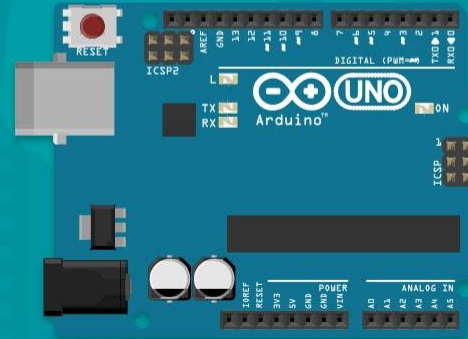
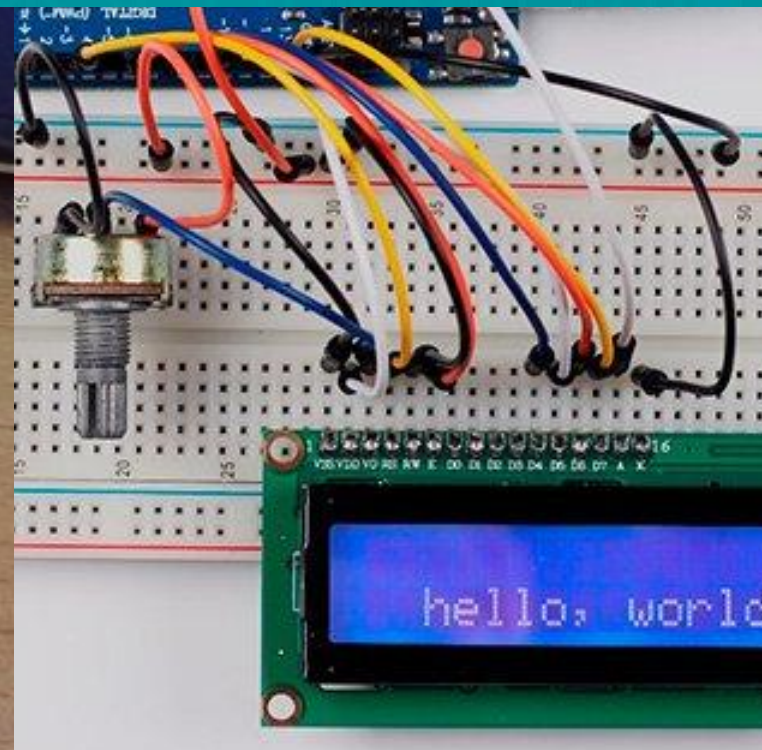
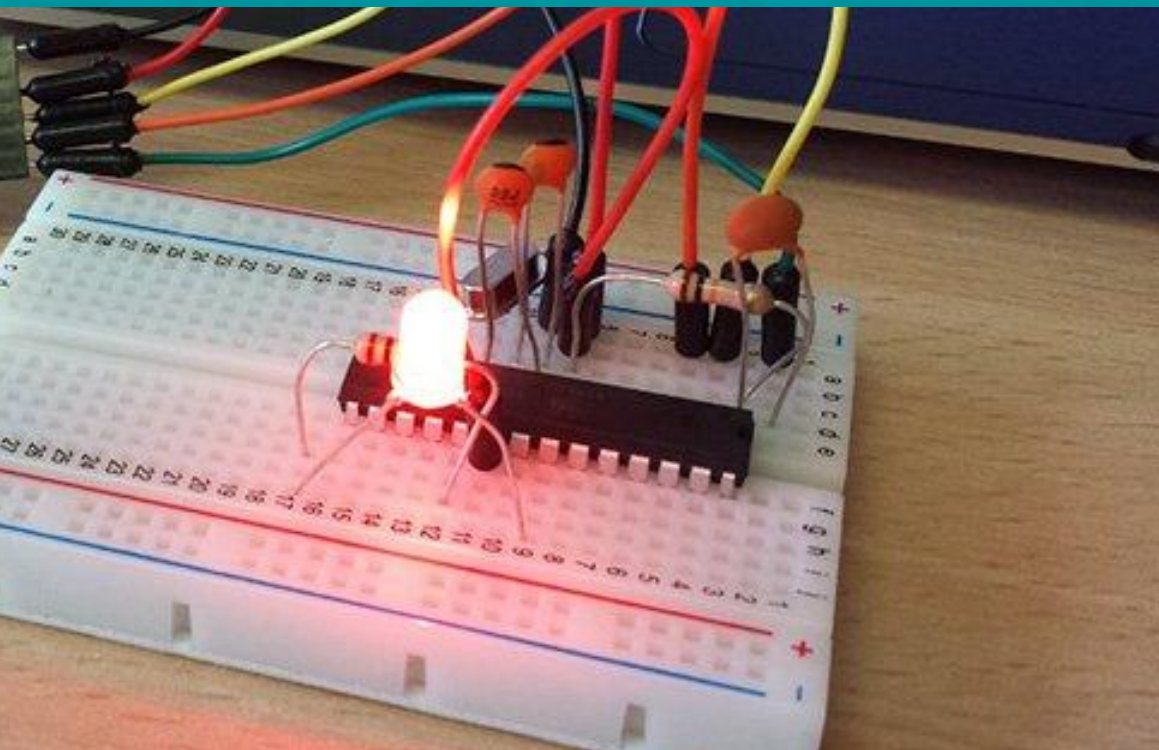


