**FUNGSI DAN PERINTAH YANG DIJALANKAN ARDUINO**

1. **STRUCTURE**
2. setup()

berfungsi ketika sketsa dimulai

1. loop()

menginisialisasi dan menetapkan nilai awal, untuk melakukan perulangan pada program.

**Control Structure**

1. If

Sebagai operator perbandingan untuk mencapai suatu kondisi tertentu.

1. If.. else

Untuk control yang mempunyai lebih dari satu kondisi.

1. for

Digunakan untuk peningkatan dan mengakhiri loop. Berguna untuk setiap operasi berulang-ulang, dan sering digunakan dalam kombinasi dengan array untuk beroperasi pada koleksi data / pin.

1. Switch case
2. Mengontrol aliran program dengan memungkinkan programmer untuk menentukan kode yang berbeda yang harus dilaksanakan dalam berbagai kondisi. membandingkan nilai variabel dengan nilai-nilai yang ditentukan jika ditemukan kecocokan maka dijalakan.
3. While

Selama statement bernilai TRUE maka akan terus melakukan perulangan perintah.

1. Do..while

Hampir sama dengan pernyataan “while”, bedanya bila pada pernyataan “while” , test kondisi diuji dahulu, dan bila test kondisi bernilai benar maka pernyataan yang ada di dalam blok “while” akan dieksekusi, kemudian test kondisi akan diinkremental. Pada pernyataan “do-while”, kondisi menjadi terbalik, yaitu pernyataan utama akan dieksekusi terlebih dahulu, setelah itu baru test kondisi diuji, jika test kondisi benar maka pernyataan utama akan diulang, dan jika salah program akan keluar dari blok “do-while”.

1. Break

Digunakan untuk keluar dari pernyataan **do** , atau **while loop**, melewati kondisi loop normal. Hal ini juga digunakan untuk keluar dari pernyataan **switch**.

1. Continue

Pernyataan ini melewatkan sisa dari iterasi saat ini loop (lakukan, untuk, atau saat). Dengan memeriksa ekspresi kondisional dari loop, dan melanjutkan dengan iterasi berikutnya.

1. Return  
   Untuk menghentikan fungsi dan mengembalikan nilai dari fungsi ke fungsi panggilan, jika diinginkan
2. Goto

Instruksi untuk mengarahkan eksekusi program ke pernyataan yang diawali oleh sebuah label atau identifier khusus seperti tanda (:) lalu mengarahkan eksekusi program **tampil** yang diikuti dengan mengerjakan perintah yang sudah ditaruh pada label. Perintah goto ini juga bisa digunakan untuk keluar dari *loop* dan *nested loop* dengan lebih mudah dan ringkas.

**Further Syntax**

1. ; (semicolon)

Digunakan setelah berakhirnya *statement*. Merupakan suatu pembatas disetiap *statement*.

1. {} (curly braces)

Mendefinisikan awal dan akhir dari sebuah blok fungsi

1. // (single line comment)

Sama dengan *multi-line coment* hanya saja perbaris.

1. /\*\*/ (multi-line comment)

Semua *statement* yang di tulis dalam block comments tidak akan di eksekusi dan tidak akan di compile sehingga tidak mempengaruhi besar program yang di buat untuk di masukan dalam board arduino

1. #define

Komponen yang digunakan untuk memberi nama pada nilai konstanta sebelum program dikompilasi

1. #include

Memasukkan sketsa program ke dalam *libraries*.

**Arithmetic Operator**

Operator yang digunakan untuk memanipulasi angka.

1. = (assignment operator)

Sama dengan (hasil)

1. + (addition)

Penambahan

1. – (subtraction)

Pengurangan

1. \* (multiplication)

Perkalian

1. / (division)

Pembagian

1. % (modulo)

Menghasilkan sisa dari hasil pembagian suatu angka dengan angka yang lain

**Comprasion Operator**

Operator yang digunakan untuk membandingkan nilai logika.

1. == (equal to)

Sama dengan (misalnya: 12 == 10 adalah FALSE (salah) atau 12 == 12 adalah TRUE (benar))

1. != (not equal to)

Tidak sama dengan (misalnya: 12 != 10 adalah TRUE (benar) atau 12 != 12 adalah FALSE (salah))

1. [<](https://www.arduino.cc/en/Reference/If) (less than)

Lebih kecil dari (misalnya: 12 < 10 adalah FALSE (salah) atau 12 < 12 adalah FALSE (salah) atau 12 < 14 adalah TRUE (benar))

1. [>](https://www.arduino.cc/en/Reference/If) (greater than)

Lebih besar dari (misalnya: 12 > 10 adalah TRUE (benar) atau 12 > 12 adalah FALSE (salah) atau 12 > 14 adalah FALSE (salah))

1. [<=](https://www.arduino.cc/en/Reference/If) (less than or equal to)

Menyatakan operand pertama lebih kecil atau lebih kecil sama dengan operand kedua.

1. [>=](https://www.arduino.cc/en/Reference/If) (greater than or equal to)

Menyatakan operand pertama lebih besar atau lebih besar sama dengan operand kedua.

**Boolean Operators**

Digunakan pada kondisi if dalam *statement*.

1. [&&](https://www.arduino.cc/en/Reference/Boolean) (and)

Ouput akan bernilai TRUE jika kedua operand bernilai TRUE.

