

УРОКИ ARDUINO

Версия 1.1 от 15.01.2019

Данное пособие является сборником команд, операторов и функций (ШПАРГАЛКА), стандартных для языка C++ и *Arduino Wiring*. Пособие создано с целью иметь в одном месте список всех «инструментов» для работы с Arduino, чтобы всегда можно было его открыть и сразу найти нужное.

- Информация в пособии дублирует, а иногда даже дополняет теоретическую часть в видеоуроках канала «Заметки Ардуинщика» <https://goo.gl/rgM5Kj>
- По следующему адресу находится **пополняемый сборник полезных алгоритмов** для различных областей применения <http://alexgyver.ru/arduino-algorithms/>
- По жизненно важным вопросам обращаться на почту alexgyvershow@ya.ru
- **Принимаю пожертвования** :3 http://alexgyver.ru/support_alex/

Приятного чтения!

С уважением, автор канала и пособия Александр «AlexGyver» Майоров

Оглавление

Урок 1 – данные и переменные.....	3
Структура программы	3
Типы данных	3
Особенности использования переменных	4
Особенности float	4
Урок 1.1 – операции с переменными и константами	4
Типы переменных.....	4
Константы.....	4
Математические операторы.....	5
Урок 2 – последовательный порт	5
Операторы библиотеки Serial.....	5
Урок 3	6
Условный оператор if.....	6
Операторы сравнения.....	7
Логические операторы.....	7
Оператор выбора switch.. case	7
Урок 4	8
Функции задержек	8

Функции таймера.....	8
Урок 5	9
Режимы работы цифровых портов	9
Генерация цифрового сигнала	9
Чтение цифрового сигнала	9
Урок 10	10
Чтение аналогового сигнала.....	10
Изменение диапазона значений	10
Урок 11	10
Генерация ШИМ сигнала	10
Урок 12	11
Цикл for, «счётчик».....	11
Цикл while, «с условием»	11
Цикл do while, «с условием»	11
break – выход из цикла	11
continue – пропустить ход.....	12
Урок 13	12
Функция, которая ничего не берёт и не возвращает	12
Функция, которая берёт и ничего не возвращает	12
Функция, которая берёт и возвращает.....	12
Урок 14	13
Генерируем случайные числа	13
Урок 15	13
Создание массива.....	13
Чтение – запись.....	13
Урок 16	14
Аппаратные прерывания	14

Урок 1 – данные и переменные

Видео вариант: <https://youtu.be/CRR1bz3VA>

Структура программы

// Однострочный комментарий (комментарии не занимают память)

/* Многострочный комментарий (комментарии не занимают память) */

#include - подключить файл (библиотеку)

void setup() {} - всё находящееся внутри {} будет выполнено 1 раз при загрузке Ардуино

void loop() {} - всё находящееся внутри {} бесконечно повторяется из начала в конец

После каждого «действия» ставится точка запятой ;

Типы данных

Переменная объявляется вот таким образом: <тип данных> <имя>;

```
int my_val; // объявить переменную my_val
```

Также можно сразу присвоить значение <тип данных> <имя> = <значение>;

```
int my_val = 2300; // объявить переменную my_val и присвоить ей число 2300
```

Также можно объявить несколько переменных одного типа сразу

```
int my_val = 2300, my_val2, my_val4, lolkek = 5; // объявить переменные
```

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

Особенности использования переменных

- Внимательно следите за значением, которое принимает переменная. Если значение превысит максимальное или принизит минимальное (выйдет из диапазона) для этого типа данных, то переменная сбросится в 0, а затем продолжит увеличение или уменьшение в том же направлении (например если присвоить byte значение 300, то она примет 300-255=45). Такую ошибку потом будет трудно отследить.
- Тип данных указывается при объявлении переменной ТОЛЬКО ОДИН РАЗ, далее переменная используется чисто по имени (обращение к переменной). При попытке сменить тип переменной (переобъявить переменную) вы получите ошибку. Но только в том случае, если переменная глобальная, либо когда локальная переобъявляется внутри функции, в которой она уже была объявлена.

