

## Лабораторная работа №1 “Основы работы с Arduino IDE”

1. Откройте программу Arduino IDE

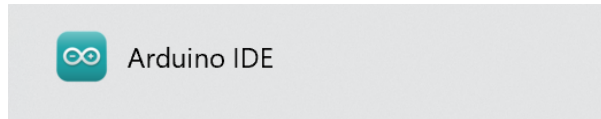


Рисунок 1 - Программа Arduino IDE в Пуск

2. Интерфейс программы изображен на рисунке 2.



Рисунок 2 - Интерфейс программы

1 - Компиляция скетча. Компиляция в программировании — это процесс преобразования исходного кода, написанного на языке высокого уровня (например, C++, Rust), в машинный код или другой низкоуровневый формат, понятный процессору. После компиляции в нижнем поле могут появиться ошибки связанные с синтаксисом.

2 - Загрузка программы в микроконтроллер. После нажатия компилирует и загружает программу, которую вы написали. После загрузки программа остается во внутренней энергонезависимой памяти микроконтроллера, то есть после выключения будет по новой воспроизводиться та программа, которую загрузили последней.

3 - Режим дебаггинга. Режим пошагового воспроизведения программы для отладки и поиска ошибок.

4 - Выбор платы. В данном окне необходимо выбрать тип платы и порт, к которому подключен микроконтроллер.

5 - Альбом, быстрого доступа к созданным скетчам.

6 - Менеджер плат. В данном менеджере можно выбирать и скачивать дополнительные библиотеки плат. После установки программы Arduino IDE есть только стандартные платы Arduino.

7 - Менеджер библиотек. В данном менеджере можно установить дополнительные библиотеки.

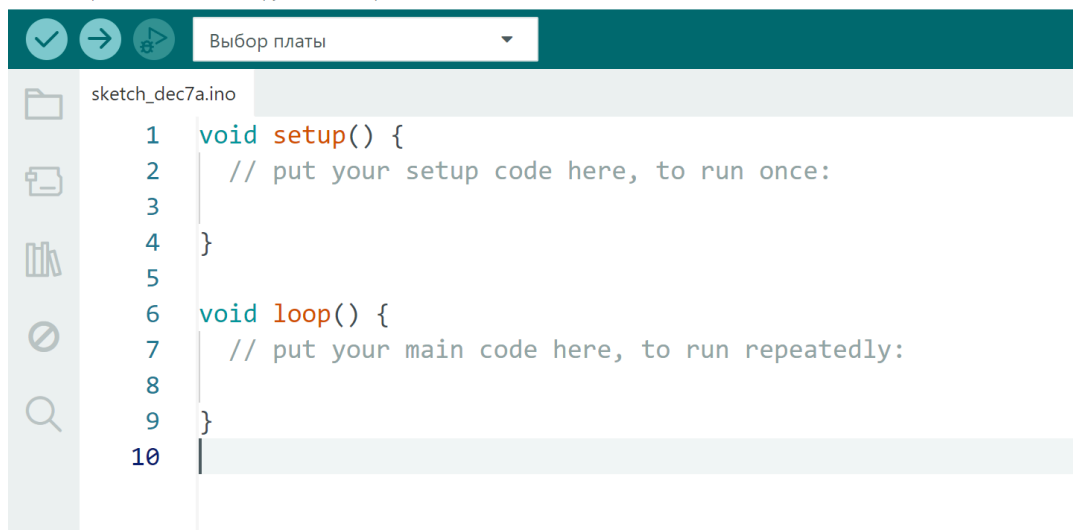
8 - Отладка.

9 - Поиск по коду

10 - Плоттер. Строит графики из полученных в монитор порта данных.

11 - Монитор порта. Монитор для общения компьютера с микроконтроллером. Сюда приходят данные с микроконтроллера по Serial порту.

3. Стандартная программа состоит из двух функций `void setup` и `void loop`.



```
1 void setup() {
2 // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8
9 }
10
```

Рисунок 3 - Стандартная программа

`void setup()` - функция, которая воспроизводится единожды после включения питания у микроконтроллера.

`void loop()` - функция, которая воспроизводится по кругу до тех пор, пока не выключится питание.

Важно отметить, что микроконтроллер всегда выполняет программу шаг за шагом, строчка за строчкой! Обращайте на это внимание при написании программы.

4. Немного про синтаксис языка C++. Имеется программа. Она реализовывается построчно. В самом начале до функций вызываются библиотеки, назначаются переменные и объекты.

Программы могут включать функции (подробнее - <https://arduino.ru/Reference/FunctionDeclaration>). В круглых скобках пишутся переменные, которые должны возвращаться после окончания работы функции. Внутри фигурных скобок пишутся команды, которые должны исполняться внутри функции. Окончание строки обозначается точкой с запятой - „ ; “

### Синтаксис функции

тип возвращаемого значения  
"void" если функция ничего не возвращает

Имя функции

Параметры, передаваемые в функцию

```
int myMultiplyFunction(int x, int y){  
  int result;  
  result = x * y;  
  return result;  
}
```

Оператор, возвращающий значение соответствующего типа

Фигурные скобки, обязательны

Рисунок 4 - Синтаксис

5. Элементы платы изображены на рисунке 5.