1. [||](https://www.arduino.cc/en/Reference/Boolean) (or)

Output akan bernilai FALSE jika kedua operand bernilai FALSE.

1. [!](https://www.arduino.cc/en/Reference/Boolean) (not)

Operator NOT menyatakan kebalikan nya. Contoh jika input bernilai FALSE output akan bernilai TRUE.

**Pointer Access Operators**

Berfungsi sebagai penunjuk baik ke variable, fungsi atau objek.

1. [\* dereference operator](https://www.arduino.cc/en/Reference/Pointer)

Address of atau alamat dari variable/fungsi tersebut di dalam memory cel

1. [& reference operator](https://www.arduino.cc/en/Reference/Pointer)

Value pointed by atau nilai dari variable yang direferensikan.

**Bitwise Operators**

1. [&](https://www.arduino.cc/en/Reference/BitwiseAnd) (bitwise and)

Digunakan diantara dua ekpresi bilangan bulat yang dioperasikan pada perhitungan biner. Jika kedua masukan bit adalah 1, output yang dihasilkan adalah 1, jika output adalah 0.

1. [|](https://www.arduino.cc/en/Reference/BitwiseAnd) (bitwise or)

Digunakan diantara dua ekpresi bilangan bulat yang dioperasikan pada perhitungan biner. Output bernilai 1 kecuali jika input kedua bilangan biner adalah 0.

1. [^](https://www.arduino.cc/en/Reference/BitwiseAnd) (bitwise xor)

Operator ini mirip dengan bitwise operator OR kecuali jika bernilai 1 untuk posisi tertentu kapan tepatnya salah satu bit masukan untuk posisi yang 1. Jika keduanya 0 atau keduanya 1, operator XOR mengevaluasi ke 0

1. [~](https://www.arduino.cc/en/Reference/BitwiseXorNot) (bitwise not)

Tidak seperti & dan |, bitwise NOT operator diterapkan pada operan tunggal. Bitwise NOT berubah setiap bit untuk kebalikannya: 0 menjadi 1, dan 1 menjadi 0.

1. [<<](https://www.arduino.cc/en/Reference/Bitshift) (bitshift left)

Digunakan untuk operator pergeseran nilai biner ke kiri.

Contoh : int a = 00000111

int b = a << 3

int c = b << 3

artinya b = a digeser kekiri 3 bit atau b = 00111000

1. [>>](https://www.arduino.cc/en/Reference/Bitshift) (bitshift right)

Digunakan untuk operator pergeseran nilai biner ke kanan.

Contoh : int a = 00000111

int b = a >> 2

int c = b >> 3

artinya b = a digeser kekanan 2 bit atau b = 11000001

**Compound Operators**

1. [++](https://www.arduino.cc/en/Reference/Increment) (increment)

Kenaikan variable. Menambahkan suatu nilai variabel

1. [--](https://www.arduino.cc/en/Reference/Increment) (decrement)

Penurunan variable. Mengurangi suatu nilai variable.

Melakukan operasi matematika pada variabel dengan konstanta lain atau variabel.

1. [+=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound addition)

Penjumlahan gabungan

1. [-=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound subtraction)

Pengurangan gabungan

1. [\*=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound multiplication)

Perkalian gabungan

1. [/=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound division)

Pembagian gabungan

1. [%=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound modulo)

Hasil bagi gabungan

1. [&=](https://www.arduino.cc/en/Reference/BitwiseCompoundAnd) (compound bitwise and)

sering digunakan dengan variabel dan konstanta untuk memaksa bit tertentu dalam variabel untuk negara LOW (dengan 0). Hal ini sering disebut dalam panduan pemrograman sebagai "clearing" atau "ulang" bit.

1. [|=](https://www.arduino.cc/en/Reference/BitwiseCompoundOr) (compound bitwise or)

Digunakan dengan variabel dan konstanta untuk "mengatur" (set ke 1) bit tertentu dalam variabel.

1. **VARIABLES**

**Konstanta**

Konstanta adalah ekspresi yang telah ditetapkan dalam bahasa Arduino. Digunakan untuk membuat program lebih mudah dibaca. Konstanta diklasifikasikan dalam kelompok :

1. Mendefinisikan *Logical levels*: *true* dan *false* (Boolean Konstanta)

Ada dua konstanta yang digunakan untuk menyatakan kebenaran dan kesalahan dalam bahasa Arduino: *true*, dan *false*.

***false*** *: false* adalah lebih mudah untuk ditentukan. *false* didefinisikan sebagai 0 (nol).

***true*** *: true* sering didefinisikan sebagai 1, yang memang benar, tetapi konstanta *true* memiliki definisi yang lebih luas. Setiap bilangan bulat yang bukan nol dinyatakan *true*, dalam logika Boolean. Jadi -1, 2 dan -200 semua dapat didefinisikan sebagai *true* juga.