Особенности float

- Присваивать только значение с точкой, даже если оно целое (10.0)
- Делить тоже только на числа с точкой, даже если они целые (переменная / 2.0)
- При делении целочисленного типа с целью получить число с плавающей точкой, писать (float) перед вычислением!
- Операции с числами типа float занимают гораздо больше времени, чем с целыми! Если нужна высокая скорость вычислений, лучше применять всякие хитрости, в стиле выполнения всех вычислений типом long, и результат уже переводить во float. Например вместо 3.25 вычислять в 100 раз большие числа, то есть 325

Урок 1.1 – операции с переменными и константами

Видео вариант: <https://youtu.be/tm831gRkscY>

Типы переменных

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.

Константы

const <тип> <имя> = <значение>; - объявить константу

```
const int my_val = 2300; // объявить константу my_val и присвоить ей число 2300
```

#define <имя> <значение> - объявить константу через define (точка запятой НЕ НУЖНА)

```
#define my_val 2300 // определить константу my_val и присвоить ей число 2300
```

Константа, объявленная через #define, работает немного по-другому: на этапе компиляции кода указанное название ЗАМЕНЯЕТСЯ на указанное значение, и хранится во флэш-памяти МК.

При попытке сменить значение константы ПОСЛЕ её объявления, вы получите ошибку!

Математические операторы

```
+ , - , * , / // сложить, вычесть, умножить...
pow(x, a); // возвести "x" в степень "a" ( xa ), pow может возводить в дробную степень!
sq(x); // возвести число "x" в квадрат ( x2 )
sqrt(x); // взять квадратный корень числа "x"
abs(x); // найти модуль числа, |x|
sin(x), cos(x), tan(x); // синус, косинус, тангенс
round(x); // математическое округление (если после запятой больше или равно 5, то округляем в большую сторону)
ceil(x); // округлить в БОльшую сторону
floor(x); // округлить в меньшую сторону
x += a; // прибавить "a" к "x"
x -= a; // вычесть "a" из "x"
x *= a; // домножить "x" на "a"
x /= a; // разделить "x" на "a"
x++; // увеличить "x" на 1
x--; // уменьшить "x" на 1
```

У меня на сайте есть статья, в которой можно посмотреть полный список поддерживаемых математических функций. Их там ОЧЕНЬ много! <https://alexgyver.ru/arduino-math/>

Урок 2 – последовательный порт

Видео вариант: <https://youtu.be/gmgw6nLgzbY>

Операторы библиотеки Serial

Serial - объект библиотеки Serial для работы с последовательным портом (COM портом)

Serial.begin(<скорость>); - открыть порт

Serial.begin(9600); // открыть порт на 9600 БОД

ВНИМАНИЕ! Скорость, установленная в begin(), должна быть равна скорости монитора порта (в самом мониторе правый нижний угол). Иначе в выводе получите крокозябры!

Serial.print(); // вывод в порт. Переменные и цифры напрямую, текст – в кавычках " "

Serial.println(); // вывод с переводом строки

Serial.println(val, n); // вывод переменной val (типа float) с n числом знаков после запятой

Serial.println(val, <базис>); // вывод с указанным базисом:

- DEC - десятичный (человеческие числа)
- HEX - 16-ричная система
- OCT - 8-ричная система
- BIN - двоичная система

Данные с компьютера попадают в буфер с объёмом 64 байта, и ждут обработки

Serial.available(); // проверить буфер на наличие входящих данных

Serial.read(); // прочитать входящие данные в символьном формате! Согласно ASCII

```

Serial.read() - '0'; // прочитать данные в целочисленном формате. По одной цифре!
Serial.parseInt(); // прочитать данные в целочисленном формате. Число целиком!
Serial.parseFloat(); // прочитать данные в формате с плавающей точкой. Число целиком!
Serial.setTimeout(); // поставить таймаут для parseInt и parseFloat. Можно поставить около 50
Serial.flush(); // очистить буфер порта

