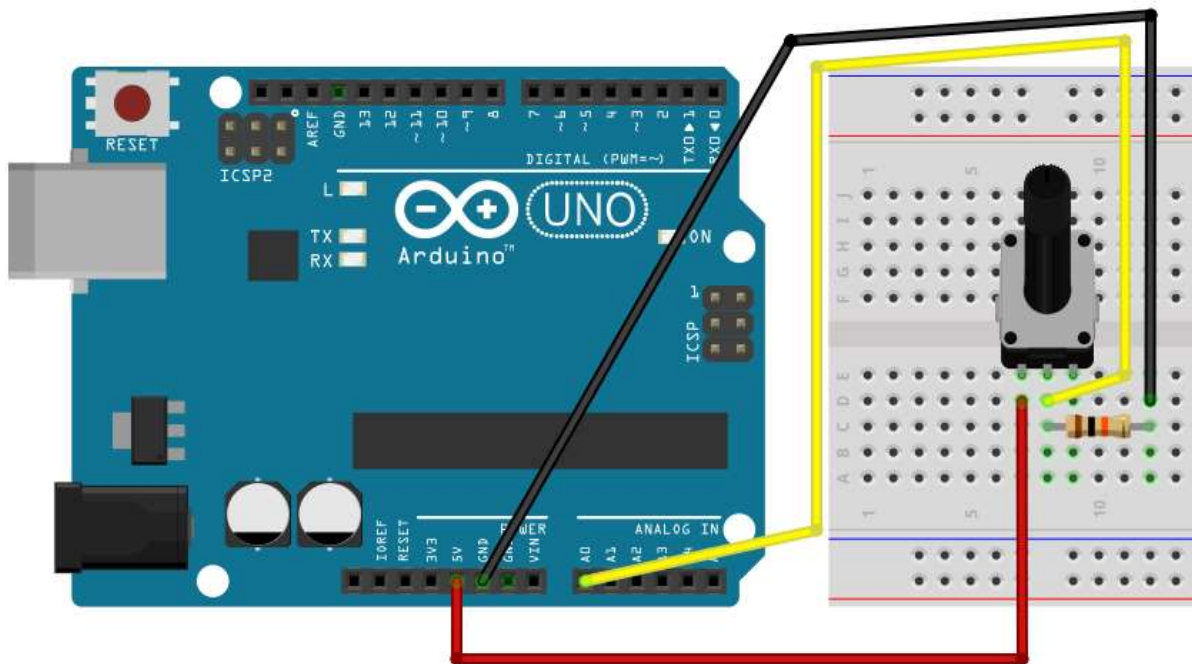
Потенциометр



Давайте теперь познакомимся с потенциометром. Потенциометр представляет из себя резистор с поворотной ручкой для изменения сопротивления, думаем вы встречали такие потенциометры в регуляторах громкости на динамиках например или для изменения температуры на холодильнике. Такие потенциометры бывают разного сопротивления, например на рисунке изображен потенциометр B100K, это означает что мы можем менять его сопротивление от 0 до 100кОм, когда поворачиваем ручку. У потенциометра есть 3 ножки, используется обычно всегда

две: центральная и крайняя. Но зачем тогда третья? Смотрите; если потенциометр повернут на максимум, то между 1 и 2 ножкой будет максимальное сопротивление, а если на минимум, то минимальное. Со 2 и 3 ножками наоборот, если потенциометр повернут максимально, то

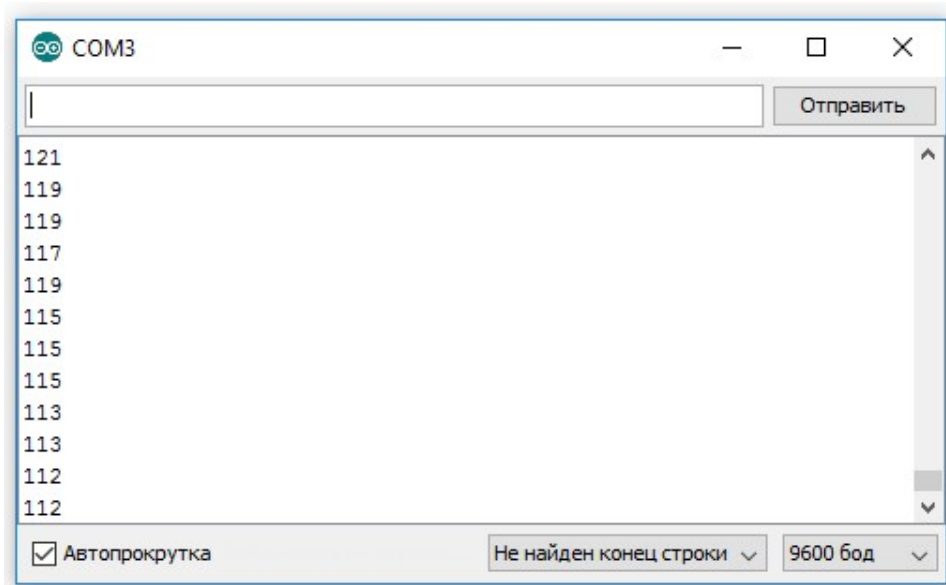
между 2 и 3 ногой минимальное сопротивление, а если на минимум - то максимальное. При подключении к Arduino на одну из крайних ножек обычно подается плюс а со средней считываются показания, то есть среднюю мы подключаем к аналоговому порту A0 например. Давайте подключим наш потенциометр:



И напомним код который считывает и выводит показания с аналогового порта A0:

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(analogRead(A0));  
}
```

Теперь давайте откроем монитор последовательного порта и покрутим ручку потенциометра, тогда мы увидим как значения выводимые на экран изменяются:



А теперь давайте выполним несколько заданий:

1. *Сделайте программную регулировку яркости светодиода с помощью потенциометра. Используйте для этого функции `analogRead()` и `analogWrite()`. Получилось, тогда подключите вместо светодиода тональный динамик и проделайте тоже самое с ним.*
2. *Помните на прошлом уроке мы делали так, чтобы при достижении определенной температуры (40 градусов Цельсия например) начинал пищать динамик? Теперь сделайте тоже самое, но так, чтобы порог температуры можно было менять с помощью потенциометра, в диапазоне от 0 до 60 градусов Цельсия, используйте для этого функцию `map()`.*

Циклы `for()`

В программировании есть такое понятие как циклы. Это такие блоки программы, которые повторяются. В Arduino вы уже знакомы с циклом `loop()`, который выполняется бесконечно. Но мы так же можем создавать свои циклы. Для этого существует команда `for()`, полностью пишется она следующим образом:

```
for (int i=0; i < 255; i++){  
  //Здесь наш код который мы хотим повторять  
}
```

Теперь разберем `for()` по порядку, внутри фигурных скобок всё ясно, там как всегда будет лежать наш код, в данном случае он будет повторяться, а вот внутри круглых всё немного интереснее, как мы видим сначала мы прямо внутри скобок объявляем переменную `int i` и сохраняем в нее число 0. После этого ставится точка с запятой и пишется условия `i < 255`. Для чего же в цикле у нас условие, это ведь не `if()`?

Условие здесь нужно для того чтобы цикл повторялся, если это условие будет верным то цикл будет повторяться до тех пор пока условие не станет неверным, после каждого выполнения цикл проверяет это условие и если оно верно, то цикл повторяет весь код снова, но ведь переменная `i` у нас равно нулю, это значит что условие `i < 255` будет всегда верным и цикл будет выполняться вечно? Для этого после условия идет команда на увеличение `i++`, благодаря ей наша переменная будет увеличиваться, эта команда будет выполняться после каждого прохода цикла, то есть каждый раз после того, как цикл сработал и выполнил наш код в

фигурных скобках, цикл начинает заново и увеличивает нашу переменную **i** еще на единицу, в итоге через 255 срабатываний цикла она увеличится до 255 и условие перестанет срабатывать. Это и есть основное отличие цикла **for()** от **loop()**, здесь мы можем задавать нужное нам количество выполнений цикла, переменную **i** мы можем называть как угодно, условие тоже можем ставить какое захотим **>**, **<**, **==**, **!=**, **<=** или **>=**, мы можем увеличивать переменную **i**, уменьшать, присваивать ей другую переменную и т.д. Например следующий цикл выполнится тоже 255 раз, только на этот раз **i** будет равно 255 и мы будем его уменьшать:

```
for (int i=255; i > 0; i--){  
  //Здесь наш код который мы хотим повторять  
}
```

Мы можем использовать цикл как в блоке **loop()** так и в **setup()**. Давайте попробуем написать тот же самый код мигания плавного зажигания светодиода, но сделаем это в блоке **setup()**

```
void setup() {  
  for (int i=0; i < 255; i++){  
    analogWrite(3, i);  
    delay(10);  
  }  
}  
  
void loop() {  
  
}
```

Согласитесь намного быстрее и удобнее? не нужны никакие условия и переменные, единственная переменная которая используется, это переменная **i**.

Давайте сделаем теперь следующие задания:

1. *Допишите код выше так, чтобы светодиод еще и плавно потух, но не используйте при этом условия **if()**. Что нужно сделать чтобы светодиода все время плавно загорался и выключался? Как сделать чтобы светодиод все время загорался и выключался в **setup()**?*
2. *Сделайте программу которая возводит число в степень и возвращает ответ, это необходимо сделать с помощью цикла, мы знаем что например 2 в 3 степени, это все равно, что 2 * 2 * 2. Воспользуйтесь этим правилом и посчитайте с помощью цикла, сколько будет 2 в степени 9.*

Урок №8

Тема: Подключение библиотек, знакомство с ультразвуковым дальномером

На этом уроке мы познакомимся с ультразвуковым модулем измерения расстояния (дальномером) и подключением библиотек кода в Arduino.

Дальномер мы будем изучать на примере модуля HC-SR04. Принцип работы ультразвуковых модулей измерения расстояния - прост, сначала модуль посылает ультразвуковой сигнал (частотой 40кГц) после чего слушает его эхо, сигнал отправляется, ударяется о какой либо предмет и возвращается обратно, засекая время между отправкой сигнала и получением эха, можно узнать расстояние до объектов. По такому же принципу измеряют расстояние до объектов дельфины.

Модуль HC-SR04 измеряет расстояние в диапазоне от 4 до 200 см. Точность измерения при этом варьируется от +1см до +3 см, в зависимости от дальности.

Недостатком этого модуля является, то что он не определит правильно расстояние до объекта, который поглощает или пропускает звук, в общем предметы которые не отражают звук.

Например густая шерсть, или тоненький слой марлевой ткани

Но этот модуль очень прост в использовании, у модуля всего 4 разъема:

VCC	Порт питания модуля, модуль питается от 5 вольт.
TRIG	На этот порт необходимо подавать сигнал +5V для того, чтобы модуль отправлял ультразвуковой сигнал, подачу нужно осуществлять в течении 10 микросекунд (0,01 миллисекунд)
ECHO	ЭХО - на этот подается импульс когда дальномер получает эхо, чем дольше времени шло эхо тем длиннее импульс, тем больше расстояние, зная это - можно высчитывать расстояние до объектов.
GND	Земля

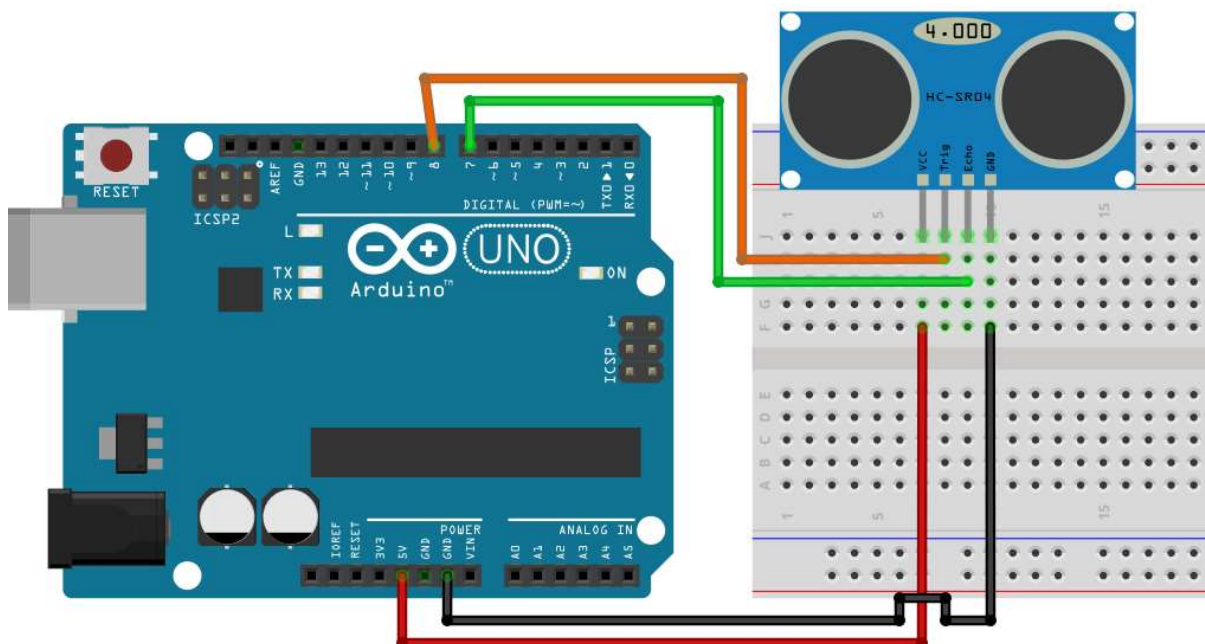
И так, чтобы измерить расстояние до объекта, нам необходимо сначала послать импульс затем считать длину импульса полученного от сенсора. Для того чтобы узнать длину импульса мы будем использовать функцию **pulseIn()**, у этой функции 2 параметра: номер порта и уровень сигнала HIGH или LOW с которого нужно начать отсчет измерения, например если мы напишем **pulseIn(9, HIGH)**, то функция **pulseIn()** будет ждать пока на 9 порт будет подан сигнал HIGH (5 вольт) и после этого начнет отсчет времени до тех пор пока на порт не будет подан низкий сигнал LOW (0 вольт). Измерив время функция возвращает его в микросекундах. И так мы подаем 5 вольт на порт trig, ждем 10 микросекунд и начинаем считывать длину импульса с порта echo. Но как подождать 10 микросекунд? Ведь у функции **delay()** минимальное значение 1 миллисекунда, а это в 100 раз больше чем 10 микросекунд.

В этом нам поможет другая функция **delayMicroseconds()**, эта функция как и **delay()** делает паузу при выполнении программы, но в отличии от **delay()** она принимает время в микросекундах, от 1 до 1000. В одной миллисекунде - 1000 микросекунд. Это означает что **delayMicroseconds(1000)** будет делать тоже самое, что и **delay(1)**.

И так мы отправили звук, считали длину импульса функцией **pulseIn()**, теперь нам необходимо зная длину импульса узнать расстояние до предмета. Для этого достаточно разделить время импульса на 58 (на это число рекомендует делить производитель датчиков) и тогда мы получим расстояние до объекта в сантиметрах. Число 58 это просто константа вычисленная для этого модуля. Чтобы вычислить длину в дюймах необходимо делить на число 148.

Дальше цикл повторяется и можно посылать звук и измерять расстояние заново. Очень желательно между отправкой ультразвука делать паузу не меньше 50 миллисекунд иначе измерения могут быть не точными.

Давайте теперь наконец подключим наш модуль и протестируем его работу:



Теперь напомним следующую программу и загрузим ее:

```
const int trig = 8; //Константа с номером порта, в который подключен trig
const int echo = 7; //Константа с номером порта, в который подключен echo

void setup()
{
  pinMode(trig, OUTPUT); //На порт trig мы будем подавать напряжение
  pinMode(echo, INPUT); //С порта INPUT мы будем считывать

  Serial.begin(9600);
}
```

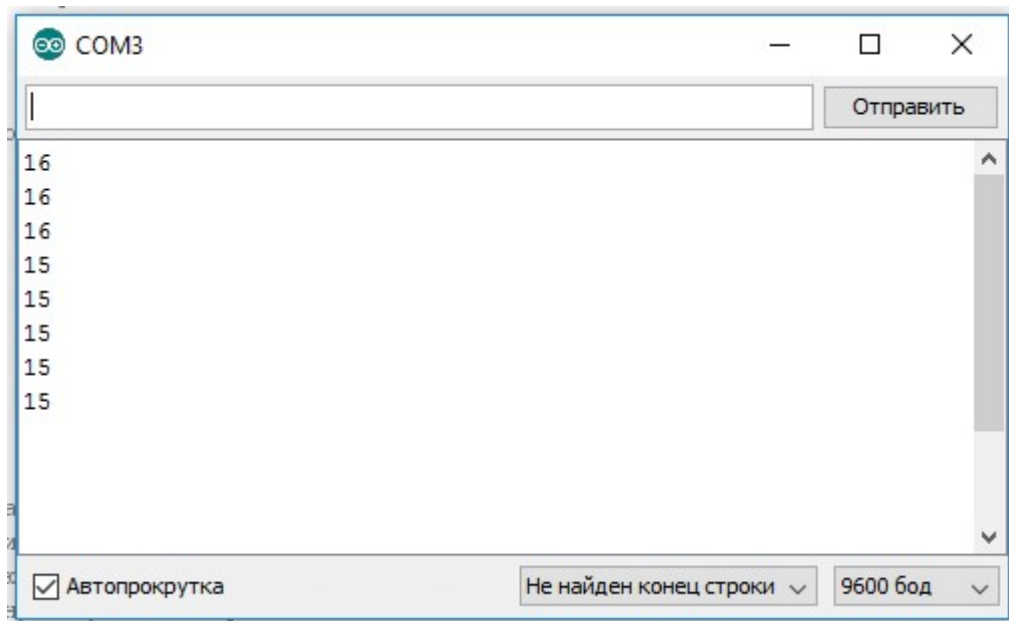
```

unsigned int timeSignal = 0;
unsigned int distanceSm = 0;

void loop()
{
  digitalWrite(trig, HIGH);          // Подаем сигнал на на разъем TRIG
  delayMicroseconds(10);             // Удерживаем 10 микросекунд
  digitalWrite(trig, LOW);           // Отключаем сигнал
  timeSignal = pulseIn(echo, HIGH);  // Замеряем длину импульса с порта ECHO
  distanceSm = timeSignal / 58;      // Пересчитываем в сантиметры
  Serial.println(distanceSm);        // Выводим на порт
  delay(100);
}

```

Теперь запустим нашу программу и откроем монитор последовательного порта. Здесь мы увидим расстояние до ближайшего объекта в сантиметрах, попробуйте направить дальномер на разные предметы и измерить расстояние до них.



Вот таким не особо сложным способом можно определять расстояние до объектов. Но существует еще более простой способ, не писать весь код самому, а использовать библиотеку кода. Что такое библиотека кода? Библиотека кода, это просто наборы различного кода, что бы вам не самим писать его с нуля, а просто подключить эту библиотеку и пользоваться, там уже есть готовые функции которые сами производят все вычисления, вам остается только чуть изменить своим параметры, загрузить код и наслаждаться работой устройства.

Для нашего ультразвукового дальномера тоже существует библиотека. Скачать ее можно по следующей ссылке https://radiomart.org/index.php?controller=attachment&id_attachment=572 Либо на странице с описанием данного модуля https://RadioMart.org/_975.

Теперь необходимо установить нашу библиотеку в Arduino, для этого распаковываем скачанный архив, в нем есть папка Ultrasonic

Загрузки > ultrasonic-HC-SR04			Поиск: ultrasonic-HC-SR04
Имя	Дата изменения	Тип	
Ultrasonic	15.02.2012 14:22	Папка с файлами	

Эта папка нам и нужна. Внутри это папки хранятся файлы нашей библиотеки.

ultrasonic-HC-SR04 > Ultrasonic			Поиск: Ultrasonic
Имя	Дата изменения	Тип	
examples	15.02.2012 14:26	Папка с файлами	
keywords.txt	24.04.2010 20:55	Текстовый докум...	
Ultrasonic.cpp	15.02.2012 14:33	Файл "CPP"	
Ultrasonic.h	15.02.2012 14:25	Файл "H"	

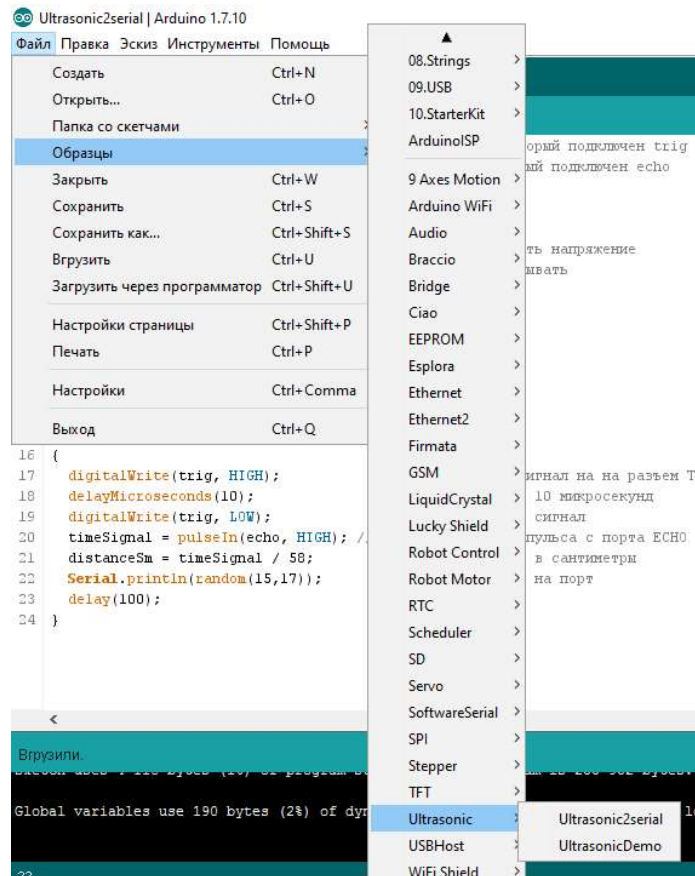
Чтобы установить библиотеку в Arduino необходимо скопировать папку Ultrasonic в папку с библиотеками Arduino. Она находится в папке с установленной Arduino IDE.

Обычно это папка **C:\Program Files (x86)\Arduino\libraries**, у вас это может быть другая папка, смотря куда вы устанавливали Arduino IDE. В этой папке есть уже множество библиотек. Скопируем туда и нашу:

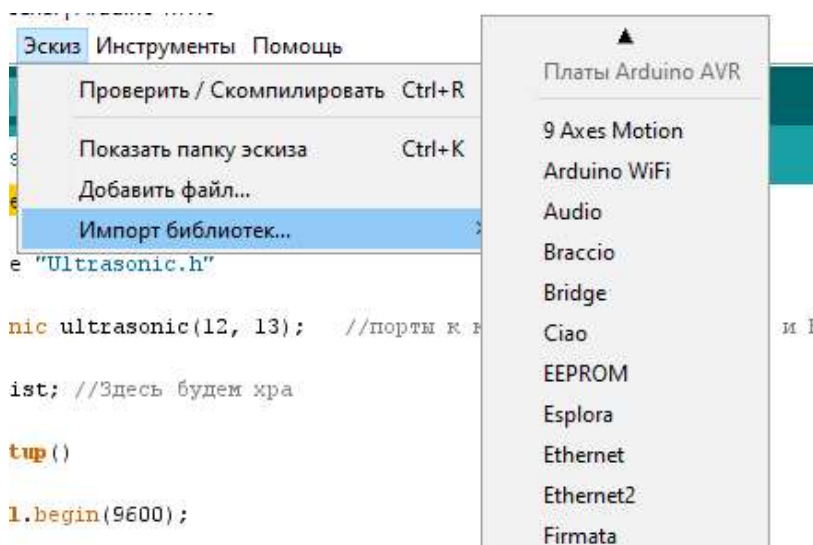
6) > Arduino > libraries			Поиск:
Имя	Дата изменения	Тип	
NAxesMotion	29.05.2016 21:27		
Robot_Control	29.05.2016 21:27		
Robot_Motor	29.05.2016 21:27		
RobotIRremote	29.05.2016 21:27		
RTC	29.05.2016 21:27		
Scheduler	29.05.2016 21:27		
SD	29.05.2016 21:27		
Servo	29.05.2016 21:27		
SpacebrewYun	29.05.2016 21:27		
Stepper	29.05.2016 21:27		
Temboo	29.05.2016 21:27		
TFT	29.05.2016 21:27		
Ultrasonic	22.06.2016 5:49		
USBHost	29.05.2016 21:27		
WiFiShield	29.05.2016 21:28		

Теперь необходимо перезапустить Arduino IDE если программа у нас открыта.

Новая библиотека должна появиться у нас в меню **Файл -> Образцы -> Ultrasonic**.



Поздравляем! Библиотека установлена, теперь мы сможем ее использовать, для того чтобы использовать библиотеку необходимо ее подключить к программе, для этого используется такая команда: `#include "Ultrasonic.h"` или `#include <Ultrasonic.h>` разницы между ними - нет, обе команды подключают библиотеку "Ultrasonic". Библиотеки подключаются обычно в самом верху программы, команда `#include` переводится с английского языка как включить, подключить. В Arduino IDE так же есть удобное меню для подключения библиотек, чтобы вам в ручную не писать команду подключения, можно просто перейти в меню **Эскиз -> Импорт библиотек...** и выбрать библиотеку которую вы хотите подключить, тогда подключение библиотеки автоматически пропишется у вас в коде.



И так давайте теперь попробуем переписать нашу программу с использованием библиотеки:

```
#include "Ultrasonic.h" //Подключаем библиотеку

Ultrasonic ultrasonic(8, 9); //порты к которым подключено TRIG и ECHO

float dist; //Здесь будем хранить расстояние

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

void loop()
{
  dist = ultrasonic.Ranging(CM); //Получаем расстояние в сантиметрах
  Serial.println(dist);          //Выводим в монитор порта

  delay(100);
}
```

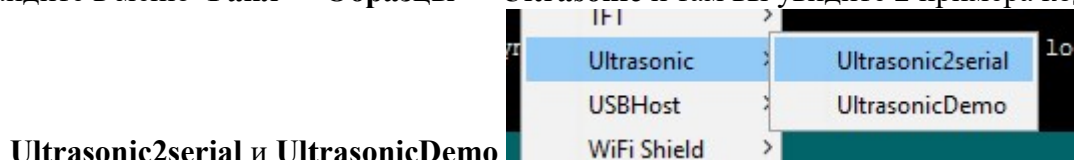
Теперь загрузим наш код в Arduino и откроем монитор последовательного порта, мы увидим, что он работает как и предыдущий, только нам не пришлось вручную подавать ультразвук, ждать 10 микросекунд, потом считывает импульс и делить его на 58. Наша библиотека сделала все сама с помощью функции **ultrasonic.Ranging(CM)**.

Эта функция как вы могли догадаться делает все расчеты сама и возвращает расстояние до объекта в сантиметрах. Если мы в параметрах у функции укажем не **CM** а **INC**, например: **ultrasonic.Ranging(INC)**, то эта функция будет возвращать расстояние до предмета в дюймах.

Думаю вы еще заметили не знакомый код **Ultrasonic ultrasonic(8, 9);** Этот код инициализирует (подготавливает) модуль и библиотеку к работе, цифры 8 и 9 это номера портов Arduino к которым подключены пины TRIG и ECHO модуля.

Теперь наша программа занимает намного меньше кода.

Но расскажем вам еще 1 небольшой секрет, весь этот код мы тоже могли не писать вручную, обычно у библиотек есть примеры, в которых уже есть готовый код, а вам нужно только изменить какие-то параметры. Как же получить эти примеры? Да очень просто, для этого зайдите в меню **Файл -> Образцы -> Ultrasonic** и там вы увидите 2 примера кода,



Ultrasonic2serial и UltrasonicDemo

кликните например по первой и у вас откроется готовая программа, в ней уже есть комментарии кода, объясняющие, что этот код делает. Комментарии на английском, но если вы не знаете английского то всегда можно воспользоваться переводчиком, например <https://translate.google.com/>. И так мы видим что код почти как наш, только номера портов отличаются, здесь они 12 и 13. Мы можем переподключить наш модуль в эти порты, либо изменить в программе порты, на те в которые подключен наш модуль, как хотите. В итоге мы загружаем программу и она работает, и на всё про всё ушло пару минут, даже если мы не знаем как работает модуль изнутри.