1. Mendefinisikan *Pin levels* : HIGH and LOW

Hanya ada dua kemungkinan untuk mengatur nilai *digital pin* yaitu : HIGH atau LOW

***HIGH***

Ketika pin dikonfigurasi sebagai INPUT dengan pinMode (), dan membaca dengan digitalRead (), Arduino (Atmega) akan memberikan nilai *HIGH* jika:

* tegangan lebih besar dari 3.0V pada pin (5V boards);
* tegangan lebih besar dari 2.0V pada pin (3.3V boards);

Sebuah pin juga dapat dikonfigurasi sebagai INPUT dengan pinMode (), dan kemudian membuat *HIGH* dengan digitalWrite (). Hal ini akan mengaktifkan internal 20K resistor pullup, yang akan menarik pin input untuk membaca *HIGH* kecuali ditarik menjadi *LOW* oleh sirkuit eksternal. Ini adalah bagaimana INPUT\_PULLUP bekerja. Ketika pin dikonfigurasi ke OUTPUT dengan fungsi pinMode (), dan diatur menjadi *HIGH* dengan digitalWrite (), pin berada pada tegangan:

* 5 volt (5V boards);
* 3,3 volt (3.3V boards);

**LOW**

Nilai LOW dapat dinyatakan secara berbeda tergantung dari apakah pin diatur sebagai INPUT atau OUTPUT. Ketika pin dikonfigurasi sebagai INPUT dengan pinMode (), dan membaca dengan fungsi digitalRead (), Arduino (Atmega) akan menyatakan LOW jika:

* tegangan kurang dari 1.5V pada pin (5V boards);
* tegangan kurang dari 1.0V (Approx) pada pin (3.3V boards);

Ketika pin dikonfigurasi untuk OUTPUT dengan pinMode (), dan diatur untuk LOW dengan digitalWrite (), pin berada pada 0 volt (baik 5V dan 3.3V papan). Ini dapat terjadi misalnya cahaya LED yang terhubung melalui resistor seri untuk 5 volt (atau 3,3 volt).

1. Mendefinisikan mode Digital Pins: **INPUT, INPUT\_PULLUP,** dan **OUTPUT**

Digital pin dapat digunakan sebagai INPUT, INPUT\_PULLUP, atau OUTPUT. Mengubah pin dengan pinMode () dapat mengubah *behaviour* listrik pada pin.

**INPUT**

Arduino (Atmega) pin dikonfigurasi sebagai INPUT dengan pinMode () dinyatakan memiliki *high-impedance state*. Pin yang dikonfigurasi sebagai INPUT membuat *demands* yang sangat kecil pada sirkuit yang mereka coba, setara dengan resistor seri 100 megaohms di depan pin. Hal ini membuat mereka berguna untuk membaca sensor.

**INPUT\_PULLUP**

The Atmega mikrokontroler pada Arduino memiliki resistor pull-up internal (resistor yang terhubung ke daya internal) yang dapat Anda akses. Jika Anda memilih untuk menggunakan ini dibandingkan eksternal resistor pull-up, Anda dapat menggunakan argumen INPUT\_PULLUP di pinMode ().

Pin yang dikonfigurasi sebagai input dengan menggunakan INPUT atau INPUT\_PULLUP bisa rusak atau hancur jika mereka terhubung ke tegangan di bawah tanah (tegangan negatif) atau di atas rel listrik positif (5V atau 3V).

**OUTPUT**

Pin yang dikonfigurasi sebagai OUTPUT dengan pinMode () dinyatakan memilki *low-impedance state*. Ini berarti bahwa mereka dapat memberikan sejumlah besar arus untuk sirkuit lainnya. Atmega pin dapat memberikan arus menyerap arus hingga 40 mA (milliamps) dari arus untuk perangkat lain / sirkuit. Hal ini membuat mereka berguna untuk menyalakan LED karena LED biasanya menggunakan kurang dari 40 mA. Beban lebih dari 40 mA akan membutuhkan transistor atau sirkuit antarmuka lainnya.

Pin dikonfigurasi sebagai output dapat rusak atau hancur jika mereka terhubung ke salah satu tanah atau rel listrik positif.

1. Mendefinisikan *built-ins:* LED\_BUILTIN

Kebanyakan Arduino boards memilki sebuah pin yang terhubung ke LED secara seri dengan sebuah resistor. Konstanta LED\_BUILTIN adalah angka pin yang menunjukan LED mana yang terhubung. Kebanyakan *boards* memiliki LED yang terhubung dengan *digital pin* 13

1. *Integer Constants*

*Integer Constants* (konstanta bilangan bulat) adalah angka-angka yang digunakan langsung, misalnya 123. Biasanya, konstanta bilangan bulat diperlakukan sebagai basis 10 (desimal) bilangan bulat, tapi notasi khusus *(formatters*) dapat digunakan untuk memasukkan angka dalam basis-basis lain.

1. Mirip dengan konstanta integer, konstanta *floating point* digunakan untuk membuat kode lebih mudah dibaca. konstanta *floating point* ditukar pada waktu *compile* untuk nilai yang dievaluasi oleh *expression*.

contoh:

n = 0,005;

konstanta *floating point* juga dapat dinyatakan dalam berbagai notasi ilmiah. 'E' dan 'e' keduanya diterima sebagai indikator eksponen valid.

|  |  |  |
| --- | --- | --- |
| Floating-point constant | Evaluates to : | Also Evaluates to : |
| 10.0 | 10 |  |
| 2.34E5 | 2.34 \* 10^5 | 234000 |
| 67e-12 | 67.0 \* 10^-12 | .000000000067 |