# ∞+ ARDUINO

ЛЫСЬ  
ВА



Педагог  
Павленк  
о  
Валерий





**Микроконтроллер Arduino программируется на специальном языке программирования, основанном на C/C++. Язык программирования Arduino является лишь разновидностью языка C++.**

**Дело в том, что все написанные скетчи компилируются с минимальными изменениями в программу на языке C/C++. Компилятор Arduino IDE значительно упрощает написание программ для этой платформы и создание устройств на Ардуино становится намного доступней людям, не имеющих больших познаний в языке C/C++.**

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



## Основы программирования Arduino

Arduino программируется на языке C. Данный раздел ориентирован на тех, кто имеет небольшой опыт программирования, и нуждается только в пояснении особенностей языка C и интерфейса Arduino. Если все это кажется Вам немного сложным, не беспокойтесь, начинайте работать с примерами устройств и понимание придет в процессе. Для более подробного изучения основ используйте сайт [arduino.cc](http://arduino.cc).

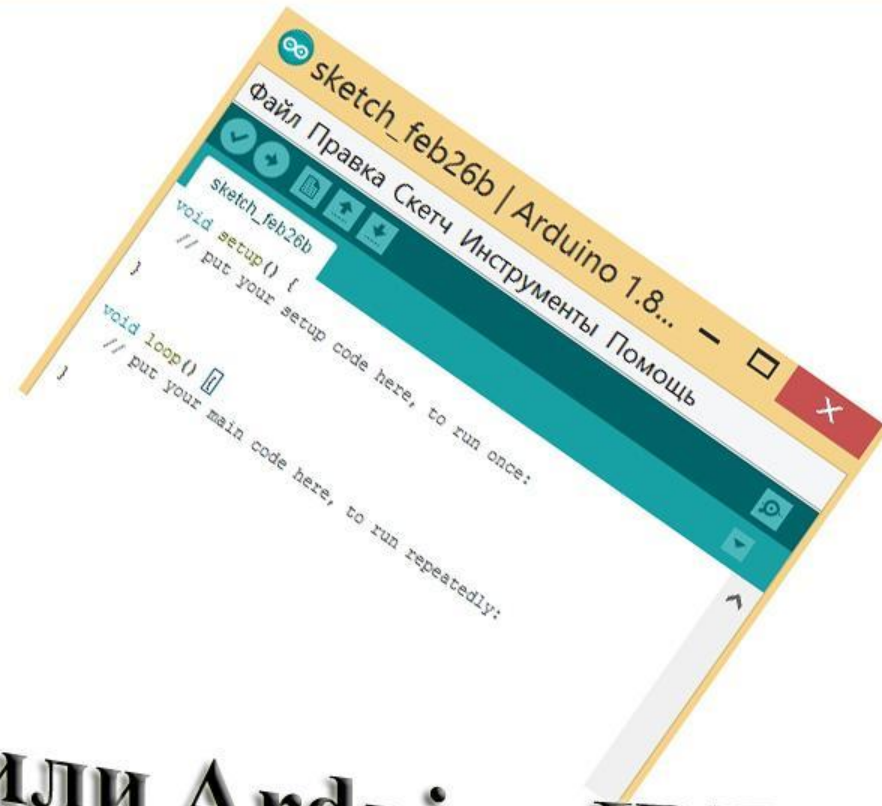


```
void setup() {  
  // put your setup code here, to run once:
```

```
}
```

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

```
}
```



**Пиши в блокноте или Arduino IDE**

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

}

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

**Всего две  
обязательные  
функции - подпрограммы**

### СТРУКТУРА

Каждая программа Arduino (часто называемая «скетч») имеет две обязательные функции (также называемые подпрограммами).

```
void setup() { }
```

Все команды, заключенные между фигурными скобками, выполняются только один раз, при первом запуске программы.

```
void loop() { }
```

Эта подпрограмма выполняется циклически вплоть до отключения питания, после завершения подпрограммы setup().