```

Все остальные методы Serial смотрите на официальном референсе

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

Код		Код		Код		Код		Код		Код		Код		Код	
32	пробел	44	,	56	8	68	D	80	P	92	\	104	h	116	t
33	!	45	-	57	9	69	E	81	Q	93		105	i	117	u
34	..	46	.	58	:	70	F	82	R	94	^	106	j	118	v
35	#	47	/	59	:	71	G	83	S	95	~	107	k	119	w
36	\$	48	0	60	<	72	H	84	T	96	`	108	l	120	x
37	%	49	1	61	=	73	I	85	U	97	a	109	m	121	y
38	&	50	2	62	>	74	J	86	V	98	b	110	n	122	z
39	'	51	3	63	?	75	K	87	W	99	c	111	o	123	{
40	(52	4	64	@	76	L	88	X	100	d	112	p	124	
41)	53	5	65	A	77	M	89	Y	101	e	113	q	125	}
42	*	54	6	66	B	78	N	90	Z	102	f	114	r	126	~
43	+	55	7	67	C	79	O	91	[103	g	115	s	127	Δ

Урок 3

Видео вариант: <https://youtu.be/hnKlmcN3jYE>

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 `if (!val)` - если `val` - ложь, т.е. 0)

Добавлено от WakeUp4Life

Использовать **boolean (bool)** лучше со значениями **true** и **false**.

C++ приравнивает ноль к **false**, а любое число к **true**.

К примеру:

```
bool x = 2;
if (x == 1) then {
Serial.println("истина");
} else {
Serial.println("ложь");
}
```

В порт выведется слово истина, хотя присваивали двойку!

Добавлено от Alexei Belousov

Небольшое добавление по условиям. Существует и укороченная запись условий

```
(a > b) ? c = true : c = false;
```

```
(a > b) ? Serial.println("a больше b") : Serial.println("b больше a");
```

Если A больше B то C равно истина, иначе C равно ложь..

Также имеет место запись присваивания переменной значения результата сравнения:

```
c = (a > b); // c принимает true, если a > b. Или false, если нет
```

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;
}
```

Добавлено от WakeUp4Life

Еще стоило указать возможность использования одновременно нескольких условий **switch** оператора:

```
switch (val) {
case 1:
case 2:
Serial.println("1 или 2");
break;
case 3:
Serial.println("3");
break;
}
```

// то есть при пропуске **break**; будут проверены и отработаны оба условия

Урок 4

Видео вариант: <https://youtu.be/lk7SwQ477mA>

Функции задержек

delay() - задержка, в скобках указывается число миллисекунд (в 1 сек 1'000 миллисекунд).

Максимальное значение типа *unsigned long* (4 байта), 4'294'967'295 мс, или около 1200 часов, или 50 суток.

delayMicroseconds() - задержка, в скобках указывается число микросекунд (в 1 сек 1'000'000 микросекунд). Максимальное значение 16'383 мкс, или 16 миллисекунд.

ИСПОЛЬЗОВАТЬ ЗАДЕРЖКИ НЕ РЕКОМЕНДУЕТСЯ! ОНИ ПОЛНОСТЬЮ "ВЕШАЮТ" СИСТЕМУ!

Функции таймера

millis() - возвращает количество миллисекунд, прошедших с момента включения МК.

- Макс. значение: 4'294'967'295 мс или 50 суток.
- Разрешение: 1 миллисекунда.

micros() - возвращает количество микросекунд, прошедших с момента включения МК.

- Макс. значение: 4'294'967'295 мкс или 70 мин
- Разрешение: 4 микросекунды

Пишем (*long*) перед умножением, чтобы программатор выделил нужное количество памяти для проведения операции! (для работы с большими числами).

Пример: **(long)23*24*60*60*1000**, если хотим получить ПРАВИЛЬНЫЙ результат умножения в виде числа миллисекунд, равного 23 дням.

На основе **millis()**, переменной *unsigned long* и условия можно сделать простой таймер, который будет срабатывать через указанные промежутки времени.

```

unsigned long last_time;           // глобальная переменная!!!