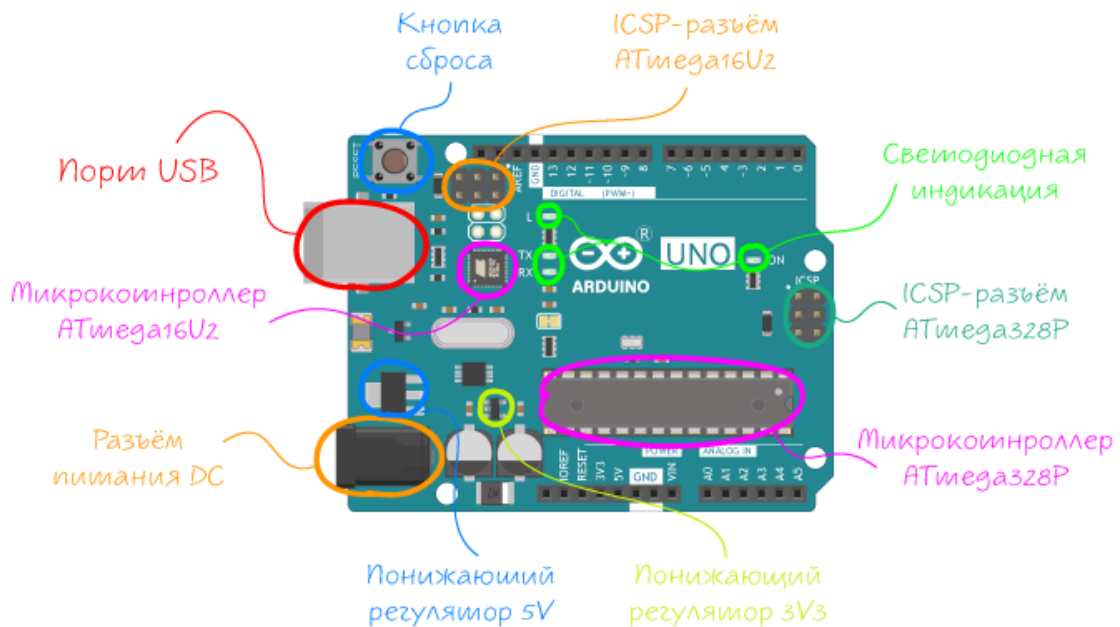


Рисунок 5 - Элементы платы

На каждую плату можно найти распиновку (Pin out) - это карта того где какие ножки расположены и какие возможности у этих ножек есть.