# В каждой программе

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

```

```
}
```

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

```

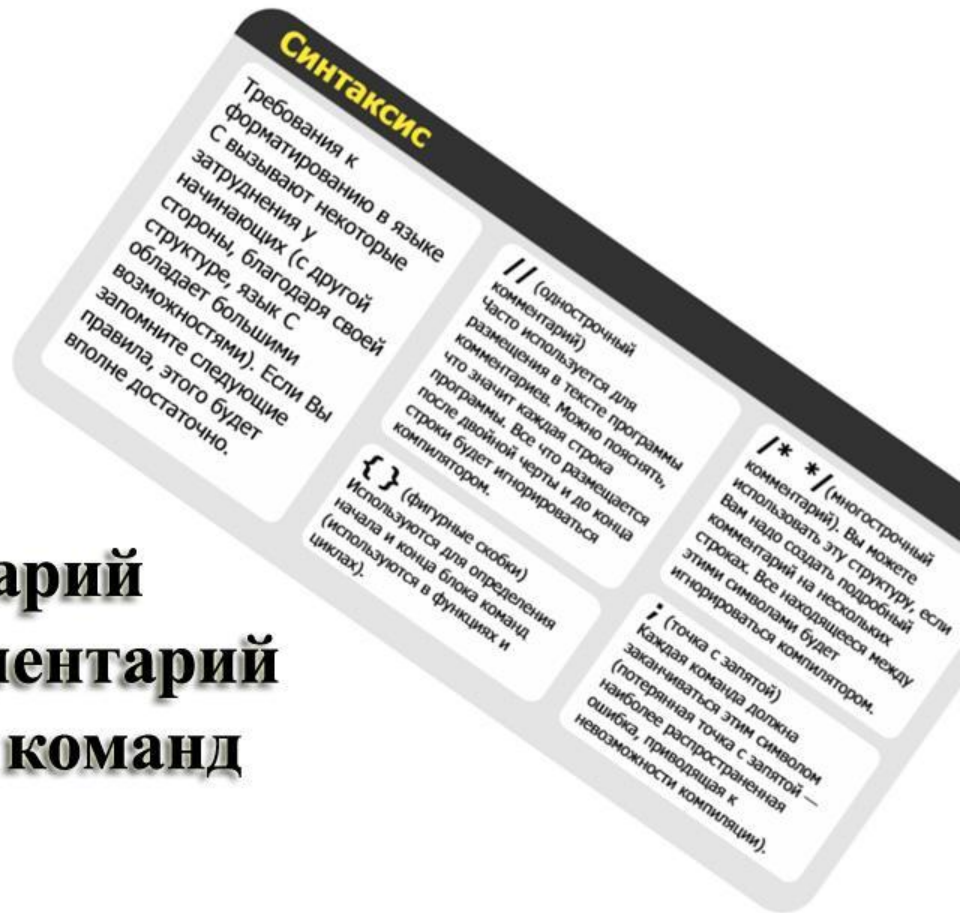
```
}
```

**// - однострочный комментарий**

**/\* \*/ - многострочный комментарий**

**{ } - начало и конец блока команд**

**;- окончание команды**





# Видео урок:

## Программирование длительности мигания светодиода.

Для просмотра щелкните по изображению

```
/* Здравствуйте, с Вами Валерий Павленко.  
Учимся программировать на C/C++  
Включение светодиода на 1 секунду и выключение его т  
Между знаками /* и */ размещается многострочный ком  
*/  
  
int ledPin = 13; // светодиод подключен к выходу 13 на пл  
  
// После двойного правого флэш размещается однострочн  
// Функция начальных установок setup () вызывается 1 р  
void setup() {  
// устанавливаем 13 контакт в режим вывода:
```

В СЕТЯХ ВИДЕО КЛИП РАЗМЕЩАЕТСЯ ОТДЕЛЬНО ОТ  
ПРЕЗЕНТАЦИИ

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

## Int - основная переменная

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

## Жонглирование цифрами

**Long - когда не хватает int**

### Переменные

Любая программа всего лишь определенным образом манипулирует числами. Переменные помогают жонглировать цифрами.