**Tipe Data**

1. **void,** hanya digunakan dalam deklarasi fungsi. Hal ini menunjukkan bahwa fungsi diharapkan untuk mengembalikan informasi kosong kepada fungsi yang telah dipanggil.
2. **boolean**, hanya memiliki 2 data yaitu benar (true) dan salah (false). Tipe data boolean hanya membutuhkan 1 byte memori.
3. **char** atau karakter adalah tipe data untuk menyatakan suatu karakter seperti “A”, “B”, “C” , “+” , “.” dan sebagainya. Karakter ini disimpan dalam bentuk angka. Untuk mengkonversikan bentuk angka ke bentuk karakter dapat menggunakan tabel ASCII
4. **unsigned char** serupa dengan tipe char tetapi tanpa nilai negatif sehingga unsigned char memiliki nilai dari 0 hingga 255 atau sama dengan tipe data byte.
5. **byte** memiliki 8 bit data (0 – 255). Tipe data byte tidak memiliki nilai negatif
6. **int** digunakan untuk menyatakan tipe data integer (bilangan bulat). Tipe data integer berkisar antara -32768 sampai 32768 (-215 hingga (215-1)). Nilai bilangan ini membutuhkan 2 byte memori mikrokontroller
7. **unsigned int** digunakan untuk mendeklarasikan tipe data bilangan bulat positif atau bernilai 0 hingga 65535. Sama dengan integer, tipe data ini juga membutuhkan 2 byte memori mikrokontroller.
8. **word** sama dengan tipe data unsigned int. Besar data word adalah 16 bit atau membutuhkan 2 byte memori mikrokontroller. Tipe data ini jarang digunakan.
9. **long** adalah tipe data untuk menampung bilangan bulat yang berkisar antara -2.147.483.648 hingga 2.147.483.647. variabel bertipe long ditulis dengan akhiran L atau l. Contoh 456789980L.
10. **unsigned long** adalah tipe data yang sama dengan long, tetapi dihitung dari angka 0 atau mempunyai nilai berkisar 0 hingga 4.292.967.295. Nilai variabel ini ditulis dengan kode UL diakhir konstanta
11. **short,** pada semua Arduinos (basis ATMega dan ARM) sebuah short menyimpan nilai 16-bit (2-byte)
12. **float** berguna untuk menyimpan bilangan real. Angka yang bisa disimpan dari -3,4028235 x 1038 hingga 3,4028235 x 1038. Angka dengan tipe float sangat besar sekali, sehingga sangat jarang digunakan karena akan memperlambat kerja prosessor mikrokontroller dan banyak memakai memori, kecuali bila memang sangat dibutuhkan dalam program.
13. **double**, tipe data double dan float untuk arduino tidak ada bedanya.
14. string-char array, sistem Arduino mendukung tipe data string dalam bentuk kumpulan karakter. Contoh : char inistringku[]=”haha.hehe”;
15. string-object, sama seperti string dalam bahasa pemrograman lain. Penggunaan string-object ini dapat menghabiskan *dynamic memory* yang dapat memunculkan *debug.*
16. **array**, adalah kumpulan variabel yang diakses dengan nomor indeks. Array dalam bahasa pemrograman C (bahasa yang digunakan pada Arduino), terbilang cukup rumit, tetapi dengan menggunakan array yang sederhana menjadi relatif mudah

**Konversi**

1. char(), merubah suatu nilai ke tipe data char
2. byte(), merubah suatu nilai ke tipe data byte
3. int(), merubah suatu nilai ke tipe data int
4. word(), merubah suatu nilai ke tipe data word atau membuat suatu *word* dari 2bytes
5. long(), merubah suatu nilai ke tipe data long
6. float(), merubah suatu nilai ke tipe data float

**Lingkup Variabel dan Kualifikasi**

1. variable scope

Variabel dalam bahasa pemrograman C, yang digunakan Arduino, memiliki sifat yang disebut lingkup. Hal ini berbeda dengan versi awal bahasa seperti BASIC dimana setiap variabel adalah variabel global.

Sebuah variabel global adalah salah satu yang dapat dilihat oleh setiap fungsi dalam sebuah program. Variabel lokal hanya dapat dilihat didalam fungsi. Dalam lingkungan Arduino, setiap variabel dinyatakan di luar fungsi (misalnya setup (), lingkaran (), dll), adalah variabel global.

Ketika program menjadi lebih kompleks, variabel lokal adalah cara yang berguna untuk memastikan bahwa hanya satu fungsi yang memiliki akses ke variabelnya sendiri. Hal ini untuk mencegah kesalahan pemrograman ketika salah satu fungsi secara tidak sengaja memodifikasi variabel yang digunakan oleh fungsi lain.

Hal ini juga kadang-kadang berguna untuk mendeklarasikan dan menginisialisasi variabel untuk loop. Hal ini menciptakan sebuah variabel yang hanya dapat diakses dari dalam kurung loop.

1. static

digunakan untuk membuat variabel yang hanya terlihat oleh satu fungsi. Namun tidak seperti variabel lokal yang bisa diciptakan dan dihancurkan setiap kali fungsi dipanggil, variabel static bertahan melampaui panggilan fungsi, melestarikan data mereka diantara fungsi yang dipanggil.

Variabel yang dideklarasikan sebagai static hanya akan dibuat dan diinisialisasi pada saat fungsi pertama kali dipanggil.

1. volatile,

volatile merupakan kata kunci yang dikenal sebagai pengkualifikasi variabel, biasanya digunakan sebelum tipe data dari suatu variabel, untuk memodifikasi cara memperlakukan variabel pada compiler dan program selanjutnya.