# UNO PINOUT

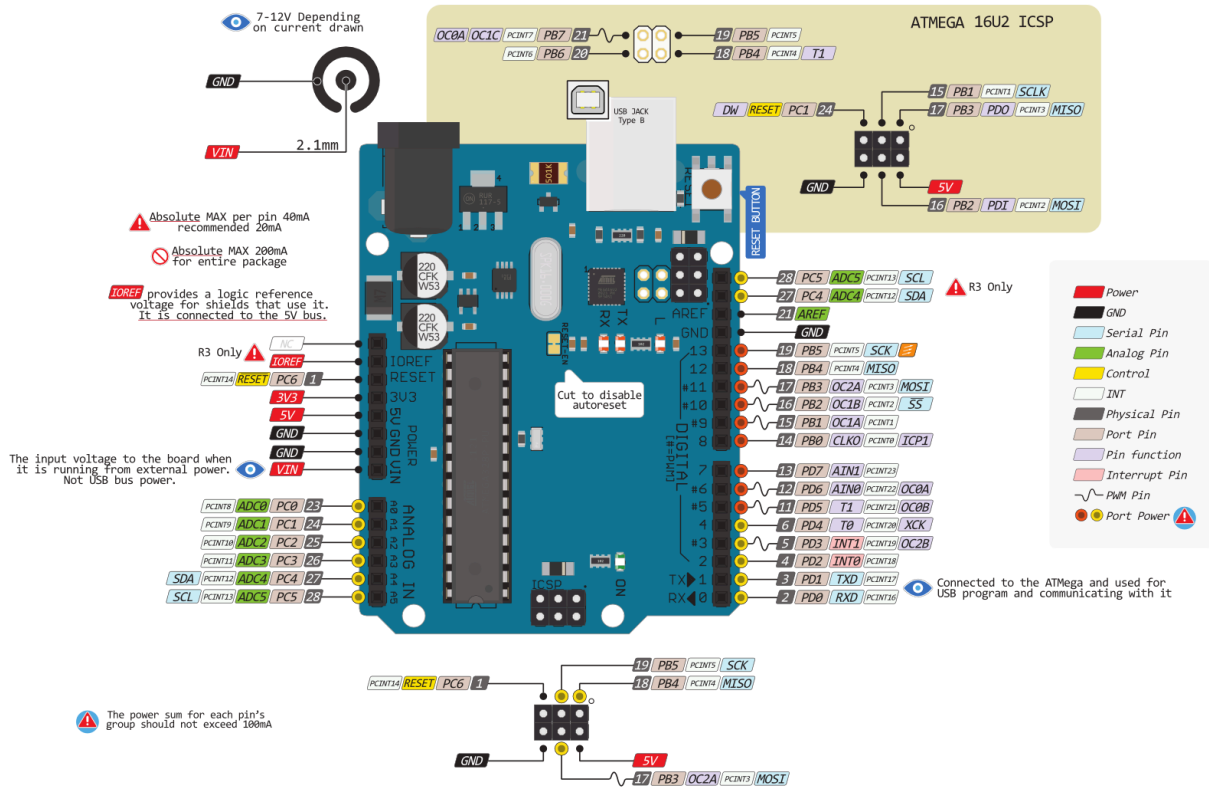


Рисунок 6 - Распиновка Arduino UNO

## Пины питания

- **VIN:** Входной пин для подключения внешнего источника напряжения в диапазоне от 7 до 12 вольт.
- **5V:** Выходной пин от стабилизатора напряжения с выходом 5 вольт и максимальным током 1 А. Регулятор обеспечивает питание микроконтроллера и другой обвязки платы.
- **3V3:** Выходной пин от стабилизатора напряжения с выходом 3,3 вольта и максимальным током 150 мА.
- **IOREF:** Вывод предоставляет платам расширения информацию о рабочем напряжении микроконтроллера. В нашем случае рабочее напряжение платформы 5 вольт.
- **AREF:** Пин для подключения внешнего опорного напряжения АЦП относительно которого происходят аналоговые измерения при

использовании функции `analogReference()` с параметром «EXTERNAL».

- **GND:** Выводы земли.

#### **Порты ввода/вывода**

- **Пины общего назначения:** 20 пинов: 0–19. Логический уровень единицы — 5 В, нуля — 0 В. К контактам подключены подтягивающие резисторы, которые по умолчанию выключены, но могут быть включены программно.
  - **АЦП:** 6 пинов: 14–19 / A0–A5. Позволяет представить аналоговое напряжение в цифровом виде. Разрядность АЦП не меняется и установлена в 10 бит. Диапазон входного напряжения от 0 до 5 В, при подаче большего напряжения микроконтроллер может выйти из строя.
  - **ШИМ (широтно-импульсная модуляция):** 6 пинов: 3, 5, 6 и 9–11. Позволяет выводить аналоговое напряжение в виде ШИМ-сигнала из цифровых значений. Разрядность ШИМ не меняется и установлена в 8 бит.
  - **I<sup>2</sup>C:** Для общения контроллера с платами расширения и сенсорами по интерфейсу I<sup>2</sup>C. Пины SDA/18/A4 и SCL/19/A5
  - **SPI:** Для общения контроллера с платами расширения и сенсорами по интерфейсу SPI. Пины MOSI/11, MISO/12 и SCK/13
  - **Serial/UART:** Для общения контроллера с платами расширения и сенсорами по интерфейсу UART. Пины TX1/1 и RX1/0. Контакты также соединены с соответствующими выводами сопроцессора ATmega16U2 для общения платы по USB. Во время прошивки и отладки программы через ПК, не используйте эти пины в своём проекте.
6. Для начала познакомимся с цифровыми сигналами, их поддерживают все порты GPIO. Цифровые сигналы могут выдавать

напряжение 0V и VCC (напряжение питания МК). GPIO не может измерить или выдать промежуточное напряжение между этими границами.

Логической единицей Arduino UNO считает напряжение от 5 до 3 вольт, а низкий - от 1,5 до 0 вольт.

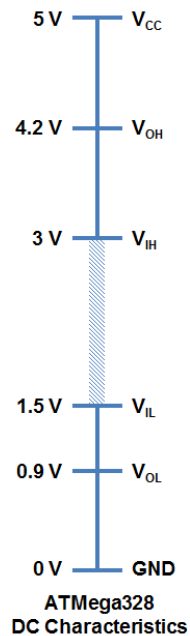


Рисунок 7 - Уровни напряжения

Пины GPIO могут работать в разных режимах - как минимум вход (чтение напряжения) и выход (выдача напряжения), в некоторых случаях это может быть только вход или только выход - см. документацию на конкретную плату.

В Arduino режим работы пина настраивается при помощи функции **pinMode(пин, режим)**:

- пин - цифра, номер пина GPIO, соответствует номеру на распиновке.  
На некоторых неофициальных платах подписи на плате не соответствуют номерам GPIO, в этом случае нужно смотреть распиновку или документацию на плату
- режим - режим работы пина, константа:
  - OUTPUT - выход
  - INPUT - вход (по умолчанию)

Так как это настройка, которая требуется единожды, то её располагают в `void setup()` внутри фигурных скобок.