```

#include "Ultrasonic.h"

// sensor connected to:
// Trig - 12, Echo - 13
Ultrasonic ultrasonic(12, 13);

void setup()
{
  Serial.begin(9600); // start the serial port
}

void loop()
{
  float dist_cm = ultrasonic.Ranging(CM); // get distance
  Serial.println(dist_cm); // print the distance

  delay(100); // arbitrary wait time.
}

```

Вот так можно упростить кодирование с помощью библиотек. В будущем мы будем очень часто использовать библиотеки, поэтому запомните хорошенько как их подключать.

А теперь давайте выполним следующие задания:

1. Помните на предыдущих уроках мы делали терменвокс, мы делали, чтобы динамик звучал тише либо громче от того, что мы ближе или дальше подносим нашу руку, но делали мы это с помощью датчика света. Давайте теперь то же самое сделаем с дальнометром, чем дальше находится предмет, тем выше пусть пищит тональный динамик, а чем ближе, тем тише.
2. Сделайте чтобы при приближении предмета, динамик начинал пищать все быстрее и быстрее как датчики парковки на автомобилях, при приближении препятствия начинают все быстрее и быстрее пищать.
3. Сделайте датчик движения с включением света, для этого используйте светодиод. Установите дальнометр в неподвижное положение, а в коде сделайте чтобы при первом запуске высчитывалось расстояние до ближайшего объекта, а если это расстояние изменилось, значит мимо что-то двинулось или кто-то прошел, и нам необходимо включить светодиод на 5 сек, чтобы осветить путь.

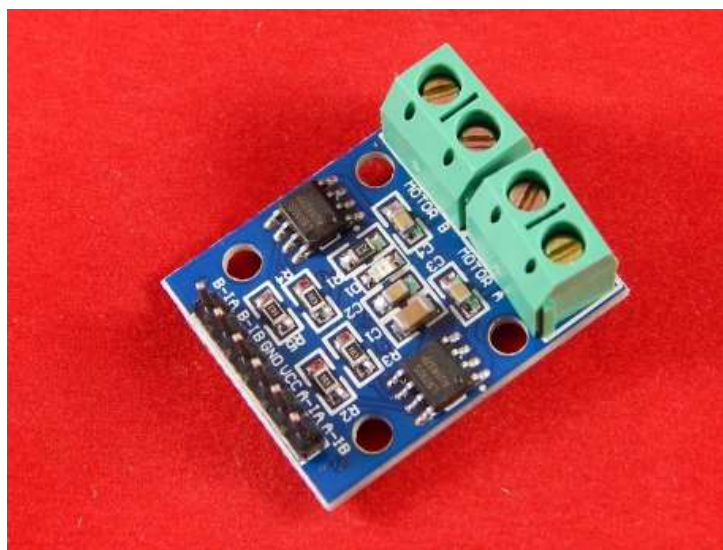
Урок №9

Тема: Управление DC двигателем через Arduino

Наконец-то мы дошли до урока где будем учиться управлять двигателями при помощи Arduino. Учиться мы будем на примере 6V двигателя постоянного тока с редуктором 1:120.



Этот двигатель без нагрузки дает 100 оборотов в минуту, и потребляет минимум 60мА, а если дать двигателю нагрузку, то потребление вырастет еще больше. Порты Arduino могут выдавать ток максимум 40мА. Это означает, что нам не хватит тока чтобы просто запустить эти двигатели на полную мощность, поэтому просто так подключить этот двигатель к порту Arduino нельзя. Как же быть? Для этих целей были созданы драйверы двигателей, это такие устройства, которые могут управлять двигателями, регулировать их направление и скорость вращения, при этом они могут использовать собственное, отдельное питание, а управлять самим драйвером, давать ему команды, мы уже можем с помощью Arduino. Что же из себя представляют драйверы? Они всегда выглядят по разному, в зависимости от типа двигателя, от его мощности и т.д. Мы будем работать с драйвером L9110S



Это не дорогой драйвер, который как раз подойдет для наших целей, он может одновременно управлять 2 двигателями и максимальный ток который он может выдавать на 1 двигатель составляет 800мА, а поддерживаемое напряжение 5 - 12 вольт, это более чем достаточно для нашего двигателя.

Данный драйвер двигателей может управлять как направлением вращения так и его скоростью. У драйвера имеется 6 контактов для подключения описанные в таблице ниже:

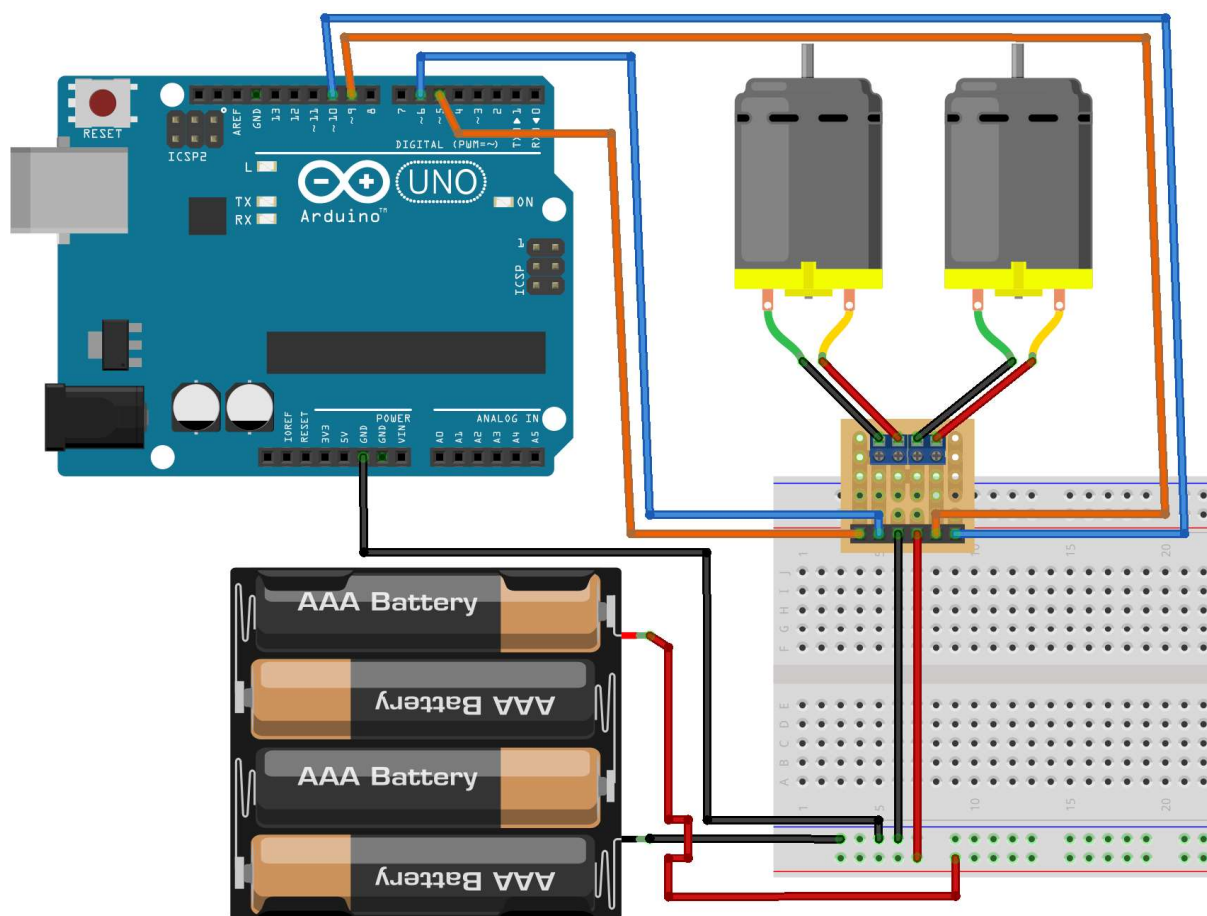
Название	Куда подключаем
B-IA	Подключается к цифровому порту Arduino, с помощью этого порта можно менять направление вращения двигателя подключенного к клемме Motor B.
B-IB	Подключается к цифровому порту Arduino, с помощью этого порта можно скорость и направление вращения двигателя подключенного к клемме Motor B.
GND	Земля, подключается к минусу (-) питания
VCC	Порт питания модуля, подключается к питанию от +2.5 до +12 вольт
A-IA	Подключается к цифровому порту Arduino, с помощью этого порта можно менять направление вращения двигателя подключенного к клемме Motor A.
A-IB	Подключается к цифровому порту Arduino, с помощью этого порта можно скорость и направление вращения двигателя подключенного к клемме Motor A.

Так же у двигателя имеются 2 клеммы, для подключения самих двигателей, на клеммах написано Motor A и Motor B. Сюда мы подключаем плюс и минус двигателя, разницы нет куда подключать плюс двигателя, а куда минус, но желательно чтобы оба двигателя подключались одинаково, чтобы например у обоих слева был плюс, а справа минус или наоборот. Иначе, если мы подключим их по разному то в коде будет неудобно ими управлять. И так, как же работать с нашим драйвером двигателя.

Для начала необходимо его подключить к питанию и к Arduino. Подключить питание к драйверу от Arduino напрямую нельзя, потому что у стабилизатора Arduino не хватит мощности вращать наши двигатели. Наши двигатели работают от 6 вольт, значит и на наш драйвер - на порт VCC мы должны подавать не меньше 6 вольт, для максимальной скорости работы двигателя. Для этого можно использовать 4 пальчиковых батарейки 1.5 вольта. Желательно использовать дорогие батарейки типа DURACELL, потому что от китайских кислотно-щелочных батареек двигателям может даже не хватить мощности на запуск. Либо можно использовать 2 литий-ионных батарейки на 3.7 вольта, тогда 2 такие батарейки будут давать 7.4 вольта, это чуть больше чем нам нужно. Но в нашем случае ничего страшного нет, двигатели будут работать от такого напряжения. И так мы подключили VCC от драйвера к плюсу от нашего дополнительного источника питания, а GND подключили к минусу. Прежде чем подключить наш драйвер к Arduino. Необходимо минус драйвера GND так же подключить к Arduino. VCC подключать к Arduino не нужно! Только GND. Получается что порт GND у нас одновременно будет подключен и к минусу на батарейках и к GND на Arduino. Если 2

устройства подключаются друг к другу и у них разные источники питания (Arduino например работает через USB, а драйвер от батареек) то их минусы (-) ВСЕГДА должны объединяться, иначе устройства могут глючить либо вообще выйти из строя.

И так мы подключили VCC плюсу, а GND подключили к минусу и Arduino.



Теперь осталось подключить управляющие контакты B_IA и B_IB для одного мотора, а так же A_IA и A_IB для второго мотора. Их можно подключать к любым цифровым портам, для примера давайте подключим их следующим образом.

Порт драйвера	Порт Arduino
A_IA	D9
A_IB	D10
B_IA	D5
B_IB	D6

И так, все подключено, как же управлять нашими двигателями. И так схема здесь не сложная, рассмотрим для начала управление одним двигателем. Возьмем для примера двигатель подключенный к Motor A. Чтобы заставить его вращаться необходимо на порт A_IA подать высокий сигнал HIGH а на порт A_IB - низкий LOW. Тогда двигатель начнет вращение на максимальной скорости в одну сторону. И если мы сделаем наоборот и на A_IA подадим

низкий сигнал LOW, а на A_IB - высокий HIGH, то двигатель начнет вращаться в другую сторону. Для того чтобы приостановить вращение нужно и на порт A_IA и на порт A_IB подать низкий сигнал LOW. Но в таком режиме двигатели будут просто остановлены, но все равно будут немного расходовать энергию батарей. Чтобы полностью отключить двигатели нужно подать на порт A_IA и на порт A_IB высокий сигнал HIGH. Чтобы легче запомнить все эти комбинации мы посмотрите на таблицу ниже:

Напряжение на A_IA	Напряжение на A_IB	Получаемое действие
LOW (0 вольт)	LOW (0 вольт)	Двигатель Motor A остановлен
HIGH (5 вольт)	HIGH (5 вольт)	Двигатель Motor A отключен
HIGH (5 вольт)	LOW (0 вольт)	Двигатель Motor A вращается в одну сторону
LOW (0 вольт)	HIGH (5 вольт)	Двигатель Motor A вращается в другую сторону

С двигателем, который подключен к клемме Motor B всё точно так же:

Напряжение на B_IA	Напряжение на B_IB	Получаемое действие
LOW (0 вольт)	LOW (0 вольт)	Двигатель Motor B остановлен
HIGH (5 вольт)	HIGH (5 вольт)	Двигатель Motor B отключен
HIGH (5 вольт)	LOW (0 вольт)	Двигатель Motor B вращается в одну сторону
LOW (0 вольт)	HIGH (5 вольт)	Двигатель Motor B вращается в другую сторону

И так принцип работы нам понятен, давайте попробуем теперь поворачивать хотя-бы 1 мотор, для этого, как указано в таблицах выше нужно подавать на A_IB всегда высокое напряжение, тогда если мы будем подавать на порт A_IA низкий сигнал LOW, то двигатель будет вращаться. если будем подавать высокий сигнал HIGH, то двигатель будет останавливаться:

```
#define A_IA 9 //Константа - Номер порта к которому подключен A_IA
#define A_IB 10 //Константа - Номер порта к которому подключен A_IB

void setup()
{
  pinMode( A_IA, OUTPUT ); //Устанавливаем цифровые порты в режим вывода
  pinMode( A_IB, OUTPUT ); //Устанавливаем цифровые порты в режим вывода

  digitalWrite( A_IA, HIGH ); //Подаем на оба порта высокое напряжение
```

```

digitalWrite( A_IB, HIGH ); //Чтобы отключить двигатели
}

void loop()
{
digitalWrite( A_IA, LOW ); //Вращаем двигатель в одну сторону
delay(3000);

digitalWrite( A_IA, HIGH ); //Отключаем вращение двигателя
delay(3000);
}

```

Теперь двигатель каждые 3 секунды будет вращаться, потом 3 секунды не вращаться. Теперь давайте сделаем чтобы двигатель каждые 3 секунды вращался в обратную сторону. Для этого, как говорилось выше, нужно наоборот на порт A_IA подавать всегда высокое напряжение, тогда если на порт A_IB мы будем подавать низкое напряжение, то двигатель будет крутиться в обратную сторону:

```

#define A_IA 9 //Константа - Номер порта к которому подключен A_IA
#define A_IB 10 //Константа - Номер порта к которому подключен A_IB

void setup()
{
pinMode( A_IA, OUTPUT ); //Устанавливаем цифровые порты в режим вывода
pinMode( A_IB, OUTPUT ); //Устанавливаем цифровые порты в режим вывода

digitalWrite( A_IA, HIGH ); //Подаем на оба порта высокое напряжение
digitalWrite( A_IB, HIGH ); //Чтобы отключить двигатели
}

void loop()
{
digitalWrite( A_IB, LOW ); //Вращаем двигатель в обратную сторону
delay(3000);

digitalWrite( A_IB, HIGH ); //Отключаем вращение двигателя
delay(3000);
}

```

Так как у нас есть дополнительный источник питания - батарейки, то мы можем сделать так, чтобы Arduino работало без питания от USB, для этого нужно отключить USB от Arduino, и соединить плюс (+) от батареек к порт VIN на Arduino, а минус (-) от батареек с минусом GND от Arduino, но земли мы и так объединили в начале урока, так что нам остается только подключить плюс к VIN и наша Arduino будет работать автономно без питания от компьютера.

ВНИМАНИЕ! Не подключайте к Arduino одновременно питание от батареи и от USB, это может вывести из строя как Arduino так и Ваш компьютер. Потому перед прошивкой, **ОБЯЗАТЕЛЬНО ОТКЛЮЧАЙТЕ ПИТАНИЕ ОТ VIN.**

А теперь выполните следующие задания:

1. Так чтобы двигатель вращался сначала в одну сторону потом в другую поочередно, например каждые 3 секунды, поставьте машинку на пол и посмотрите как она будет себя вести.
2. Подключите второй двигатель и сделайте чтобы оба двигателя сначала вращались в одну сторону потом в другую поставьте машинку на пол и посмотрите как она будет себя вести.
3. Сделайте, чтобы двигатели вращались в противоположенные стороны поставьте машинку на пол и посмотрите как она будет себя вести.
4. Подключите датчик расстояния и сделайте так, чтобы если расстояние до ближайшего объекта меньше 40 см, то машинка отъезжала назад, пока расстояние не станет больше 40 см, а если расстояние до ближайшего объекта больше 60 см, то машинка подъезжает вперед, пока расстояние не станет меньше 60 см.

Так же мы можем управлять скоростью двигателей, для этого нам нужно лишь только подключить управляющие порты A_IA, A_IB, B_IA и A_IB к ШИМ портам. Тогда кода будет еще меньше потому что нам не нужно будет делать команду pinMode(); Давайте сделаем опять же первый пример в котором двигатель подключенный к Motor A будет вращаться 3 секунды а потом останавливаться на 3 секунды, только теперь это сделаем с помощью функции analogWrite():

```
#define A_IA 9 //Константа - Номер порта к которому подключен A_IA
#define A_IB 10 //Константа - Номер порта к которому подключен A_IB

void setup()
{
  analogWrite( A_IA, 255 ); //Подаем на оба порта высокое напряжение
  analogWrite( A_IB, 255 ); //Чтобы отключить двигатели
}

void loop()
{
  analogWrite( A_IA, 0 ); //Вращаем двигатель в одну сторону
  delay(3000);

  analogWrite( A_IA, 255 ); //Отключаем вращение двигателя
  d
```

У нас пока ничего не изменилось, двигатель так же как и раньше то крутится, то не крутится. Только происходит это с помощью ШИМ сигнала, когда подаем 0 на порт A_IA и 255 на порт A_IB, то двигатель вращается, а если на оба порта подаем 255 то двигатель отключается. А давайте попробуем изменить кода и на порт A_IA подавать не 0, например 127:

```
#define A_IA 9 //Константа - Номер порта к которому подключен A_IA
#define A_IB 10 //Константа - Номер порта к которому подключен A_IB

void setup()
{
  analogWrite( A_IA, 255 ); //Подаем на оба порта высокое напряжение
  analogWrite( A_IB, 255 ); //Чтобы отключить двигатели
```

```

}

void loop()
{
  analogWrite( A_IA, 127 ); //Вращаем двигатель в одну сторону, но медленнее
  delay(3000);

  analogWrite( A_IA, 255 ); //Отключаем вращение двигателя
  delay(3000);
}

```

Теперь двигатель будет вращаться в 2 раза медленнее, если мы заменим число на 200, то он будет вращаться еще медленнее, и так чем больше мы будем подавать тем медленнее он будет вращаться, если подадим 255, то он остановится, потому что на обоих портах будет высокое напряжение, а по таблицам выше мы знаем что если на портах высокое напряжение, то двигатель отключается.

Чтобы заставить вращаться двигатель в обратную сторону с помощью ШИМ порта, нам точно так же теперь нужно подавать 0 не на A_IA, а на A_IB, а на A_IA всегда подавать 255, и если мы будем менять числа на A_IB от 0 до 254 то двигатель будет вращаться в другую сторону от быстрой скорости к медленной:

```

#define A_IA 9 //Константа - Номер порта к которому подключен A_IA
#define A_IB 10 //Константа - Номер порта к которому подключен A_IB

void setup()
{
  analogWrite( A_IA, 255 ); //Подаем на оба порта высокое напряжение
  analogWrite( A_IB, 255 ); //Чтобы отключить двигатели
}

void loop()
{
  analogWrite( A_IB, 127 ); //Вращаем двигатель в обратную сторону, но медленнее
  delay(3000);

  analogWrite( A_IB, 255 ); //Отключаем вращение двигателя
  delay(3000);
}

```

Теперь выполните следующие задания:

1. Сделайте чтобы оба двигателя вращались с помощью `analogWrite()`
2. Сделайте чтобы оба двигателя вращались в 2 раза медленнее максимума
3. Сделайте чтобы 1 двигатель вращался на максимум а другой наполовину, поставьте машинку на пол и посмотрите, что будет
4. Сделайте так чтобы оба двигателя плавно набирали скорость, а потом плавно снижали.

Урок №10

Тема: Создание своих функций

Сегодня мы изучим создание своих функций, это необходимо для того, чтобы удобнее использовать одни и те же написанные вами куски кода. Мы с Вами познакомились уже с некоторыми стандартными функциями Arduino, такими как: `delay()`, `delayMicroseconds()`, `digitalWrite()`, `digitalRead()`, `analogWrite()`, `analogRead()` и другими.

Настало время научиться писать свои функции. У функций как и у переменных есть типы данных, например `int`, `float`, `bool` и другие. Но если в переменной мы указываем тип данных для того, чтобы указать какие данные будут храниться в переменной, то у функций мы задаем тип данных для того, чтобы указать какое значение функция будет возвращать. Создать функцию очень просто, создаются они за пределами `setup()` { } и `loop()` { } и очень похожи на них, давайте посмотрим пример создания своей функции.

```
int myFunction(){
  return 123;
}

void setup(){
}

void loop(){
}
```

Мы создали свою функцию под названием **myFunction**, внутри этой функции только одна команда **return 123**; `return` - с английского языка переводится как “вернуть”. Как можно догадаться то эта команда всегда возвращает число 123 из функции. Поэтому тип нашей функции мы указали **int**, потому что число у нас всегда возвращается целое число. Мы назвали функцию **myFunction**, это произвольное название, вы можете придумать себе свое любое название, правила такие же как и у названий переменных: только латинские буквы, цифры и знак подчеркивания, название функции так же не может начинаться с цифры и совпадать с другими названиями функций, команд, констант и переменных которые существуют.

Мы написали свою функцию, но сама по себе она ничего не делает если просто написана, теперь нам необходимо ее использовать, использование функции в программировании называется еще вызов функции. Теперь нам достаточно вызвать нашу функцию в любом месте в **setup()** или **loop()**, чтобы она сделала свое единственное и бесполезное действие - вернула число 123. Давайте сделаем это:

```
int myFunction(){
```

```

return 123;
}

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

void loop(){
  Serial.println(myFunction());
}

```

Теперь у нас в монитор последовательного порта будет выводиться наше число 123. Мы можем сохранить возвращаемое функцией число в переменную да и вообще действовать с ней как с обычным числом, ведь у нее тип данных **int**.

Например следующий код сохранить в переменную xxx число 123:

```

int xxx;
xxx = myFunction();

```

А следующий код умножит число возвращаемое функцией на 2 и сохранит в переменную число 246.

```

int xxx;
xxx = myFunction()*2;

```

То есть программа действует с нашей функцией как с обычным числом или переменной типа **int**, но в отличии от переменной - функция возвращает значение с помощью команды **return**.

Но зачем нам писать отдельную функцию, если можно использовать переменную?

Конечно же если в нашей функции есть только строка с командой **return 123;**, то смысла от такой функции мало и проще использовать переменную. Но обычно функции содержат в себе наиболее сложный код. Например:

```

int myFunction(){
  int box=2;
  for(int i=0; i<10; i++){
    box = box * 2;
  }
  return box;
}

```

В этом коде наша функция умножает переменную бокс на 2 в цикле 10 раз и возвращает полученное значение, то-есть мы возвели число 2 в 10 степень. Если мы вызовем нашу функцию, то она вернет число 1024 (то есть она умножила число 2 на 2 целых 10 раз, это все равно что 2 в 10 степени):

```

int myFunction(){
  int box=2;
  for(int i=0; i<10; i++){
    box = box * 2;
  }
  return box;
}

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

void loop(){
  Serial.println(myFunction());
}

```

Теперь если на придется использовать один и тот же код несколько раз, то мы можем просто положить его в функцию и так где необходим наш код - вызывать нашу функцию, а она уже будет в свою очередь выполнять наш код и возвращать результат.
Например:

```

int myFunction(){
  int box=2;
  for(int i=0; i<10; i++){
    box = box * 2;
  }
  return box;
}

int aaa;
void setup(){
  aaa = myFunction(); //Сохраняем наше значение 1024 в переменную aaa
  Serial.begin(9600);
}

void loop(){
  //Этот код выдаст на монитор последовательного порта сумму чисел
  // 1024+1024+1024, то-есть на экран выведется число 3072.
  Serial.println(myFunction() + myFunction() + myFunction());
}

```

В этом коде мы используем нашу функцию несколько раз.
Как вы заметили внутри нашей функции мы создали переменную box, мы можем создавать любые переменные внутри функций. Причем эти переменные программа не увидит снаружи функции. Например следующий код выдаст ошибку, потому что переменную box мы создаем в нашей функции myFunction(), а выводим в функции loop(), поэтому Arduino IDE выдает ошибку, что переменная box не создана :