void loop() {
  if (millis() - last_time >= 5000) { // если прошло больше 5000 мс (5 секунд)
    // код, который выполняется каждые 5 секунд
    last_time = millis();           // сброс таймера
  }
}

```

Также есть моя библиотека **GyverTimer**, которая позволяет очень компактно использовать эту самую конструкцию. Скачать/почитать можно тут <https://community.alexgyver.ru/resources/biblioteka-gyvertimer.11/>

Урок 5

Видео вариант: <https://youtu.be/3UwgMAdV4xQ>

Режимы работы цифровых портов

- Аналоговые и цифровые пины могут работать как ВХОДЫ и как ВЫХОДЫ
- По умолчанию все пины работают КАК ВХОДЫ
- «Аналоговые» пины – это заблуждение. **Все пины цифровые**, но у некоторых есть АЦП (аналогово-цифровой преобразователь), у Нано и Уно эти пины подписаны как А. В то же время у Нано есть пины А6 и А7, к которым подключен ТОЛЬКО АЦП! То есть эти могут только измерять напряжение при помощи analogRead(), другие функции для них недоступны!

pinMode(pin, mode); - настроить порт

- **pin** - номер порта. Цифровые: 0 – 13. Аналоговые: 14 - 19, либо А0 - А5
- **mode** - режим работы порта
 - **INPUT** - вход, принимает сигнал
 - **OUTPUT** - выход, выдаёт 0 или 5 Вольт
 - **INPUT_PULLUP** - вход с подтяжкой к 5 В

Генерация цифрового сигнала

digitalWrite(pin, signal); - подать цифровой сигнал

- **pin** - номер порта. Цифровые: 0 – 13. Аналоговые: 14 - 19, либо А0 - А5
- **signal** - какой сигнал подаём
 - **LOW**, или 0 (ноль), или false - 0 Вольт
 - **HIGH**, или 1, или true - 5 Вольт

Чтение цифрового сигнала

digitalRead(pin); - прочитать цифровой сигнал

- **pin** - номер порта. Цифровые: 0 – 13. Аналоговые: 14 - 19, либо А0 - А5

Урок 10

Видео вариант: <https://youtu.be/ypH3W8r41Cw>

Чтение аналогового сигнала

analogRead(pin); - прочитать аналоговый сигнал (оцифровать)

- **pin** - номер пина. Аналоговые: 0 - 7

Функция возвращает значение 0.. 1023 в зависимости от напряжения на пине от 0 до опорного напряжения. По умолчанию опорное напряжение равно напряжению питания, при питании от USB это около 4.7 Вольт. Можно выбрать источник опорного напряжения:

- **analogReference(DEFAULT)** – напряжение питания как опорное
- **analogReference(INTERNAL)** – встроенное опорное на 1.1 Вольт
- **analogReference(EXTERNAL)** – опорное напряжение подаётся на пин Aref

Изменение диапазона значений

map(val, min, max, new_min, new_max); - возвращает величину в новом диапазоне

- **val** - входная величина
- **min, max** - минимальное и максимальное значение на входе в map
- **new_min, new_max** – соответственно мин. и макс. значения на выходе

```
val = analogRead(0);           // читаем с пина (0-1023)
val = map(val, 0, 1023, 50, 100); // переводим диапазон в 50-100
```

При изменении val от 0 до 1023 мы получим плавное изменение значения на выходе от 50 до 100

constrain(val, min, max); - ограничить диапазон переменной val до min и max

```
val = analogRead(0);           // читаем с пина (0-1023)
val = constrain(val, 50, 100); // ограничим диапазон до 50-100
```

При изменении val от 0 до 50 получим постоянные 50, от 50 до 100 получим изменение от 50 до 100, а от 100 до 1023 получим постоянные 100, так как диапазон у нас именно **ограничен**.

Урок 11

Видео вариант: <https://youtu.be/rCmaMST8qkg>

Генерация ШИМ сигнала

analogWrite(pin, duty);

- **pin** – пин, На котором генерировать ШИМ
- **duty** – величина 0.. 255, соответствует скважности ШИМ 0.. 100%

ШИМ пины Arduino NANO, UNO: 3, 5, 6, 9, 10, 11

ШИМ пины Arduino MEGA: все до 13

Частота ШИМ по умолчанию равна 490 Гц на пинах 3, 11, 9, 10 и 980 Гц на пинах 5 и 6. Для увеличения частоты ШИМ используются прямые команды процессору, можно найти в интернете. Либо смотрите мою библиотеку GyverHacks, там есть функции для удобной настройки частоты ШИМ. <https://community.alexgyver.ru/resources/biblioteka-gyverhacks.3/>

Урок 12

Видео вариант: <https://youtu.be/gIVcKbSeqFo>

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  
}
```

Урок 13

Видео вариант:

Функция, которая ничего не берёт и не возвращает

```
void myFunction() {}
```

- **void** – слово, показывающее, что функция ничего не возвращает
- **myFunction** – название функции

Пример:

```
void myFunction() { // задаём функцию  
  Serial.println("HELLO!"); // которая выведет в порт HELLO!  
}
```

В другом месте программы вызываем функцию как **myFunction()**; и в этом месте будет выслано слово HELLO! в порт.