Установить на выход высокий (HIGH) или низкий (LOW) уровень напряжения можно при помощи команды

**digitalWrite(пин, уровень), где:**

- пин: номер вход/выхода(pin)
- уровень: значение HIGH или LOW

А считать какой сейчас уровень напряжения приходит на вход можно при помощи команды

**digitalRead(пин)**

- пин: номер вход/выхода(pin)

## 7. ШИМ (широтно-импульсная модуляция)

Широтно-Импульсная модуляция, или ШИМ, это операция получения изменяющегося аналогового значения посредством цифровых устройств. Устройства используются для получения прямоугольных импульсов - сигнала, который постоянно переключается между максимальным и минимальным значениями. Данный сигнал моделирует напряжение между максимальным значением (5 В) и минимальным (0 В), изменяя при этом длительность времени включения 5 В относительно включения 0 В.

Длительность включения максимального значения называется шириной импульса. Для получения различных аналоговых величин изменяется ширина импульса. При достаточно быстрой смене периодов включения-выключения можно подавать постоянный сигнал между 0 и 5 В на светодиод, тем самым управляя яркостью его свечения.

На графике зеленые линии отмечают постоянные временные периоды. Длительность периода обратно пропорциональна частоте ШИМ. Т.е. если частота ШИМ составляет 500 Гц, то зеленые линии будут отмечать интервалы длительностью в 2 миллисекунды каждый. Вызов

функции `analogWrite()` с масштабом 0 – 255 означает, что значение `analogWrite(255)` будет соответствовать 100% рабочему циклу (постоянное включение 5 В), а значение `analogWrite(127)` – 50% рабочему циклу.

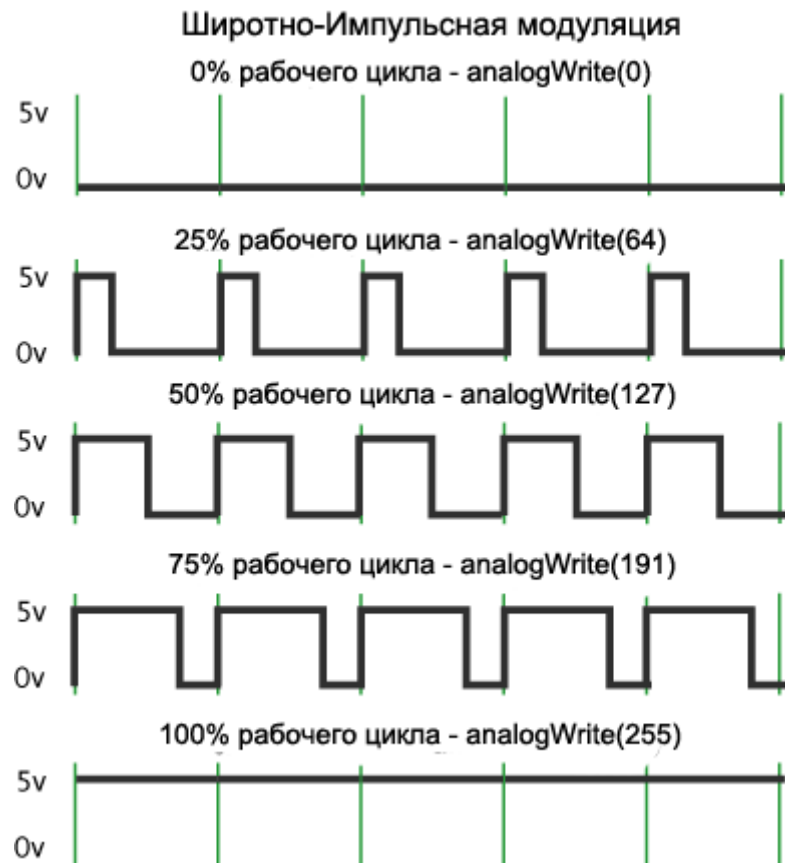


Рисунок 8 - ШИМ

### **Функция `analogWrite(пин, значение)`**

- пин: порт вход/выхода на который подаем ШИМ сигнал.
- значеник: период рабочего цикла значение между 0 (полностью выключено) and 255 (сигнал подан постоянно).

Выдает аналоговую величину (ШИМ волну) на порт вход/выхода. Функция может быть полезна для управления яркостью подключенного светодиода или скоростью электродвигателя. После вызова `analogWrite()` на выходе будет генерироваться постоянная прямоугольная волна с заданной шириной импульса до следующего вызова `analogWrite` (или вызова `digitalWrite` или `digitalRead` на том же порту вход/выхода). Частота ШИМ сигнала приблизительно 490 Hz.

На большинстве плат Arduino (на базе микроконтроллера ATmega168 или ATmega328) ШИМ поддерживают порты 3, 5, 6, 9, 10 и 11, на плате Arduino Mega порты с 2 по 13. На более ранних версиях плат Arduino `analogWrite()` работал только на портах 9, 10 и 11.

Для вызова `analogWrite()` нет необходимости устанавливать тип вход/выхода функцией `pinMode()`.

Функция `analogWrite` никак не связана с аналоговыми входами и с функцией `analogRead`.

#### 8. Команда задержки времени **delay(мс)**

Останавливает выполнение программы на заданное в параметре количество миллисекунд (1000 миллисекунд в 1 секунде).