```
int myFunction(){
  int box=2;
}

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

void loop(){
  Serial.println(box);
}
```

Ошибка компиляции.

Ultrasonic2serial.ino:10:18: error: 'box' was not declared in this scope

Ошибка компиляции.

Точно так же если мы создадим переменную внутри **loop()** или внутри **setup()**, а использовать попытаемся ее внутри нашей функции, то это тоже вызовет ошибку:

```
int myFunction(){
  box=2;
}

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

void loop(){
  int box=2;
  Serial.println(box);
}
```

Ошибка компиляции.

Ultrasonic2serial.ino:2:3: error: 'box' was not declared in this scope

Ошибка компиляции.

Но если мы создадим переменную **box** снаружи функций, то и в нашей функции и в **loop(){ }** и в **setup() { }** мы сможем ее использовать. Например так:

```
int box;
int myFunction(){
  box=2;
```

```

}

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

void loop(){
  box=5;
  Serial.println(box);
}

```

Давайте теперь вернемся к нашему примеру, где мы возводили двойку в 10 степень. И так вы наверное заметили что мы создаем нашу функцию `int myFunction(){ }` в самом верху программы, это совсем не обязательно, можно это сделать и в самом низу и по центру, главное это делать снаружи других функций, создавать функцию внутри другой функции нельзя. Вызывать можно, а создавать нельзя!

Например во всех трех следующих примерах программа будет выводить в монитор порта число 123, программе не важно где создана функция, программе важно где она вызывается, создана она может быть в любом месте но срабатывает она только в том месте где была вызвана:

```

int myFunction(){
  return 123;
}
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.println(myFunction());
}

```

```

void setup(){
  Serial.begin(9600);
}
int myFunction(){
  return 123;
}
void loop(){
  Serial.println(myFunction());
}

```

```

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

```

```
void loop(){
  Serial.println(myFunction());
}
int myFunction(){
  return 123;
}
```

Все эти три примера абсолютно одинаковые не смотря на то, что в них наша функция создается в разных местах, вызывается она всегда в одном - в **loop()**.

В функции мы можем передавать параметры, с этим мы уже сталкивались, когда мы вызываем **delay(1000)** мы передаем туда параметр - количество миллисекунд - 1000.

Когда мы вызываем функцию **digitalWrite(3, HIGH)** мы передаем туда уже 2 параметра, 3 - номер порта и HIGH - уровень сигнала который нужно подать на порт.

В нашу функцию мы тоже можем сделать передачу параметров, давайте снова рассмотрим наш пример в возведении двойки в 10- степень:

```
int myFunction(){
  int box=2;
  for(int i=0; i<10; i++){
    box = box * 2;
  }
  return box;
}

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

void loop(){
  Serial.println(myFunction());
}
```

Предположим мы хотим чтобы в box хранилась не двойка, а другое значение, 5 например.

Чтобы не изменять нашу функцию каждый раз когда мы захотим поменять значение box, мы можем передать в нее параметр, как это делается? Да очень просто, на при создании функции в круглых скобках создать переменную, она и будет нашим параметром, смотрите:

```
int myFunction(int param1){
  int box=param1;
  for(int i=0; i<10; i++){
    box = box * 2;
  }
  return box;
}
```

```
}
```

теперь если мы вызовем нашу функцию и в скобках укажем например тройку:

```
myFunction(3);
```

То эта тройка автоматически сохранится в переменную **param1**, дальше мы сохраняем ее в **int box=param1**; и значит в переменной box у нас тоже теперь число 3 ну и дальше в цикле число наша переменная box начнет умножаться на 2, и в итоге наша функция вернет уже не 1024 как раньше, а **59049**. Но это число больше чем может в себе хранить тип данных **int** поэтому нам надо изменить тип данных нашей функции и сделать ее хотя бы **long int** и тогда код указанный ниже будет выводить в монитор порта число 59049:

```
long int myFunction(int param){
    int box=param;
    for(int i=0; i<10; i++){
        box = box * 2;
    }
    return box;
}

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

void loop(){
    Serial.println(myFunction(3)); //Вызываем нашу функцию с параметром 3
}
```

А что если мы хотим передать в функцию 2 параметра? Все просто, при создлании функции укажите в скобках еще одну переменную, например давайте создадим функцию которая умножает 2 числа и выдает ответ:

```
int myFunction(int param1, int param2){
    return param1 * param2; //Умножаем одну переменную на другую и выдаем результат
}

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

void loop(){
    Serial.println(myFunction(3,10)); //Вызываем нашу функцию с параметром 3 и 10
}
```

При вызове мы передаем нашей функции 2 параметра: 3 и 10 и функция их перемножает выдавая нам результат который мы выводим монитор порта. Таким образом можно писать

функции любой сложности которые потом можно часто использовать в своем коде не повторяя одни и те же куски длинного кода.

А что если нам нужно просто поместить код в функцию который ничего не вычисляет и нам не нужно выдавать ответ, например поместить код который будет заставлять ехать машинку вперед?

Это можно сделать, тогда нам не нужен тип данных у функции, она ведь не будет возвращает какое значение она будет просто крутить двигатели и все. Для этого вместо типа данных нам нужно указать слово **void**, давайте рассмотрим пример:

```
void goBack(){
  digitalWrite( 9, LOW ); //Вращаем двигатель в обратную сторону
  delay(3000);
  digitalWrite( 10, HIGH ); //Отключаем вращение двигателя
  delay(3000);
}

void setup(){
  pinMode( 9, OUTPUT ); //Устанавливаем цифровые порты в режим вывода
  pinMode( 10, OUTPUT ); //Устанавливаем цифровые порты в режим вывода

  digitalWrite( 9, HIGH ); //Подаем на оба порта высокое напряжение
  digitalWrite( 10, HIGH ); //Чтобы отключить двигатели
}