Функция, которая берёт и ничего не возвращает

Пример:

```
void myFunction(int val) { // создаём функцию, на вход которой подаётся одно число  
  Serial.println(val); // и выводим его в порт  
}
```

В другом месте программы вызываем функцию как **myFunction(50)**; и в этом месте будет выслано в порт число 50.

Функция, которая берёт и возвращает

Пример:

```
int mySum(int val1, int val2) { // создаём функцию, на вход которой подаётся два числа  
  return val1 + val2; // возвращаем сумму чисел  
}
```

- **return** – оператор, возвращающий результат

В другом месте программы вызываем функцию как **mySum(50, 70)**; и функция вернёт результат 120.

Как пример:

```
Serial.println(mySum(50, 70)); // в порт будет послана сумма чисел, т.е. 120
```

Урок 14

Видео вариант:

Генерируем случайные числа

```
randomSeed(value); // функция, задающая начало отсчёта генератору псевдослучайных чисел
```

- **value** – любое число типа *long* (смотри урок №1)

```
random(min, max); // функция, возвращающая случайное число в диапазоне от min до max - 1
```

```
random(max); // то же самое, но возвращает от 0 до max - 1
```

Пример:

```
Serial.println( random(20) ); // вывести в порт случайное число от 0 до 19
```

Как получать максимально случайную последовательность чисел?

```
randomSeed(analogRead(0)); // в качестве опорного числа взять сигнал с НЕПОДКЛЮЧЕННОГО НИКУДА аналогового пина
```

Урок 15

Видео вариант:

Создание массива

```
<тип данных> <имя массива>[<число элементов>];
```

```
<тип данных> <имя массива>[<число элементов>] = {элемент1, элемент2...};
```

Если не указываются элементы, то обязательно нужно указать размер массива, чтобы под него выделилось место в памяти. Размер можно не указывать в том случае, если сразу указываются все элементы. Примеры:

```
int myInts[6];  
int myPins[] = {2, 4, 8, 3, 6};  
int mySensVals[6] = {2, 4, -8, 3, 2};  
char message[6] = "hello";
```

Чтение – запись

Главное помнить, что нумерация элементов НАЧИНАЕТСЯ С НУЛЯ!

```
myArray[5] = 10; // присвоить пятому элементу число 10
```

```
if (myArray[5] == 20) ..... // если элемент массива под номером 5 равен 20...
```

Пример. Забивка массива случайными числами

```
byte myArray[50];           // создать массив myArray на 50 ячеек
for (byte i = 0; i < 50; i++) { // счётчик от 0 до 49
myArray[i] = random(100);   // присвоить случайное число от 0 до 99 элементам массива под
номераи 0.. 49
}
```

Урок 16

Видео вариант:

Аппаратные прерывания

attachInterrupt(pin, function, state); - подключить прерывание

detachInterrupt(pin); - отключить прерывание

- **pin** – пин прерывания, для NANO и UNO это пины D2 и D3, соответствуют номерам 0 и 1
- **function** – название функции, которая будет вызвана при срабатывании прерывания
- **state** – режим обработки, их несколько:
 - **LOW** - вызывает прерывание, когда на порту LOW
 - **CHANGE** - прерывание вызывается при смене значения на порту, с LOW на HIGH и наоборот
 - **RISING** - прерывание вызывается только при смене значения на порту с LOW на HIGH
 - **FALLING** - прерывание вызывается только при смене значения на порту с HIGH на LOW

Пример:

Кнопка подключена к D2 и GND

```
void setup() {
  pinMode(2, INPUT_PULLUP);           // пин D2 подтянут к питанию
  attachInterrupt(0, myInterrupt, FALLING); // подключить прерывание на пин D2, обрабатывать
при падении сигнала и вызывать функцию myInterrupt
}

void myInterrupt() {                  // функция обработчика прерываний
  Serial.println("INTERRUPT!");       // при срабатывании вывести в порт слово INTERRUPT
}
```