9. Теперь разберемся как работают аналоговые пины. Некоторые микроконтроллеры имеют на борту АЦП - аналогово-цифровой преобразователь (ADC, Analog-to-Digital Converter). Это устройство, позволяющее измерять внешнее напряжение, преобразовывать его в численное значение и передавать в программу, по сути - вольтметр. АЦП имеет два важных параметра:

- Опорное напряжение - максимальное напряжение, которое можно подавать на АЦП. Минимальное - 0 (ноль), GND
- Разрешение (разрядность) - точность, с которой АЦП измеряет напряжение от 0 до опорного, измеряется в битах. Это буквально количество делений между 0 и опорным напряжением, например 10 бит =  $2^{10} = 1024$  значения, при опорном 5V получится цена деления  $5 / 1024 = 0.0049$  V

Само аналоговое напряжение может иметь бесконечно большое разрешение, то есть меняться с шагом например 0.00000001 В и меньше. АЦП оцифровывает напряжение с конечной точностью, то есть часть информации теряется. Непрерывный аналоговый сигнал превращается в дискретный цифровой.

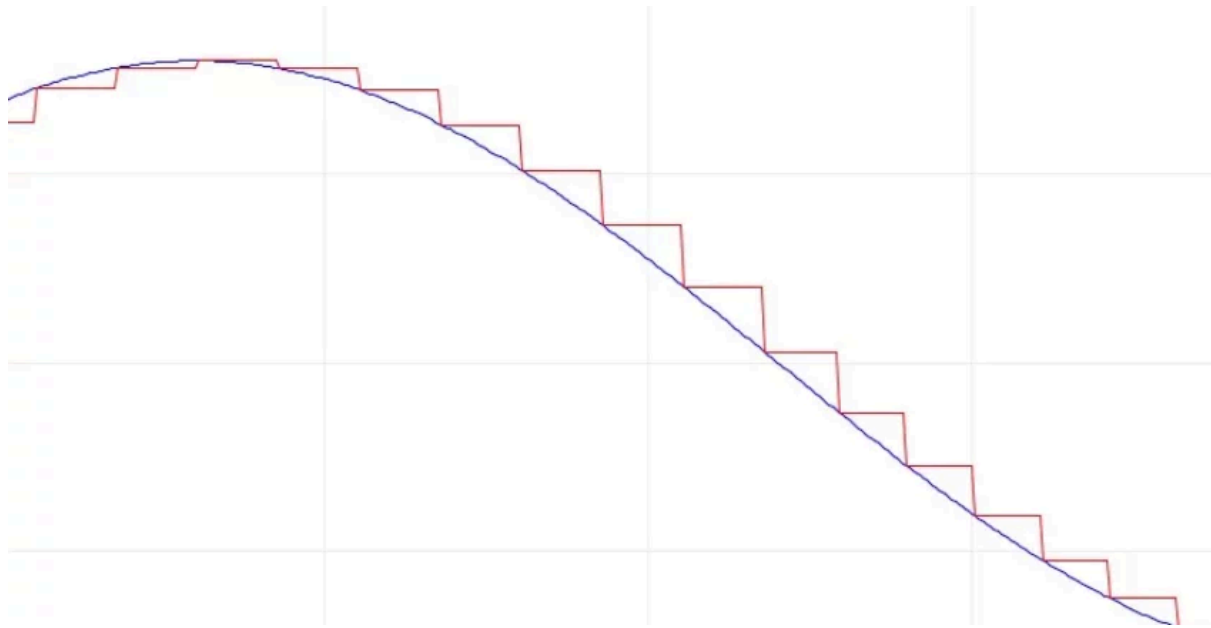


Рисунок 8 - Непрерывный аналоговый сигнал и как его видит АЦП

### **Функция `analogRead(пин)`**

Функция считывает значение с указанного аналогового входа. Большинство плат Arduino имеют 6 каналов (8 каналов у платы Mini и Nano, 16 у Mega) с 10-битным аналого-цифровым преобразователем (АЦП). Напряжение поданное на аналоговый вход, обычно от 0 до 5 вольт будет преобразовано в значение от 0 до 1023, это 1024 шага с разрешением 0.0049 Вольт. Разброс напряжение и шаг может быть изменен функцией `analogReference()`.

Считывание значение с аналогового входа занимает примерно 100 микросекунд (0.0001 сек), т.е. максимальная частота считывания приблизительно 10,000 раз в секунду.

## **10. Теперь немного о переменных и типах данных**

Переменная (`variable`) - ячейка памяти с заданным именем, по которому к ней можно обратиться для записи и чтения данных.

C/C++ - язык со статической типизацией. У переменной при создании указывается тип и она сохраняет его на всём протяжении своего существования. Если в переменную записать данные другого типа - они будут по возможности преобразованы к типу самой переменной, если это

невозможно - будет ошибка компиляции. В других языках, например Python или JavaScript, типизация динамическая, то есть в одну и ту же переменную можно записывать данные разных типов и она каждый раз будет менять свой тип на новый.