void loop(){
  goBack();
}
```

Мы создали функцию которая вращает двигатель 3 секунды назад а потом делает паузу 3 секунды. Теперь чтобы заставить двигатель делать это постоянно нам достаточно в **loop(){ }** вызвать функцию **goBack()**. Таким же образом можно сделать функции которые будут заставлять двигаться робота прямо, назад, влево, вправо и потом вызывать их в разных ситуациях.

Если вы заметили мы с самого первого урока создаем функции **setup() { }** и **loop() { }** они обе типа **void** потому что не возвращают значения, но только мы их никогда не вызываем, а они все равно выполняются, дело в том что их вызывает сама Arduino автоматически. Это единственное, что Arduino делаем сама, остальное мы ей пишем сами. Так что с первого урока мы умели писать функции, просто не знали об этом.

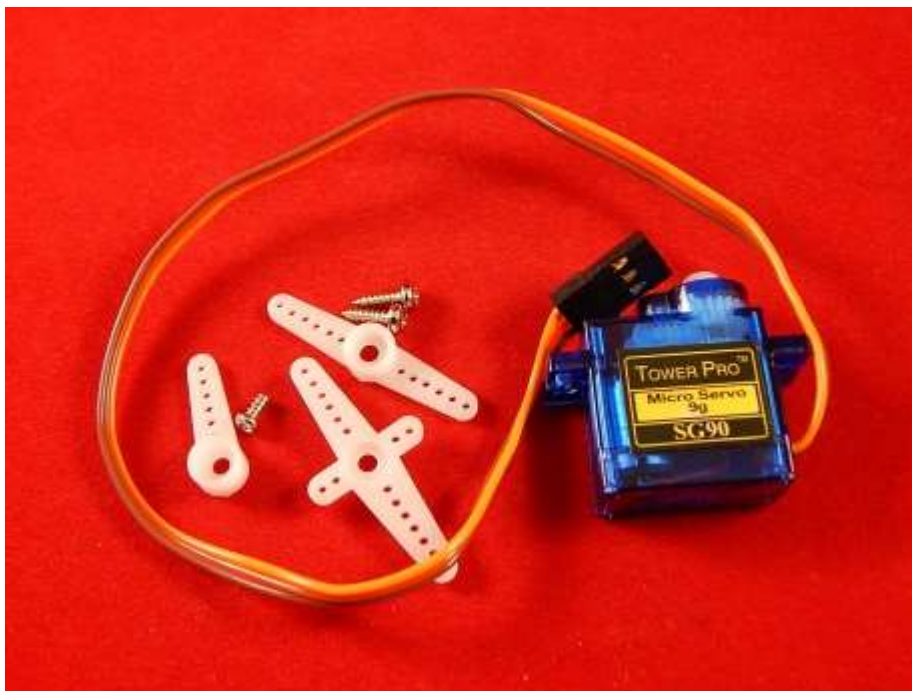
А теперь задания на закрепление темы:

1. *Сделайте функции которые заставляют двигаться робота вперед и назад, вызывайте их по очереди в loop()*
2. *Сделайте функции, которые заставляют поворачивать робота влево и вправо.*
3. *Переделайте функции которые двигают робота вперед и назад, так чтобы мы могли передавать в параметрах скорость их движения от 0 до 255*
4. *Теперь переделайте эти же функции так, чтобы мы могли задавать им скорость от 0 до 10. Сделайте это с помощью функции map()*

Урок №11

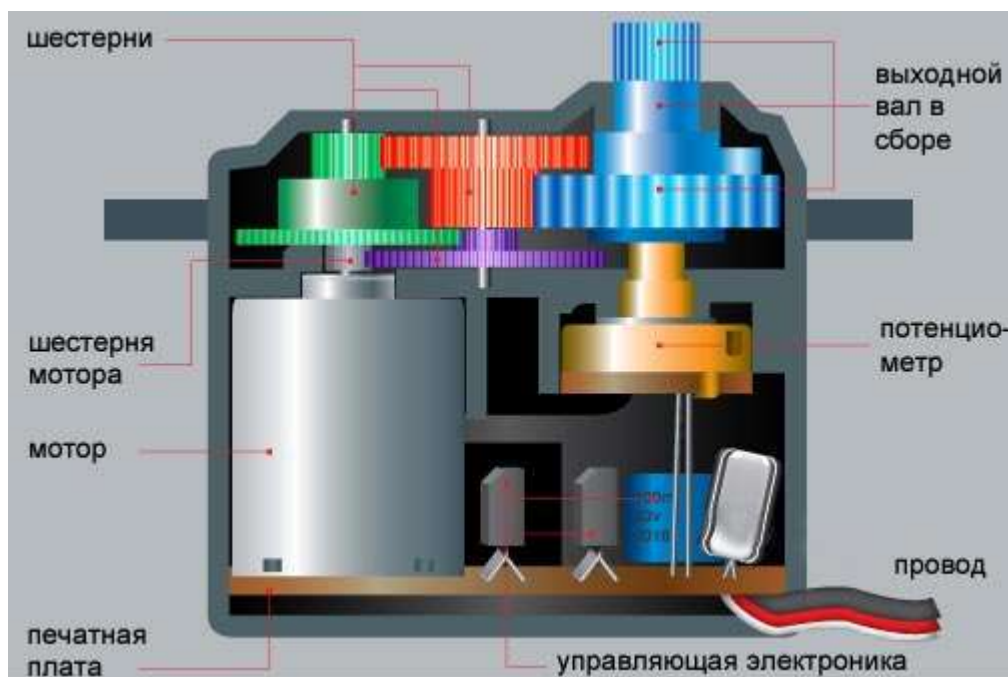
Тема: Знакомство с серводвигателем

На этом уроке мы знакомимся с работой серводвигателей на примере серводвигателя SG90.



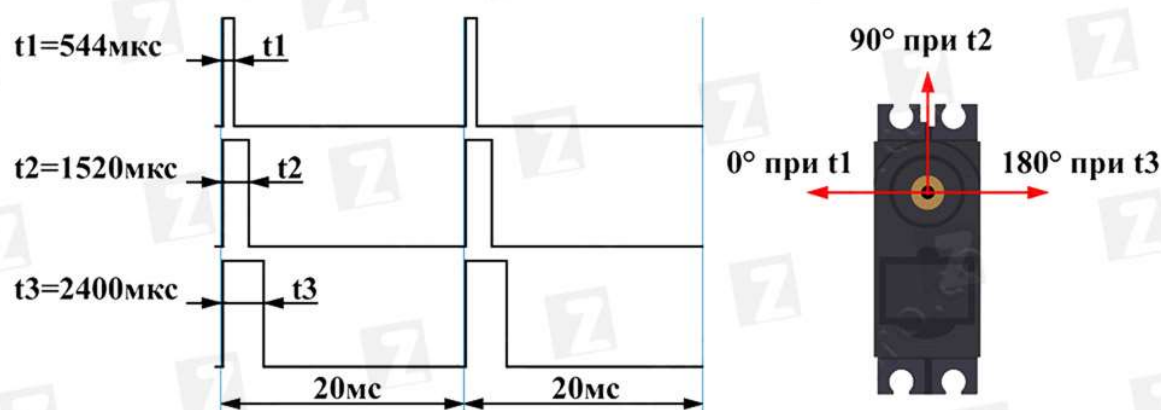
Серводвигатель отличается от обычного двигателя тем, что его можно всегда точно повернуть на определенный угол. Данная модель сервопривода, поворачивается на 180 градусов, у разных серводвигателей разные углы поворотов. так же существуют серводвигатели поворачивающиеся на 360 градусов. Серводвигатели так же иногда называют сервомоторами, сервомашинками или просто сервами. Не важно как его называть, главное что если в слове есть корень серво, то речь идет именно о таких двигателях. Давайте рассмотрим как устроен внутри такой двигатель и как он работает.

На картинке ниже указано наглядное устройство серводвигателя: внутри пластикового корпуса расположен двигатель этот двигатель быстро вращается, на его валу сидит шестерня, которая соединяясь с другими шестернями образует редуктор, который в итоге уменьшает скорость двигателя, вращается двигатель быстро но скорость вала в итоге медленная, это сделано для того чтобы увеличивать мощность поворота вала, пусть вал и вращается медленно, но зато у него за счет этого больше силы. Двигатель припаяна к печатной плате, так же к ней припаяна и вся электроника которая управляет нашим двигателем. И самый главный элемент благодаря которому двигатель понимает на какой угол он повернулся - потенциометр, как видите одна шестерня подключена к потенциометру и при вращении двигателя потенциометр поворачивается вместе с валом, и сопротивление меняется, так в зависимости от сопротивления на потенциометре двигатель понимает что он повернулся на определенный угол и останавливается.

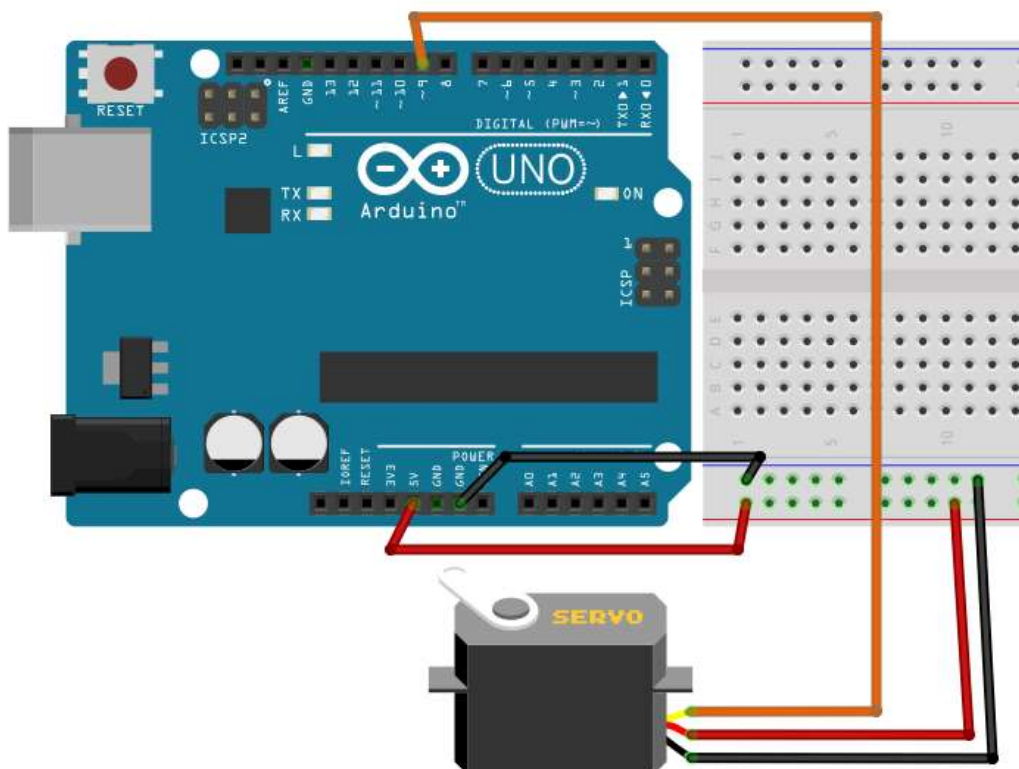


Теперь давайте наконец рассмотрим как управлять серводвигателем. У всех серводвигателей обычно 3 контакта, это плюс, минус и управляющий. Ну плюс с минусом и так ясны, сюда подается питание для двигателя, в зависимости от того какое напряжение ему необходимо, нашему серводвигателю достаточно 5 вольт получаемых от Arduino, он не сильно мощный. Плюс и минус обычно являются красным и черным проводом соответственно. Но просто подав питание на плюс и минус мы не заставим двигатель поворачиваться. Для этого и нужен третий, управляющий провод, он обычно обозначается желтым, оранжевым или белым цветом.

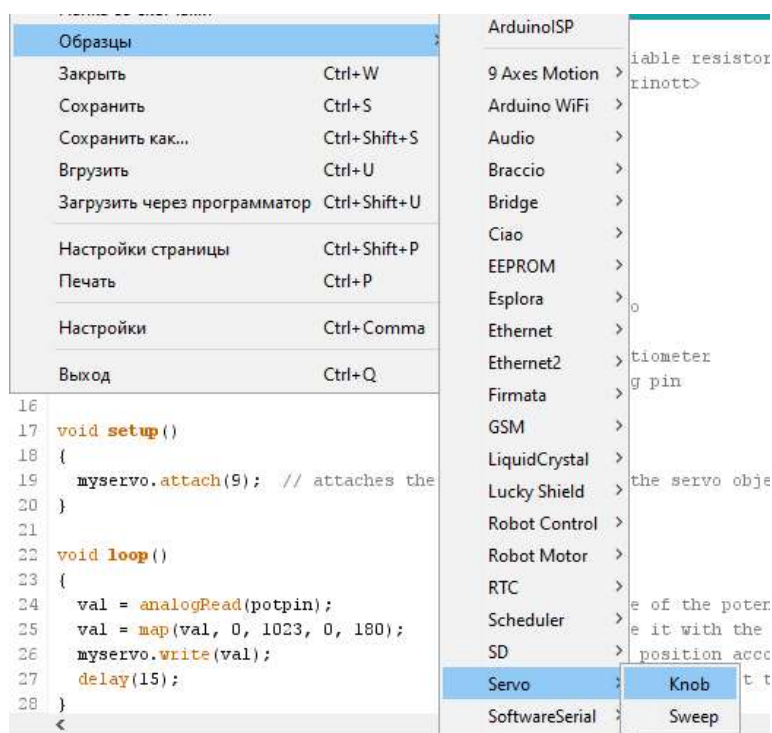
Для поворота серводвигателя используется шим порты. Угол поворота сервопривода зависит от длины импульса, посмотрите на изображение ниже, взятое с сайта www.zelectro.cc:



На изображении видно что чем дольше микросекунд будет длиться импульс тем больше угол поворота двигателя. в диапазоне примерно 500 - 2500 микросекунд. У разных серводвигателей могут отличаться диапазоны. И так, давайте подключим серводвигатель к нашей Arduino по такой вот простой схеме:



Мы не будем углубляться в пошаговое управление сервоприводом, а подключим библиотеку для управления им, для этого в Arduino уже встроена стандартная библиотека называемая Servo, в этой библиотеке есть пару примеров кода, давайте откроем и разберем их, откройте меню **Файл -> Образцы -> Servo -> Knob**



У нас откроется пример кода для работы с библиотекой Servo. И так, что собственно этот код делает, он поворачивает наш сервопривод на определенный угол в зависимости от

получаемого значение с аналогового парта A0, он преобразует с помощью функции `map` числа от 0 до 1023 в числа от 0 до 180 (угол поворота в градусах). В самой же библиотеке есть функция **`myservo.write()`**; эта функция поворачивает наш сервопривод на угол, который мы зададим в ее параметрах, например команда **`myservo.write(90)`**; повернет наш сервопривод на 90 градусов. Команда **`Servo myservo`**; инициализирует библиотеку `Servo`, команда **`#include <Servo.h>`** подключает нашу библиотеку. Всё. Это и все команды, этого абсолютно достаточно, для того, чтобы управлять сервоприводом. Ничего сложного правда? Теперь практика, выполните следующие задания:

1. *Сделайте так, чтобы сервопривод поворачивался от поворота потенциометра.*
2. *Сделайте, чтобы голова робота отворачивалась в другую сторону когда к дальномеру подносишь предмет.*
3. *Теперь давайте сделаем радар, мы будем вращать головой робота и в случае обнаружения дальномером препятствия в радиусе метра будем издавать короткие сигналы динамиком.*
4. *Теперь из полученного кода сделайте одну или несколько функций, чтобы в дальнейшем мы могли их использовать.*

Урок №12

Тема: Проекты с серводвигателем

На этом уроке у нас минимум теории максимум практики, мы продолжаем работать с серводвигателями. Но для начала хочется рассказать об одной интересной функции. Называется она **random()** эта функция сделана для генерации случайных чисел и у нее есть 2 параметра это начальное и конечное значение чисел среди которых она должна выбрать случайное значение. Например следующий код, то на экран будут выводиться случайные числа от 0 до 10, потому что, мы указали в параметрах числа от 0 до 10 **random(0, 10)**.

```
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  Serial.println(random(01,0));  
}
```

В параметрах мы можем указывать любые числа в пределах **long int**. Например **random(3852, 2000000)**. Мы так же можем вызывать функцию **random()** с одним параметром, например **random(777)** если мы так сделаем, то функция вернет любое число от 0 до 777, то-есть когда мы не указываем один параметр, то функция автоматически ставит вместо него 0. Это и вся теория на сегодня.

Теперь приступим к практике:

1. *Сделайте чтобы голова нашего робота поворачивалась на случайный угол в пределах от 0 до 180 градусов, используйте для этого функцию **random()**.*
2. *Сделайте шуточную программу “Правда или Ложь”, для этого сделайте чтобы при нажатии кнопки программа генерировала случайное число от 0 до 1, и если в результате получился 0 то пусть робот 3 раза машет головой (это будет означать Ложь), а если получается единица то пусть робот пищит динамиком (Это будет означать Правду). Вам необходимо задавать роботу вопрос и нажимать кнопку, а он будет указывать правду вы говорите или нет. Конечно же он будет это делать случайным образом, но ведь никто, кроме вас этого не знает.*
3. *Теперь сделайте чтобы робот постоянно ехал, а голова смотрела прямо, если вблизи появилось препятствие, то необходимо остановиться, повернуть головой в одну сторону, если там нет препятствия, то развернуться и ехать туда, если там тоже препятствие то повернуть голову в другую сторону и ехать туда, если и там у нас препятствие, то сдать назад и еще раз посмотреть по сторонам. и повторить действия.*

Урок №13

Тема: Проекты на закрепление

Сегодня у нас опять минимум теории, максимум практики. Давайте вспомним второй урок в котором мы проходили работу с последовательным портом. Думаю команды `Serial.print` и `Serial.println` вы хорошо запомнили, а вот `Serial.read()` и `Serial.write()` уже позабыли. И так вспоминаем. Функция `Serial.read()` считывает данные с последовательного порта, которые мы передадим с компьютера. Но перед тем как начать выполнять функцию `Serial.read()`, необходимо проверить передаются ли какие нибудь данные или нет, для этого существует функция `Serial.available()`, если эта функция возвращает число больше 0, тогда данные передаются, если нет то нет. Значит чтобы считать данные с компьютера, достаточно выполнить следующий код:

```
void setup() {  
    Serial.begin(9600); //Открываем последовательный порт  
}  
  
void loop() {  
    if (Serial.available() > 0) { //Проверяем передаются ли данные  
        Serial.read(); //Считываем переданный символ  
    }  
}
```

Значит мы сначала проверяем передаются ли данные, если передаются, то функция `Serial.read()` считывает один символ. Если мы передали не один символ, а 2 и больше, то функция `Serial.read()` все равно считывает один символ, в следующий цикл выполняется еще раз и считывает другого символ и так далее, пока не считает все символы которые мы передали. Функция `Serial.read()` не просто считывает символ, она еще и возвращает его. То есть мы можем написать например такое условие:

```
if(Serial.read() == 'Z'){ //Если мы считали символ Z  
    digitalWrite(2, HIGH); //Подать на второй порт - 5 вольт  
}
```

То есть мы можем проверять данные которые передались и делать от этого различные действия, например зажечь светодиод если передалась буква 'Z'. Обратите внимание, что символ, который проверяется заключен не в двойные кавычки "", а в апострофы '. Мы так же можем сохранить считанный символ в переменную. Но какой же должен быть тип переменной? Ведь полученный символ у нас не число. Для символов существует специальный тип переменных. Называется он **char** (с английского - символ). В переменную типа `char` мы можем сохранить один символ. Давайте попробуем:

```

char box;
void setup() {
    Serial.begin(9600); //Открываем последовательный порт
}

void loop() {
    if (Serial.available() > 0) { //Проверяем передаются ли данные
        box = Serial.read(); //Считываем переданный символ и сохраняем в box
        if(box == 'Z'){ //Если в переменной символ Z
            digitalWrite(2, HIGH); //Подать на второй порт - 5 вольт
        }
    }
}

```

Таким образом мы можем сохранять и обрабатывать символы которые получили. На этом всё на сегодня с последовательным портом закончим.

Еще одна небольшая функция которую вам полезно будет знать это **millis()** эта функция возвращает количество миллисекунд с начала запуска или перезагрузки Arduino. Она возвращает миллисекунды, и чем дольше работает Arduino тем больше будет число возвращаемое ей, максимальный предел числа это **unsigned long**, из третьего урока мы знаем, что максимальное значение у этого типа данных составляет 4 294 967 295. Этого хватит для подсчета миллисекунд в течении 49.7 дней. После этого функция **millis()** начнет отсчет заново. Для наших проектов, таких больших чисел вполне достаточно. С помощью этой функции мы можем сделать секундомер. Например в одну переменную мы можем сохранить текущие миллисекунды когда стартуем секундомер, а тогда когда мы захотим узнать сколько миллисекунд прошло, то мы просто отнимем от текущего значения **millis()** - то, которое сохранили. Например мы запустили наш секундомер когда Arduino проработала уже 20345 миллисекунд, мы сохраняем это значение, а потом когда мы остановим наш секундомер, предположим Arduino проработает уже 25350 миллисекунду, тогда мы вычтем 25350 - 20345 ,то получим 5005 миллисекунд, это значит наш секундомер насчитал 5 целых секунд и еще 5 миллисекунд. Например:

```

unsigned long start; //В эту переменную будем сохранять миллисекунды
void setup() {
    Serial.begin(9600);
    start = millis(); // Засекаем миллисекунды при старте Arduino
}

void loop() {
    Serial.println( millis() - start ); //Выводим сколько прошло времени после сохранения
    // в переменную start
}

```

Вот такая простая программа, будет в монитор последовательного порта, выводить сколько прошло времени, после выполнения функции Setup(). Так же в Arduino есть еще одна функция, называется она **micros()**, как можно догадаться, она считает микросекунды от

запуска Arduino, напоминаем что в одной секунде - миллион микросекунд. Эта функция тоже возвращает число, типа **unsigned long**, но так как здесь числа будут в 1000 раз больше чем у функции **millis()** то этой функции хватит для подсчета времени всего на 71.5 минут работы, после этого она обнулится и начнет отсчет заново. По теории на сегодня всё.

Выполните следующие задания:

1. *Сделайте включение и выключение светодиода с помощью отправки данных в Serial порт. Например, чтобы при отправке буквы 'A' светодиод включался, а при отправке 'a' - светодиод выключался.*
2. *Сделайте поворот серводвигателя в зависимости от цифры (от 0 до 9) отправленной в Serial порт с компьютера, например, если мы отправим 0, то двигатель повернется на 0 градусов, отправим 9 - повернется на 180 градусов и т.д. Используйте для этого функцию **map()**.*
3. *Сделайте устройство определяющее скорость реакции. Устройство должно состоять из динамика и кнопки. Устройство запускается нажатии на кнопку. Далее через произвольный промежуток времени звучит динамик и как только вы слышите динамик, то должны нажать кнопку. Когда начал пищать динамик, Arduino засекает время и как только вы нажали кнопку, Arduino выдает результат с количеством времени затраченным на реакцию. После этого программа ждет опять нажатие кнопки, чтобы начать заново.*
4. *Усложним задачу, теперь сделайте так, чтобы программа определяла скорость реакции двух человек, теперь нам понадобится 2 кнопки. Нажатием одной из кнопок мы так же будем запускать программу, далее если прозвучал динамик, то начинаем отсчет времени. Как только нажали одну из кнопок сохраняем ее время, как только нажали вторую из кнопок, сохраняем второе время. После этого сравниваем результаты и пишем кто победил и какая скорость реакции.*

Урок №14

Тема: Знакомство с bluetooth модулем

Наконец мы дошли до bluetooth модуля. К Arduino возможно подключение bluetooth модуля для передачи данных с устройств и на устройства поддерживающие технологию bluetooth, это может быть ноутбук, телефон или любое другое устройство. Мы будем рассматривать работу с модулем HC-06.

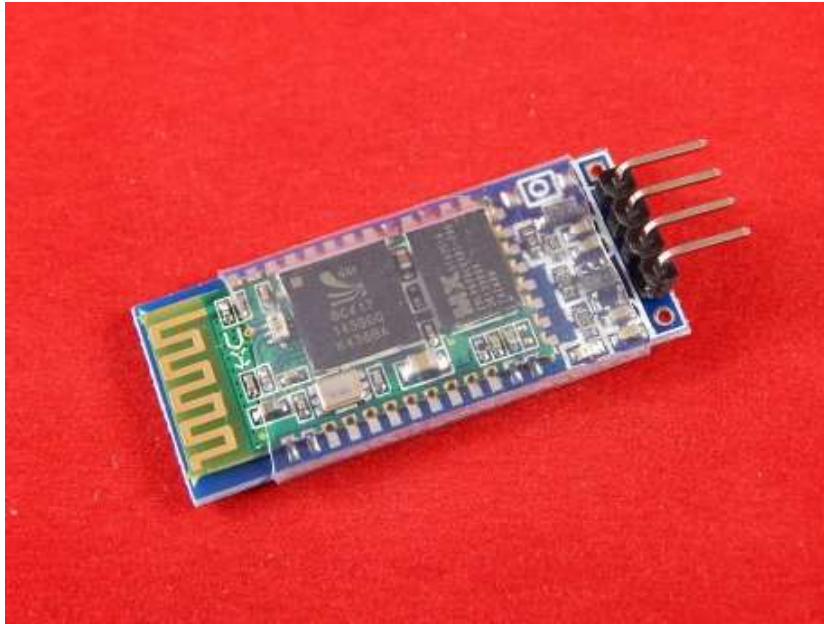
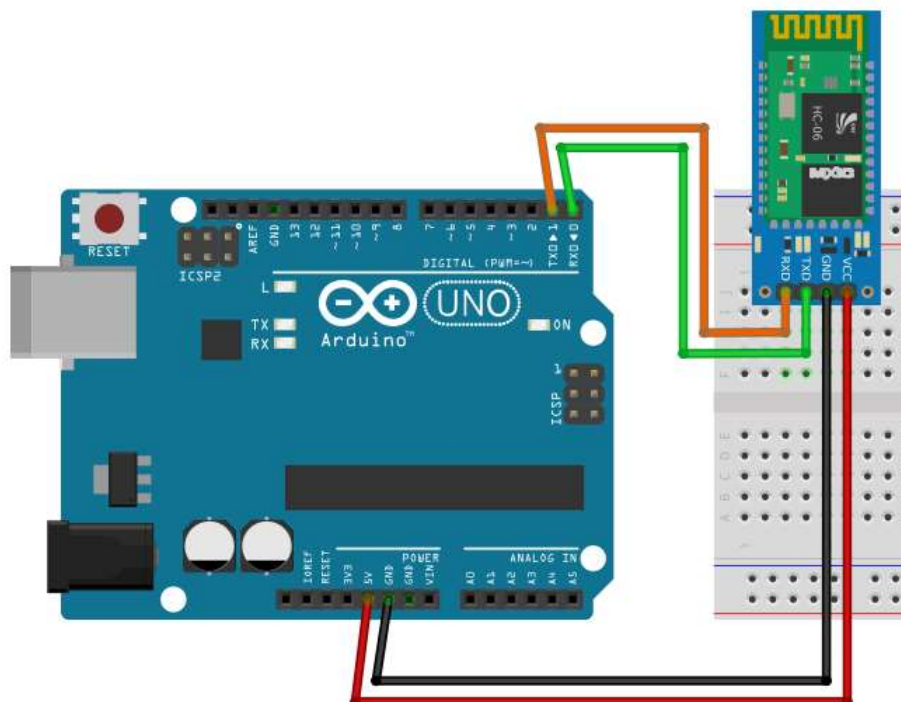


Схема его подключения довольно проста:



Как мы видим у нашего модуля есть 4 контакта, это VCC, GND, RX и TX. Ну с контактами GND и VCC все просто, к ним подключается земля и +5V соответственно.

Контакты RX и TX как раз сделаны для того, чтобы передавать и получать информацию. На нашем модуле контакт RX используется для чтения данных полученных с Arduino, а контакт TX для передачи данных на Arduino.

У нашей платы Arduino тоже есть такие контакты находятся они на цифровых портах 0 и 1. Задача у них точно такая же: RX читает данные, а TX передает. Соответственно чтобы Arduino смог читать данные надо к его RX подключить ножку TX от модуля bluetooth. А чтобы модуль bluetooth мог читать данные с Arduino надо контакт TX от Arduino подключить к контакту RX на модуле bluetooth. Схема подключения получается такая:

Контакты Arduino	Контакты модуля HC-06
5V	VCC
GND	GND
RX (цифровой порт 0)	TX
TX (цифровой порт 1)	RX

Чтобы лучше запомнить какая ножка передает данные, а какая принимает, знайте что буква T (Tx) - это сокращение Transceive с английского отправление, а буква R (Rx) - сокращение Receive с английского получение или прием, можно ее еще запомнить как Read - чтение. И так модуль мы подключили, как теперь передать данные?

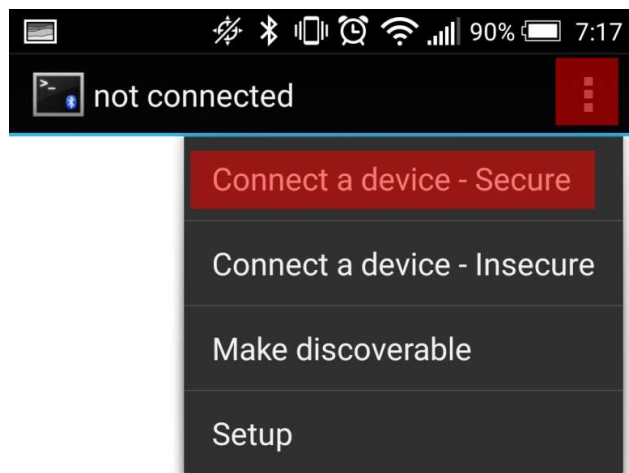
Давайте рассмотрим это на примере передачи данных с телефона и на телефон, на котором установлена ОС Android. Для начала необходимо зайти в Google Play Market и скачать какойнибудь bluetooth терминал, их большое множество и они мало чем отличаются друг от друга. Мы будем рассматривать терминал от разработчика Qwerty. Скачать его можно по следующей ссылке <https://play.google.com/store/apps/details?id=Qwerty.BluetoothTerminal>

И так, теперь подключите наш модуль HC-06 к Arduino. Подайте на Arduino питание, включите bluetooth на вашем смартфоне и откройте только что установленное приложение bluetooth терминал.

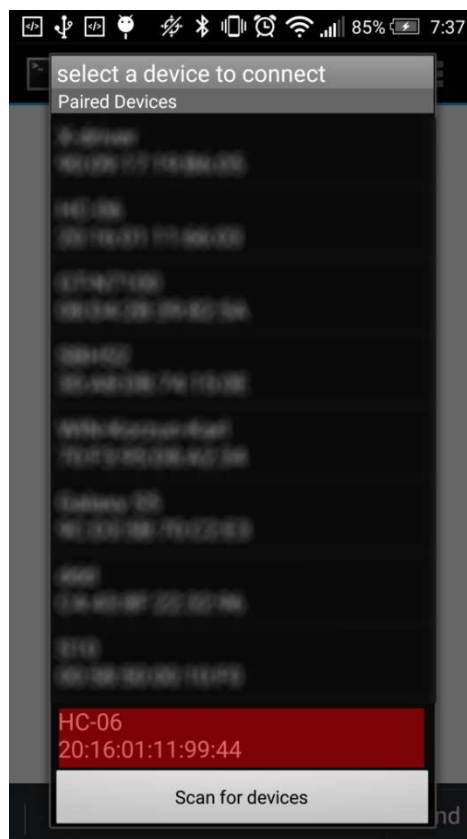
По умолчанию у вас в верхней строке будет надпись not connected,



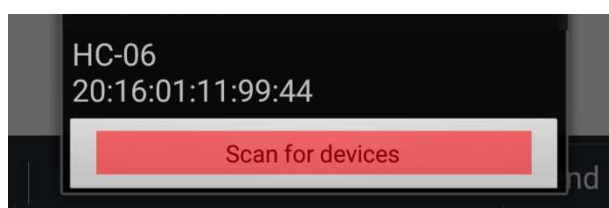
нажмите на меню и выберите там пункт **Connect a device - Secure** (подключится к устройству с паролем), это нужно для того чтобы теперь подключится к нашему модулю bluetooth.



После нажатия на пункт меню у вас откроется окно со списком bluetooth устройств по близости, вам необходимо найти в списке **HC-06** и выбрать его.



Если в списке его нет, то нажмите на кнопку **Scan for devices** (поиск устройств)



Когда в списке появится наш модуль **HC-06** можно нажать на него чтобы подключиться, если модуль не удастся найти, то проверьте правильно ли подключен модуль к Arduino, при

включении у него должен гореть светодиод. Попробуйте отключить и включить питание Arduino, выключите и включите bluetooth на телефоне.

При подключении к нашему модулю у вас будет запрошен пароль, **стандартным паролем** от модуля **НС-06** является **1234**.

После успешного подключения верхняя строка у вас поменяется на надпись **connected**.

Теперь нам всего лишь осталось написать код который будет передавать или получать данные с Arduino.

Делать это очень просто, вам не нужны другие библиотеки и вы даже знаете абсолютно все команды для этого, это те же команды, которые используются для передачи данных через Serial порт.

То есть для того чтобы передать какой-то текст на телефон, вам достаточно написать:

```
void setup() {  
  Serial.begin(9600); //Открываем последовательный порт  
}  
  
void loop() {  
  Serial.println("Text"); //Отсылаем слово Text с переносом строки  
}
```

Чтобы считать символы полученные с телефона достаточно написать:

```
char box;  
void setup() {  
  Serial.begin(9600); //Открываем последовательный порт  
}  
  
void loop() {  
  if (Serial.available() > 0) { //Проверяем передаются ли данные  
    box = Serial.read(); //Считываем переданный символ и сохраняем в box  
  }  
}
```

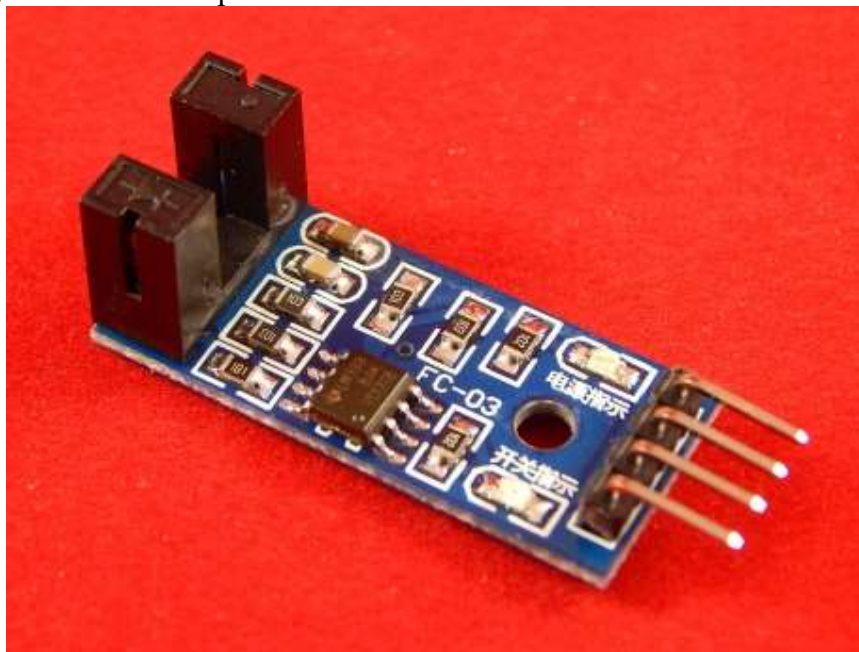
Вот собственно и всё, всё это мы проходили на прошлых уроках. И так, теперь давайте попрактикуемся и выполним следующие задания:

1. *Сделайте программу для включения и выключения светодиода с телефона.*
2. *Сделайте программу для управления тональностью динамика, чтобы можно было задать высоту звука от 0 до 9, где 0 - динамик вообще не пищит, а 9 - он издает звук на максимальной высоте.*
3. *Подключите датчик температуры и сделайте так, чтобы температура в комнате, передавалась на телефон.*
4. *Ну и наконец сделайте чтобы при отправке на Arduino буквы F наш робот ехал вперед, при отправке буквы B - назад, при отправке буквы S - останавливался.*

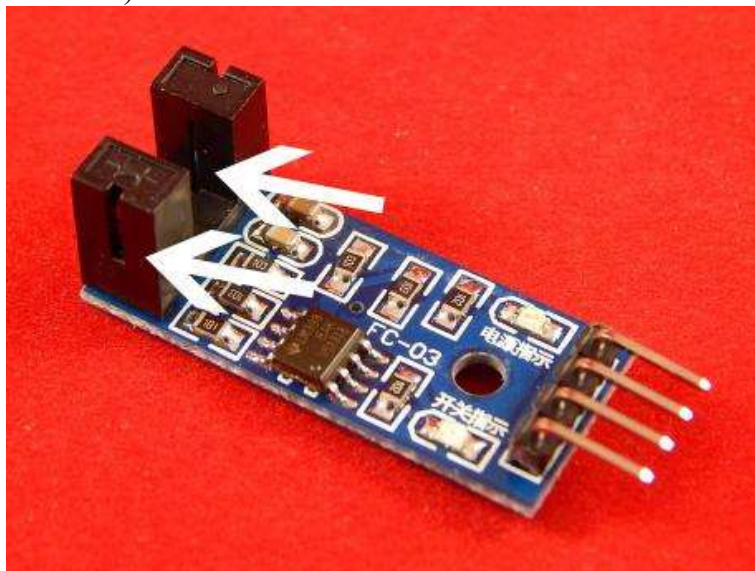
Урок №15

Тема: Знакомство с цифровым тахометром

Сегодня мы научимся определять и корректировать скорость вращения колес с помощью цифрового модульного тахометра FC-03:



Принцип работы данного тахометра очень прост, с двух сторон него расположены пластиковые крепления внутри которых находится источник света (светодиод) и фотоприемник (датчик света):



Светодиод через тонкую щель светит на фотоприемник, на самом тахометре есть зеленый светодиод который говорит о том, что свет попадает на фотоприемник. Если вдруг между источником света и фотоприемником расположить какой нибудь предмет, который закроет свет, то зеленый светодиод погаснет. Если предмет убрать то свет от источника будет опять попадать на датчик и зеленый светодиод опять загорится.

У этого тахометра есть 4 ножки, их назначение указано в таблице ниже:

Название порта тахометра:	Назначение:
VCC	Питание: +5V
GND	Питание: GND
D0	Сигнальный цифровой порт
A0	Сигнальный аналоговый порт

Как вы заметили у тахометра есть порты D0 и A0, когда свет попадает на датчик, то на порту D0 будет низкий уровень напряжением 0V - LOW, а если свет прервался то на порт D0 подается высокий уровень напряжением +5V - HIGH. Аналоговый порт A0 показывает более точное значение напряжения. Чем ярче будет попадать на датчик свет, тем больше меньше будет напряжение и наоборот. Если вы заметили то принцип работы цифрового тахометра очень похож на работу датчика линии.

Но как же нам определять скорость вращения колеса. Для этого нам понадобится диск с отверстиями, наподобие такого:



Этот диск мы закрепим на оси двигателя, а тахометр расположим так, чтобы диск находился между фотоприемником и источником света. Теперь если двигатель будет вращать диск, то благодаря отверстиям на диске, свет в тахометре будет то прерываться, то возобновляться. Соответственно на порт D0 то будет подаваться 5 и 0 вольт по очереди, и чем быстрее будет крутиться диск тем быстрее будут меняться значения. У этого диска 20 отверстий, это означает что за один оборот колеса, на порт D0 напряжение 5 и 0 вольт будет подано 20 раз.

Если мы с помощью Arduino будем считать эти импульсы и засекают время между ними, то сможем посчитать сколько оборотов колесо делает за минуту или за час например соответственно узнаем скорость.

Давайте начнем с того, что будем считать обороты колеса. И так для начала подключим тахометр к ардуино и подключим контакт D0 к любому цифровому порту, контакт A0 подключать не нужно.

```
void setup() {
  Serial.begin(115200); //Открываем монитор последовательного порта
  pinMode(2, INPUT); //Устанавливаем порт на считывание
}
```

В данном случае контакт D0 подключен к цифровому порту D2 на Arduino. Последовательный порт мы открыли на скорости 115200 бод, для того чтобы не замедлять работу Arduino, это важно, ведь мы будем измерять время пока быстро вращается диск. Теперь давайте сделаем счетчик импульсов тахометра. Для этого сначала создадим переменную в которую мы будем сохранять числа в те моменты когда мимо тахометра проходит очередное отверстие. Также нам необходимо сделать переменную-переключатель, в ней при появлении отверстия мы будем сохранять например 1, и при появлении препятствия 0. Это нужно для того, чтобы на одном и том же отверстии счетчик не сработал больше одного раза:

```
int counter=0; //счетчик оборотов
byte impulse=0; //переключатель отверстие|препятствие
```

Теперь нам необходимо считывать сигнал с порта и при появлении там низкого уровня - LOW увеличивать счётчик и присваивать в нашу переменную переключатель impulse=1. А при появлении препятствия на порт будет подаваться сигнал HIGH, мы будем это узнавать и переключать нашу переменную обратно на 0:

```
void loop() {
  //Если на порт подается LOW и в переключателе хранится 0
  if(digitalRead(2)==0 && impulse==0){
    //Увеличиваем счетчик
    counter++;
    //В переключатель сохраняем 1, чтобы больше счетчик не сработал
    impulse=1;
    //Если на порт подается 1, значит появилось препятствие
  } else if(digitalRead(2)==1 && impulse==1){
    //Обнуляем наш переключатель, чтобы при следующем отверстии
    //счетчик сработал
    impulse=0;
  }
}
```

Теперь наш счетчик работает, если в счетчике появится число 20, значит наше колесо сделало 1 оборот. Мы можем запомнить время с помощью функции millis() или для большей точности с помощью функции micros() в тот момент когда счетчик равен 0. И сравнить его после того как в счетчике появится число 20, а потом обнулить наш счётчик. Сравнив время мы узнаем сколько было затрачено миллисекунд на 1 оборот. Для этого нам понадобится переменная в которую мы будем сохранять время для сравнения:

```
unsigned long time1Turn; //Сюда сохраним время 1 оборота
```

Теперь допишем код засекающий время:

```

//Если счетчик равен 0
if(counter==0){
  //Сохраняем время
  time1Turn = millis();
}
//Если счетчик равен 20
if(counter==20){
  //Отнимаем от текущего времени - сохраненное
  //и выводим на экран
  Serial.println(millis()-time1Turn);
  //Обнуляем счетчик
  counter=0;
}

```

Теперь мы знаем сколько миллисекунд затрачено на 1 оборот, а так как в минуте у нас 60000 миллисекунд, то мы можем узнать сколько оборотов будет производиться в минуту, для этого создадим переменную, в которую будем сохранять значение:

```
float RPM=0; //Число оборотов в минуту
```

Теперь мы можем рассчитать число оборотов:

```

//Если счетчик равен 20
if(counter==20){
  //Отнимаем от текущего времени - сохраненное
  //Число миллисекунд в секунде делим на наше
  //засеченное время
  RPM = 60000/(millis()-time1Turn);
  //и выводим на экран
  Serial.println(RPM);
  //Обнуляем счетчик
  counter=0;
}

```

Теперь на экран у нас выводится число оборотов в минуту, производимое нашим колесом. Давайте сделаем еще, чтобы выводилось число оборотов в час, для этого нам нужно разделить число миллисекунд в часу на время затраченное на 1 оборот. В одной минуте у нас 60000 миллисекунд, в одном часу 60 минут, значит мы умножаем $60 \cdot 60000$ и получаем 3600000 миллисекунд в одном часу, создаем переменную в которой будут храниться обороты/час:

```
float RPH=0; //Число оборотов в час
```

Рассчитываем:

```
//Вычисляем обороты в час
```

```
RPH = 3600000/(millis()-time1Turn);  
//и выводим на экран  
Serial.println(RPH);
```

Теперь мы знаем число оборотов колеса в час, но как же на узнать скорость? Для этого нам нужно знать какое расстояние за 1 оборот колесо проезжает. Колесо за один оборот проезжает расстояние равное длине окружности этого колеса. На как узнать длину окружности? Ну тут 2 варианта, либо измерить, либо те, кто изучал математику в 5 классе знает, что длина окружности равна числу ПИ **3.14** умноженному на диаметр колеса. Диаметр колеса у машинки UnoCraft составляет примерно 70мм, то есть 7см:



Значит получается, что расстояние которое проезжает такое колесо за 1 оборот, составляет **7*3,14=21,98см** если мы хотим узнать расстояние в метра, то это значение нужно разделить на 100, так как в метре 100 сантиметров, если мы хотим узнать расстояние в километрах, то нам нужно разделить это значение на 100 000, так как в километре 100 000 метров. Теперь зная число оборотов в час мы можем рассчитать нашу скорость в километрах в час:

```
//Вычисляем скорость в КМ/Ч и выводим на экран  
Serial.println(RPM*7*3.14/100000.0);
```

Весь код целиком выглядит так:

```
int counter=0; //счетчик оборотов  
byte impulse=0; //переключатель отверстие|препятствие  
unsigned long time1Turn; //Сюда сохраним время 1 оборота  
float RPM=0; //Число оборотов в минуту  
float RPH=0; //Число оборотов в час  
void setup() {  
  Serial.begin(115200); //Открываем монитор последовательного порта  
  pinMode(2, INPUT); //Устанавливаем порт на считывание  
}  
  
void loop() {  
  //Если на порт подается LOW и в переключателе хранится 0
```

```

if(digitalRead(2)==0 && impulse==0){
  //Увеличиваем счетчик
  counter++;
  //В переключатель сохраняем 1, чтобы больше счетчик не сработал
  impulse=1;
  //Если на порт подается 1, значит появилось препятствие
} else if(digitalRead(2)==1 && impulse==1){
  //Обнуляем наш переключатель, чтобы при следующем отверстии
  //счетчик сработал
  impulse=0;
}
//Если счетчик равен 0
if(counter==0){
  //Сохраняем время
  time1Turn = millis();
}
//Если счетчик равен 20
if(counter==20){
  //Отнимаем от текущего времени - сохраненное
  //Число миллисекунд в секунде делим на наше
  //засеченное время
  RPM = 60000/(millis()-time1Turn);
  //Вычисляем обороты в час
  RPH = 3600000/(millis()-time1Turn);
  //и выводим на экран
  //Serial.println(RPM);
  //Вычисляем скорость в КМ/Ч и выводим на экран
  Serial.println(RPH*7*3.14/100000.0);
  //Обнуляем счетчик
  counter=0;
}
}
}

```

Вот мы и научились вычислять скорость наших колес.

Перейдем к выполнению заданий:

1. Переделайте код так, чтобы время засекалось в микросекундах а не миллисекундах, для большей точности.
2. Подключите оба колеса и сделайте чтобы скорость каждого выводилась в программу *bluetooth electronics*
3. Сделайте так, что если наши колеса крутятся с разной скоростью, то скорость более быстрого колеса замедлялась во столько же раз во сколько, оно крутится медленней чем другое. Либо увеличивалась, если есть куда увеличиваться.
4. Сделайте так чтобы, через программу *bluetooth electronics* мы могли задавать скорость вращения колеса от 0.5 км/ч, до 5 км/ч. То есть если мы выставяем в программе скорость 3км/ч, то и колеса крутятся с этой скоростью.