1. const,

ini adalah pengkualifikasi variabel yang mengubah perilaku dari variabel, membuat variabel menjadi "read-only"yang artinya nilai variabel tidak dapat diubah. Anda akan mendapatkan kesalahan kompilator jika Anda mencoba untuk memberikan nilai pada variabel const.

**Utility**

1. sizeof, mengembalikan jumlah byte dalam jenis variabel, atau jumlah byte yang ditempati oleh sebuah array.
2. PROGMEM, menyimpan data dalam flash (program) memori bukan SRAM. PROGMEM adalah pengubah variabel, harus digunakan hanya dengan tipe data yang didefinisikan dalam pgmspace.h. Ini memberitahu compiler "menaruh informasi ini ke dalam flash memory", bukan ke SRAM, di mana ia biasanya akan pergi. PROGMEM merupakan bagian dalam library pgmspace.h yang tersedia dalam arsitektur AVR saja.
3. **FUNCTIONS**

**Digital I/O**

* **pinMode()**

Mengkonfigurasi pin yang ditentukan untuk berperilaku baik sebagai input atau output. Lihat deskripsi pin digital untuk rincian tentang fungsi pin.

* **digitalWrite()**

Tulis HIGH atau nilai LOW ke pin digital.  
  
Jika pin telah dikonfigurasi sebagai OUTPUT dengan pinMode (), tegangan akan diatur ke nilai yang sesuai: 5V (atau 3.3V di papan 3.3V) untuk HIGH, 0V (ground) untuk LOW.  
  
Jika pin dikonfigurasi sebagai INPUT, digitalWrite () akan mengaktifkan (HIGH) atau menonaktifkan (LOW) yang pullup internal pada pin input. Dianjurkan untuk mengatur pinMode () untuk INPUT\_PULLUP untuk mengaktifkan internal pull-up resistor. Lihat pin digital tutorial untuk informasi lebih lanjut.

CATATAN: Jika Anda tidak mengatur pinMode () ke OUTPUT, dan menghubungkan LED untuk pin, saat memanggil digitalWrite (HIGH), LED mungkin muncul redup. Tanpa secara eksplisit menetapkan pinMode (), digitalWrite () akan telah mengaktifkan pull-up resistor internal, yang bertindak seperti sebuah resistor pembatas arus besar.

* digitalRead()

Membaca nilai dari pin digital ditentukan, baik HIGH atau LOW.

**Analog I/O**

* **analogReference()**

Mengkonfigurasi tegangan referensi yang digunakan untuk input analog (yaitu nilai yang digunakan sebagai bagian atas berbagai masukan). Pilihannya adalah:

DEFAULT: referensi analog default 5 volt (pada papan 5V Arduino) atau 3,3 volt (di papan 3.3V Arduino)

* INTERNAL: built-in referensi, sama dengan 1,1 volt pada ATmega168 atau ATmega328 dan 2,56 volt pada ATmega8 (tidak tersedia pada Mega Arduino)
* INTERNAL1V1: built-in referensi 1.1V (Arduino mega saja)
* INTERNAL2V56: built-in referensi 2.56V (Arduino mega saja)  
  EKSTERNAL: tegangan diterapkan pada pin AREF (0 5V hanya) digunakan sebagai referensi.
* **analogRead()**

Membaca nilai dari pin analog yang ditentukan. Papan Arduino berisi 6 channel (8 saluran pada Mini dan Nano, 16 di Mega), 10-bit analog ke digital. Ini berarti bahwa itu akan memetakan tegangan masukan antara 0 dan 5 volt menjadi nilai integer antara 0 dan 1023. Ini menghasilkan resolusi antara pembacaan dari: 5 volt / 1024 unit atau, 0,0049 volt (4,9 mV) per unit. Berbagai masukan dan resolusi dapat diubah dengan menggunakan analogReference().  
  
Dibutuhkan sekitar 100 mikrodetik (0,0001 s) untuk membaca input analog, sehingga tingkat membaca maksimum adalah sekitar 10.000 kali per detik.

* **analogWrite()**

Menulis nilai analog (gelombang PWM) ke pin. Dapat digunakan untuk menyalakan LED di berbagai brightnesses atau mengendarai motor pada berbagai kecepatan. Setelah panggilan untuk analogWrite (), pin akan menghasilkan gelombang persegi stabil siklus tertentu sampai panggilan berikutnya ke analogWrite () (atau panggilan untuk digitalRead () atau digitalWrite () pada pin yang sama). Frekuensi sinyal PWM pada kebanyakan pin adalah sekitar 490 Hz. Di Uno dan mirip papan, pin 5 dan 6 memiliki frekuensi sekitar 980 Hz. Pin 3 dan 11 pada Leonardo juga berjalan pada 980 Hz.  
Fungsi analogWrite tidak ada hubungannya dengan pin analog atau fungsi analogRead.

**Due & Zero Only**

* **analogReadResolution()**

Merupakan perluasan dari API Analog untuk Arduino Due and Zero.  
Menetapkan ukuran (dalam bit) dari nilai yang dikembalikan oleh analogRead (). Standarnya 10 bit (mengembalikan nilai antara 0-1023) untuk kompatibilitas dengan papan berbasis AVR.  
The due and the Zero memiliki kemampuan ADC 12-bit yang dapat diakses dengan mengubah resolusi ke 12. Ini akan mengembalikan nilai dari analogRead () antara 0 dan 4095.