Переменная создаётся при помощи конструкции *тип\_данных имя*;, это действие называется определением переменной (definition):

```
int a;  
uint8_t b;  
float c;
```

В этот момент процессор выделяет память под размер указанного типа по следующему свободному адресу в памяти и даёт его нам под управление по указанному имени.

Можно создать несколько переменных одного типа, указав их через запятую:

```
int a, b, c;  
float d, e, f;
```

Тип данных в программировании — это классификация, которая определяет диапазон допустимых значений для данных и набор операций, которые можно с ними выполнять, а также способ их хранения в памяти, например, целые числа, текст (строки), логические значения (истина/ложь). Типы данных помогают компьютеру понимать, как интерпретировать последовательности битов и какие действия с ними производить (сложение, сравнение).

Основные типы данных приведены в таблице.

Название	Альтернативное название	Вес	Диапазон	Особенность
<b>boolean</b>		1 байт	0 или 1	Логическая переменная, может принимать значения <b>true (1)</b> и <b>false (0)</b>
<b>char</b>	int8_t	1 байт	-128... 127	Хранит номер символа из таблицы символов ASCII
<b>byte</b>	uint8_t	1 байт	0... 255	
<b>int</b>	int16_t	2 байта	-32 768... 32 767	
<b>unsigned int</b>	uint16_t	2 байта	0... 65 535	
<b>word</b>		2 байта	0... 65 535	То же самое, что unsigned int
<b>long</b>	int32_t	4 байта	-2 147 483 648... 2 147 483 647	- 2 миллиарда... 2 миллиарда
<b>unsigned long</b>	uint32_t	4 байта	0... 4 294 967 295	0... 4 миллиарда...
<b>float</b>		4 байта	-3.4028235E+38... 3.4028235E+38	Хранит числа с плавающей точкой (десятичные дроби). <b>Точность: 6-7 знаков</b>
<b>double</b>		4 байта		То же самое, что float

Глобальная переменная – объявляется **ВНЕ** функций, например в самом начале скетча или между функциями. Обращаться к глобальной переменной (использовать её значение) можно использовать **ВЕЗДЕ**.

Локальная переменная – объявляется **ВНУТРИ** функции, и обращаться к ней можно только внутри этой функции.

Локальных переменных может быть несколько с одинаковым именем, но разными значениями. Это связано с тем, что локальная переменная выгружается из оперативной памяти микроконтроллера при выходе из функции.

## 11. Условный оператор if

`if () {}` - условный оператор, проверяет истинность в `()` и выполняет код в `{}`, если оно верно

```
if () { // проверяет условие, если верно,
      // выполняет эту часть кода
} else { // если неверно
      // выполняет вот эту часть кода
```

```

}
if () { // проверяет условие, выполняет если верно
} else if () { // если неверно, проверяет новое
} else if () { // если неверно, проверяет новое
} else if () { // если неверно, проверяет новое
} else { // если неверно, выполняет то, что ниже
}

```

В условии может быть как логическое выражение ( $a > b$ ), так и логическая переменная со значением true или false. Или обычная переменная со значением 1 или 0.

### **Операторы сравнения**

- $a == b$  - если a равно b
- $a != b$  - если a не равно b
- $>$  - если a больше b (строго)
- $<$  - если a меньше b (строго)
- $>=$  - если a больше или равна b
- $<=$  - если a меньше или равна b

### **Логические операторы**

- $\&\&$  - логическое И (одно условие И второе)
- $\|\|$  - логическое ИЛИ (либо одно, либо второе)
- $!$  – отрицание (например  $\text{if} (!\text{val})$  - если val - ложь, т.е. 0)

### **Оператор выбора switch.. case**

```

switch (val) { // рассматриваем переменную val
    case 1: // если она равна 1, выполнить код здесь
        break;
    case 2: // если она равна 2, выполнить код здесь
        break;
    default: // если что-то ещё, выполнить код здесь (default
        необязателен)

```

```
break;  
}
```

### **Цикл for, «счётчик»**

for (counter; condition; change) {} - цикл for

- counter – переменная счётчика, обычно создают новую «локальную», в стиле `int i = 0;`
- condition – условие, при котором выполняется цикл, например «счётчик меньше 5» `i < 5;`
- change – изменение, т.е. увеличение или уменьшение счётчика, например `i++, i--, i += 10;`

Пример:

```
for (byte i = 0; i < 100; i++) { // счётчик от 0 до 99  
    Serial.println(i); // вывести в монитор порта числа от 0 до 99  
}
```

### **Цикл while, «с предусловием»**

while (condition) {}

- condition – условие, при котором выполняется блок кода, заключённый в {}

Пример:

```
while (flag) {  
    // какой-то кусок кода, который выполняется, пока flag равен  
    // логической 1  
}
```

### **Цикл do while, «с постусловием»**

do {} while (condition) ;

- condition – условие, при котором выполняется блок кода, заключённый в {}

Пример:

```
do {
```

```
// какой-то кусок кода, который выполняется, пока flag равен  
логической 1  
// но в отличие от предыдущего цикла, выполнится ХОТЯ БЫ ОДИН  
РАЗ, даже если Flag равен 0  
} while (flag);
```