**int** (целочисленная)  
Основная рабочая лошадка, хранится в памяти с использованием двух байт (16 бит). Может содержать целое число в диапазоне -32 768 ... 32 767.

**long** (длинная)  
Используется в том случае, когда не хватает емкости int. Занимает в памяти 4 байта (32 бита) и имеет диапазон -2 147 483 648 ... 2 147 483 647.

**boolean** (двоичная)  
Простой тип переменной типа True/False. Занимает только один бит в памяти.

**float** (с плавающей запятой)  
Используется для вычислений с плавающей запятой. Занимает в памяти 4 байта (32 бита) и имеет диапазон -3.4028235E+38.

**char** (символ)  
Хранит один символ, используя кодировку ASCII (например «А» = 65). Использует один байт памяти (8 бит). Arduino оперирует со строками как с массивами символов.



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

## Математические операторы

Операторы  
используются для  
преобразования чисел.

= (присвоение) делает что-то равным чему-то(например  $x=10*2$  записывает в переменную  $x$  число 20).  
% остаток от деления). Например  $12\%10$  дает результат 2.  
+ (сложение)  
- (вычитание)  
\* (умножение)  
/ (деление)

# Преобразование чисел

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

# Логическое сравнение

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

```
}
```

## Операторы сравнения

Операторы,  
используемые для  
логического сравнения.

**==** (равно) (Например  $12==10$  не верно (FALSE),  $5==5$  верно (TRUE).)  
**!=** (не равно) (Например  $12!=10$  верно (TRUE),  $5!=5$  не верно (FALSE).)  
**<** (меньше) (Например  $12<10$  не верно (FALSE),  $12<12$  не верно (FALSE),  $12<14$  верно (TRUE).)  
**>** (больше) (Например  $12>10$  верно (TRUE),  $12>12$  не верно (FALSE),  $12>14$  не верно (FALSE).)

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

## Определение порядка выполнения команд

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

### Управляющие структуры

Для определения порядка выполнения команд (блоков команд) служат управляющие структуры. Здесь приведены только основные структуры. Более подробно можете ознакомиться на сайте [Arduino](#).

```
if (условие 1) {}  
else if (условие 2) {}  
else {}
```

Если условие 1 верно (TRUE) выполняются команды в первых фигурных скобках. Если условие 1 не верно (FALSE) то проверяется условие 2. Если условие 2 верно, то выполняются команды во вторых фигурных скобках, в противном случае выполняются команды в третьих фигурных скобках.

```
for (int i=0;  
i<число повторов;  
i++) {}
```

Эта структура используется для определения цикла. Цикл повторяется заданное число раз. Переменная *i* может увеличиваться или уменьшаться.



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

## Определение режима работы соответствующего порта

### Цифровые сигналы

`digitalWrite(pin, value);`  
Если порт установлен в режим OUTPUT, в него можно записать HIGH (логическую единицу, +5В) или LOW (логический ноль, GND).

`pinMode(pin, mode);`  
Используется, чтобы определить режим работы соответствующего порта. Вы можете использовать адреса портов 0...19 (номера с 14 по 19 используются для описания аналоговых портов 0...5). Режим может быть или INPUT (вход) или OUTPUT (выход).

`digitalRead(pin);`  
Если порт установлен в режим INPUT эта команда возвращает значение сигнала на входе HIGH или LOW.

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

# Возможность работы цифрового устройства с аналоговыми сигналами при помощи двух команд

## Аналоговые сигналы

Arduino - цифровое устройство, но может работать и с аналоговыми сигналами при помощи следующих двух команд:

`analogWrite(pin, value);`  
Некоторые порты Arduino (3,5,6,9,10,11) поддерживают режим ШИМ (широтно-импульсной модуляции). В этом режиме в порт посылаются логические единицы и нули с очень большой скоростью. Таким образом среднее напряжение зависит от баланса между количеством единиц и нулей и может изменяться в пределах от 0 (0В) до 255 (+5В).

`analogRead(pin);`

Если аналоговый порт настроен в режим INPUT, то можно измерить напряжение на нем. Может принимать значения от 0 (0В) до 1024 (+5В).

**Спасибо**

**за**

**ВНИМАНИЕ**