* **analogWriteResolution()**

Merupakan perluasan dari API Analog untuk Arduino Due, Genuino dan Arduino Zero dan MKR1000.  
AnalogWriteResolution () menetapkan resolusi fungsi analogWrite (). Standarnya 8 bit (nilai antara 0-255) untuk kompatibilitas dengan papan berbasis AVR.

**Advanced I/O**

* **tone()**

Menghasilkan gelombang persegi frekuensi yang ditentukan (dan 50% siklus) pada pin. Sebuah durasi dapat ditentukan, dinyatakan gelombang terus sampai panggilan untuk noTone(). pin dapat dihubungkan ke buzzer piezo atau pembicara lain untuk bermain nada.

Penggunaan fungsi tone() akan mengganggu output PWM pada pin 3 dan 11 (pada papan selain Mega).

* **noTone()**

Menghentikan generasi gelombang persegi dipicu oleh tone(). Tidak berpengaruh jika ada nada yang dihasilkan.

* **shiftOut()**

Bergeser keluar byte data satu bit pada satu waktu. Mulai dari salah sebagian (yaitu paling kiri) atau setidaknya (paling kanan) bit signifikan. Setiap bit ditulis pada gilirannya ke pin data, setelah mana pin jam berdenyut (diambil tinggi, maka rendah) untuk menunjukkan bahwa bit yang tersedia.  
Catatan: jika Anda berinteraksi dengan perangkat yang clock oleh meningkatnya tepi, Anda harus memastikan bahwa jam pin rendah sebelum panggilan untuk shiftOut(), misalnya dengan panggilan untuk digitalWrite (clockPin, LOW).

* **shiftIn()**

Pergeseran dalam byte data satu bit pada satu waktu. Mulai dari salah sebagian (yaitu paling kiri) atau setidaknya (paling kanan) bit signifikan. Untuk setiap bit, jam pin ditarik tinggi, bit berikutnya dibaca dari baris data, dan kemudian jam pin diambil rendah.  
Jika Anda berinteraksi dengan perangkat yang clock oleh meningkatnya tepi, Anda harus memastikan bahwa jam pin rendah sebelum panggilan pertama yang shiftIn (), misalnya dengan panggilan untuk digitalWrite (clockPin, LOW).  
Catatan: ini adalah implementasi perangkat lunak; Arduino juga menyediakan perpustakaan SPI yang menggunakan implementasi hardware, yang lebih cepat tetapi hanya bekerja pada pin tertentu.

* **pulseIn()**

Membaca pulsa (baik high atau low) pada pin. Misalnya, jika nilai adalah HIGH, pulseIn() menunggu untuk pin untuk pergi HIGH, dimulai waktu, kemudian menunggu untuk pin untuk pergi LOW dan berhenti waktu. Mengembalikan panjang dari pulsa di mikrodetik atau 0 jika tidak ada pulsa yang lengkap diterima dalam batas waktu.  
Waktu fungsi ini telah ditentukan secara empiris dan mungkin akan menunjukkan kesalahan dalam pulsa pendek. Bekerja pada pulsa dari 10 mikrodetik sampai 3 menit. Perlu diketahui juga bahwa jika pin sudah tinggi saat fungsi dipanggil, ia akan menunggu pin untuk pergi LOW dan kemudian HIGH sebelum mulai menghitung. rutin ini dapat digunakan hanya jika interupsi diaktifkan. Selain resolusi tertinggi diperoleh dengan interval pendek.

**Time**

**millis()**

Mengembalikan jumlah milidetik sejak papan Arduino mulai menjalankan program saat ini. Jumlah ini akan naik (kembali ke nol), setelah sekitar 50 hari.

**mikro()**

Mengembalikan jumlah mikrodetik sejak papan Arduino mulai menjalankan program saat ini. Jumlah ini akan naik (kembali ke nol), setelah sekitar 70 menit. Pada 16 MHz papan Arduino (seperti Duemilanove dan Nano), fungsi ini memiliki resolusi empat mikrodetik (yaitu nilai yang dikembalikan selalu kelipatan empat). Pada 8 MHz papan Arduino (seperti LilyPad), fungsi ini memiliki resolusi delapan mikrodetik.

Catatan: ada 1.000 mikrodetik dalam 1 milidetik dan 1.000.000 mikrodetik dalam 1 detik.

**delay()**

Jeda program untuk jumlah waktu (dalam milidetik) ditetapkan sebagai parameter. (Ada 1000 milidetik dalam 1 detik.)

**delayMicroseconds()**

Jeda program untuk jumlah waktu (dalam mikrodetik) ditetapkan sebagai parameter. Ada 1000 mikrodetik dalam 1 milidetik, dan 1 juta mikrodetik dalam 1 detik.

Saat ini, nilai terbesar yang akan menghasilkan delay yang akurat adalah 16383. Ini bisa berubah di rilis masa Arduino. Untuk keterlambatan lebih dari beberapa ribu mikrodetik, anda seharusnya menggunakan delay () sebagai gantinya.

**Math**

**min(x,y)**

Menghitung nilai minimal dari 2 angka.

**max(x,y)**

Menghitung nilai maksimal dari 2 angka.