**break – выход из цикла**

Пример:

```
for (byte i = 0; i < 100; i++) { // счётчик от 0 до 99  
if (i > 50) break; // выйти из цикла, если i больше 50  
//  
Serial.println(i); // вывести в монитор порта числа от 0 до 50!!!  
}
```

**continue – пропустить ход**

Пример:

```
for (byte i = 0; i < 100; i++) { // счётчик от 0 до 99  
if (i > 50 && i < 60) continue; // если i от 50 до 60, перейти в начало  
цикла  
Serial.println(i); // в монитор порта пойдут числа от 0 до 50, затем  
от 60 до 99  
}
```

## Задание 1

Подключите светодиод как показано на рисунке 8. Используйте резистор не меньше 200 Ом.

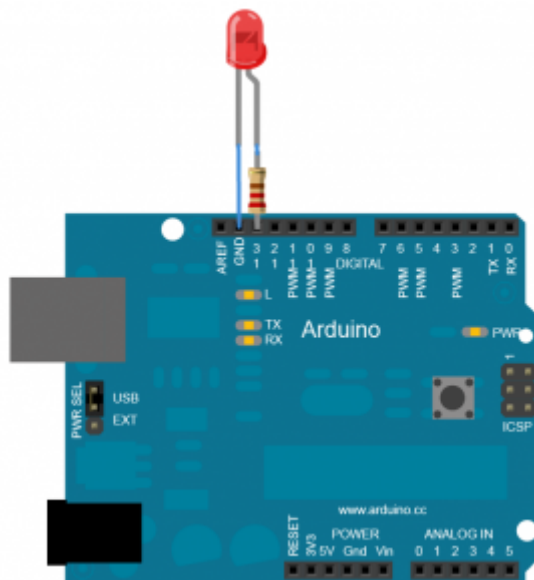


Рисунок 9 - Подключение светодиода.

1. Напишите программу мигания светодиодом используя приведенные выше команды. Светодиод включается, горит 1 секунды, выключается на 1 секунду и так по кругу.
2. Уменьшите время включения до 500 мс.
3. Уменьшите время включения и время выключения до 50 мс.
4. Сделайте вывод.