**abs(x)**

Menghitung nilai pasti dari sebuah angka.

**constrain(x, a, b)**

Membatasi sejumlah angka yang berada dalam jangkauan.

**map(value, fromLow, fromHigh, toLow, toHigh)**

Memetakan kembali nomor dari satu rentang ke yang lain. Artinya, **nilai** **fromLow** akan mendapatkan dipetakan ke **toLow**, nilai dari **fromHigh** untuk **toHigh**, nilai di antara ke nilai diantara, dll

Tidak membatasi nilai-nilai ke dalam jangkauan, karena nilai out-of-range kadang-kadang yang dimaksudkan dan berguna. Fungsi constrain() dapat digunakan baik sebelum atau setelah fungsi ini, jika batas-batas rentang yang diinginkan.

Perhatikan bahwa "batas bawah" dari berbagai baik mungkin lebih besar atau lebih kecil dari "batas atas" sehingga map() fungsi dapat digunakan untuk membalikkan kisaran angka, misalnya

y = map(x, 1, 50, 50, 1);

Fungsi ini juga menangani angka negatif dengan baik, sehingga contoh ini

y = peta (x, 1, 50, 50, -100), juga berlaku dan bekerja dengan baik.

Fungsi map() menggunakan integer matematika sehingga tidak akan menghasilkan pecahan, ketika matematika mungkin menunjukkan bahwa itu harus melakukannya. Sisanya pecahan yang dipotong, dan tidak bulat atau rata-rata.

**pow(base, exponent)**

Menghitung nilai dari sejumlah dinaikkan ke power. Pow() dapat digunakan untuk meningkatkan angka ke power pecahan. Hal ini berguna untuk menghasilkan pemetaan eksponensial dari nilai-nilai atau kurva.

**sqrt(x)**

Menghitung akar kuadrat dari angka.

**Trigonometry**

**sin(rad)**

Menghitung sinus dari sudut (dalam radian). Hasilnya akan berada di antara -1 dan 1.

**cos(rad)**

Menghitung cos dari sudut (dalam radian). Hasilnya akan berada di antara -1 dan 1.

**tan(rad)**

Menghitung tangen dari sudut (dalam radian). Hasilnya akan antara infinity negatif dan tak terhingga.

**Character**

**isAlphaNumeric(thisChar)**

menganalisa jika character berupa alphanumeric

parameter thisChar: berupa character yang akan dianalisa

hasilnya berupa true atau false

**isAlphaNumeric(thisChar)**

menganalisa apakah char berupa Alpha

**isAscii(thisChar)**

menganalisa apakah char berupa ASCII

**isWhitespace(thisChar)**

menganalisa apakah char berupa White space

**isControl(thisChar)**

menganalisa apakah char dapat mengendalikan char

**isDigit(thisChar)**

menganalisa apakah char berupa digit.

**isGraph(thisChar)**

menganalisa apakah char dapat diprint.

**isLowerCase(thisChar)**

menganalisa apakah char berupa lower case.

**isPrintable(thisChar)**

menganalisa apakah char dapat diprint.

**isPunct(thisChar)**

menganalisa apakah char berupa tanda baca.

**isSpace(thisChar)**

menganalisa apakah char berupa spasi.

**isUpperCase(thisChar)**

menganalisa apakah char berupa uppercase.

**isHexadecimalDigit(thisChar)**

menganalisa apakah char berupa hexadecimal.

Ex

*/\*  
  Character analysis operators  
  
 Examples using the character analysis operators.  
 Send any byte and the sketch will tell you about it.  
  
 created 29 Nov 2010  
 modified 2 Apr 2012  
 by Tom Igoe  
  
 This example code is in the public domain.  
 \*/*  
  
void **setup**() {  
  *// Open serial communications and wait for port to open:*  
  Serial.begin(9600);  
  while (!Serial) {  
    ; *// wait for serial port to connect. Needed for native USB port only*  
  }  
  
  *// send an intro:*  
  Serial.println("send any byte and I'll tell you everything I can about it");  
  Serial.println();  
}  
  
void **loop**() {  
  *// get any incoming bytes:*  
  if (Serial.available() > 0) {  
    int thisChar = Serial.read();  
  
    *// say what was sent:*  
    Serial.print("You sent me: **\'**");  
    Serial.write(thisChar);  
    Serial.print("**\'**  ASCII Value: ");  
    Serial.println(thisChar);  
  
    *// analyze what was sent:*  
    if (isAlphaNumeric(thisChar)) {  
      Serial.println("it's alphanumeric");  
    }  
    if (isAlpha(thisChar)) {  
      Serial.println("it's alphabetic");  
    }  
    if (isAscii(thisChar)) {  
      Serial.println("it's ASCII");  
    }  
    if (isWhitespace(thisChar)) {  
      Serial.println("it's whitespace");  
    }  
    if (isControl(thisChar)) {  
      Serial.println("it's a control character");  
    }  
    if (isDigit(thisChar)) {  
      Serial.println("it's a numeric digit");  
    }  
    if (isGraph(thisChar)) {  
      Serial.println("it's a printable character that's not whitespace");  
    }  
    if (isLowerCase(thisChar)) {  
      Serial.println("it's lower case");  
    }  
    if (isPrintable(thisChar)) {  
      Serial.println("it's printable");  
    }  
    if (isPunct(thisChar)) {  
      Serial.println("it's punctuation");  
    }  
    if (isSpace(thisChar)) {  
      Serial.println("it's a space character");  
    }  
    if (isUpperCase(thisChar)) {  
      Serial.println("it's upper case");  
    }  
    if (isHexadecimalDigit(thisChar)) {  
      Serial.println("it's a valid hexadecimaldigit (i.e. 0 - 9, a - F, or A - F)");  
    }  
  
    *// add some space and ask for another byte:*  
    Serial.println();  
    Serial.println("Give me another byte:");  
    Serial.println();  
  }  
}

**Random Number  
 randomSeed()**

menginisialisasi generator pseudo-random nomor, yang menyebabkan memulai dari titik sembarang dalam urutan random. Urutan ini sangat panjang, dan random, tetapi selalu sama.