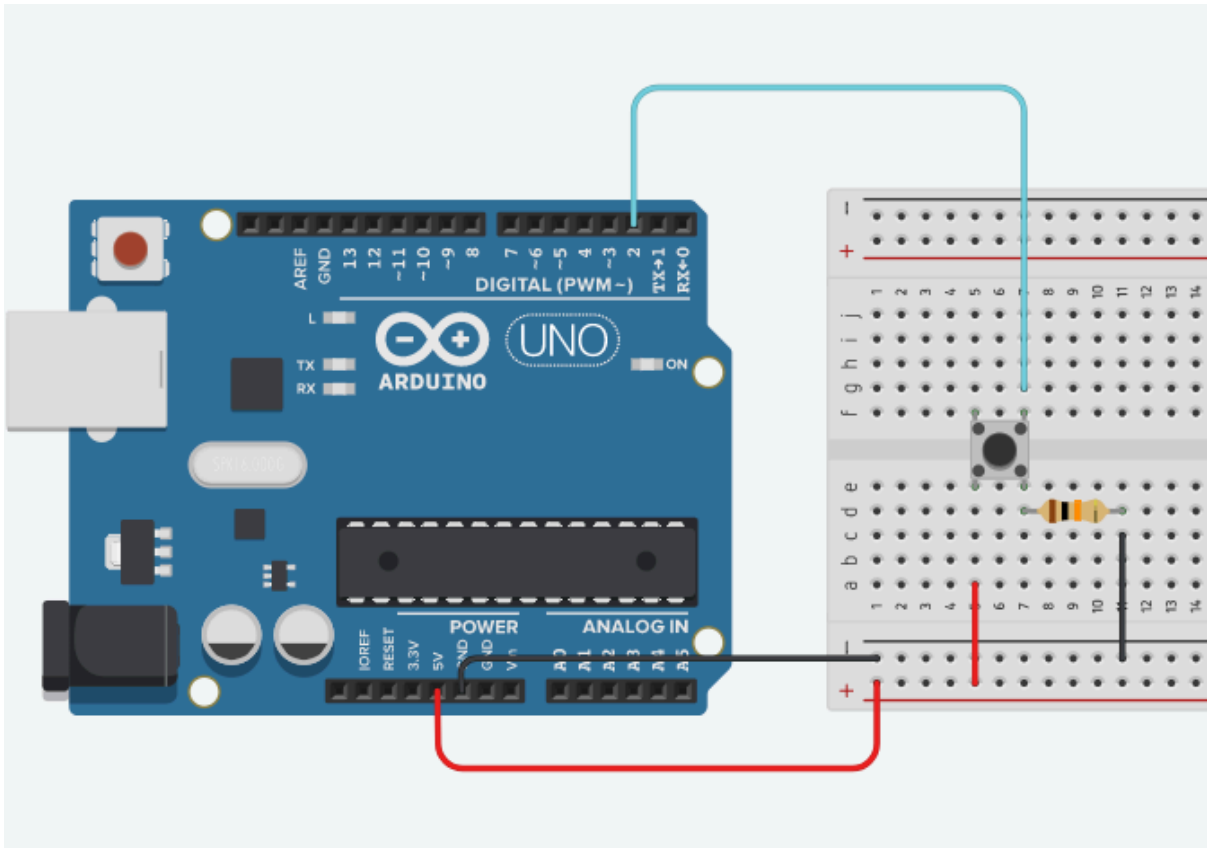
## Задание 2

1. Используйте схему с рисунка 9, только подключите положительную ножку светодиода к пину, который поддерживает ШИМ.
2. Напишите программу, которая плавно включает светодиод.
3. Напишите программу, которая сначала плавно включает, а затем плавно выключает светодиод.

Подсказка: используйте цикл `for`.

## Задание 3

1. Подключите к светодиоду кнопку, как показано на рисунке 10. Для подключения кнопки используйте резистор 10 кОм.



2. Напишите программу, которая по кнопке включает светодиод с использованием `if... else`.
3. Напишите программу, которая по кнопке включает светодиод с использованием `while`.
4. Напишите программу, которая по кнопке включает светодиод с использованием `do {} while`.
5. Подключите 2 кнопки. При помощи конструкции `switch case` при нажатии первой кнопки включайте первый светодиод, а при нажатии второй кнопки - второй светодиод.

#### Задание 4

1. Подключите светодиод к любому пину, поддерживающему ШИМ и подключите потенциометр как показано на рисунке 11.

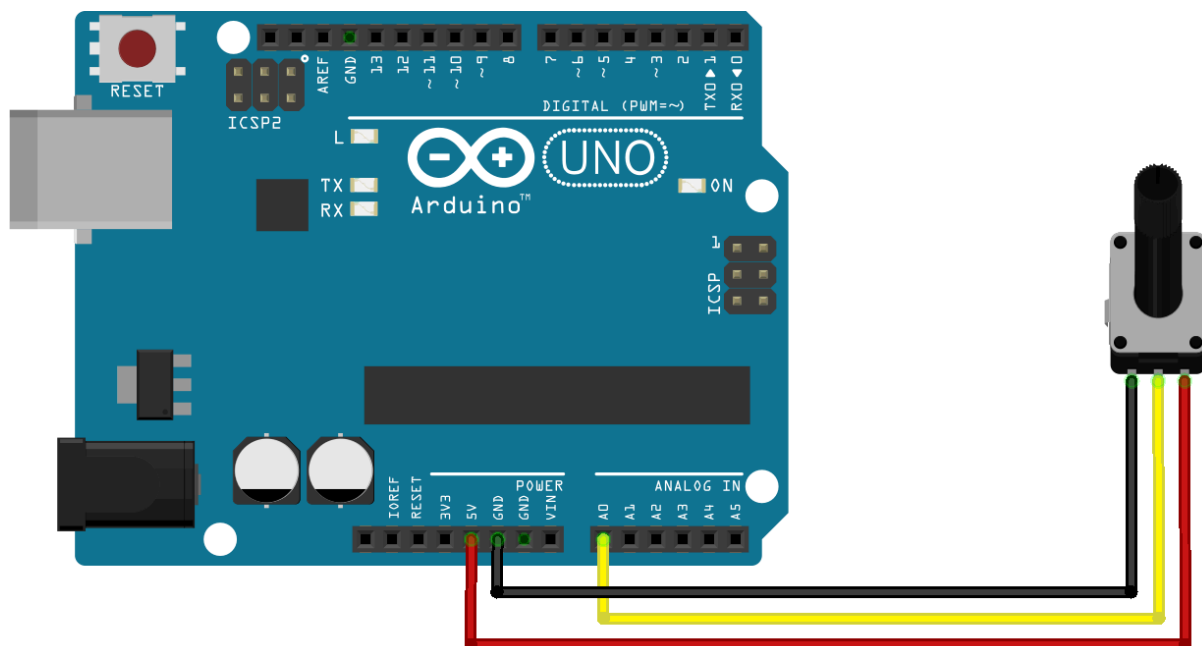


Рисунок 11 - Подключение потенциометра к ардуино

2. Напишите программу, которая регулирует яркость светодиода пропорционально повороту потенциометра.

#### Задание 5

1. Ознакомьтесь со статьей - <https://alexgyver.ru/lessons/serial/>
2. Отправьте в монитор порта "Hello, World!"
3. Отправьте в монитор порта напряжение с потенциометра
4. Отправьте данные в виде:

Напряжение на потенциометре: 1,25 В

Цифра должна меняться пропорционально повороту

### **Полезные ссылки:**

1. Справочник языка Arduino - <https://arduino.ru/Reference>
2. Уроки от AlexGyver - <https://alexgyver.ru/lessons/>
3. Пособие по Arduino - [https://alexgyver.ru/arduino/Arduino\\_lessons.pdf](https://alexgyver.ru/arduino/Arduino_lessons.pdf)