Ketika itu penting untuk urutan nilai yang dihasilkan oleh random() ke berbeda, pada eksekusi berikutnya dari sketsa, menggunakan randomSeed() untuk menginisialisasi nomor generator acak dengan inputan yang cukup random, seperti analogRead() pada pin yang tidak terhubung.

Sebaliknya, kadang-kadang dapat berguna untuk menggunakan urutan pseudo-random yang berulang-ulang. Hal ini dapat dilakukan dengan memanggil randomSeed() dengan nomor tetap, sebelum memulai urutan acak.

**random()**

fungsi random yang menghasilkan pseudo-random number

syntax

random(MAX)

random(MIN, MAX)

Ex

long randNumber;

void setup(){

Serial.begin(9600);

// if analog input pin 0 is unconnected, random analog

// noise will cause the call to randomSeed() to generate

// different seed numbers each time the sketch runs.

// randomSeed() will then shuffle the random function.

randomSeed(analogRead(0));

}

void loop() {

// print a random number from 0 to 299

randNumber = random(300);

Serial.println(randNumber);

// print a random number from 10 to 19

randNumber = random(10, 20);

Serial.println(randNumber);

delay(50);

}

**Bits and Bytes**

**lowByte()**

Mengekstrak low-order (paling kanan) byte dari dari variable (seperti kata).

**highByte()**

Mengeksrak high-order (paling kiri) dari variable (atau byte terendah kedua dari tipe data yang lebih besar).

**bitRead()**

Membaca bit dari sebuah nomor.

**bitWrite()**

Menulis bit dari sebuah variable angka.

**bitSet()**

Menetapkan (menulis 1 untuk) satu bit dari variable angka.

**bitClear()**

Menghapus (menulis 0 untuk) satu bit dari variable angka.

**bit()**  
Menghitung nilai bit tertentu (bit 0 adalah 1, bit 1 adalah 2, bit 2 adalah 4, dll).

**External Interrupts**

**attachInterrupt()**

Menambahkan interrupt pada variable.

**detachInterrupt()**

Mematikan atau menonaktifkan interrupt yang telah terpasang atau ditambahkan pada variabel

**Interrupts**

**Interrupt()**

Memungkinkan kembali interupsi (setelah mereka telah dinonaktifkan oleh noInterrupts ()). Interupsi memungkinkan tugas-tugas penting tertentu terjadi di latar belakang dan diaktifkan secara default. Beberapa fungsi tidak akan bekerja saat interupsi dinonaktifkan, dan komunikasi yang masuk dapat diabaikan. Interupsi dapat sedikit mengganggu waktu kode, bagaimanapun, dan mungkin dinonaktifkan karena terutama bagian penting dari kode.

**noInterrupt()**

Menonaktifkan interupsi (dapat diaktifkan kembali dengan interupsi ()). Interupsi memungkinkan tugas-tugas penting tertentu terjadi di latar belakang dan diaktifkan secara default. Beberapa fungsi tidak akan bekerja saat interupsi dinonaktifkan, dan komunikasi yang masuk dapat diabaikan. Interupsi dapat sedikit mengganggu waktu kode, bagaimanapun, dan mungkin dinonaktifkan karena terutama bagian penting dari kode.

**Communication**

**Serial**komunikasi serial pada pin TX / RX menggunakan tingkat logika TTL (5V atau 3.3V tergantung yang tertera pada papan). Jangan menghubungkan pin ini secara langsung ke port serial RS232; mereka akan beroperasi +/- 12V dan dapat merusak papan Arduino Anda.

Serial digunakan untuk berkomunikasi antara papan Arduino dan sebuah komputer atau perangkat lain. Semua papan Arduino memiliki setidaknya satu serial port (dikenal sebagai UART atau USART): Serial. Berkomunikasi pada digital pin 0 (RX) dan 1 (TX) serta dengan komputer melalui USB. Jadi, jika Anda menggunakan fungsi-fungsi ini, Anda tidak dapat menggunakan pin 0 dan 1 untuk input atau output digital.

Anda dapat menggunakan built-in monitor seri lingkungan Arduino untuk berkomunikasi dengan papan Arduino. Klik tombol Monitor seri pada toolbar dan pilih baud rate yang sama digunakan dalam panggilan untuk begin().

Arduino mega memiliki tiga port serial tambahan: serial1 pada pin 19 (RX) dan 18 (TX), Serial2 pada pin 17 (RX) dan 16 (TX), Serial3 pada pin 15 (RX) dan 14 (TX). Untuk menggunakan pin ini untuk berkomunikasi dengan komputer pribadi Anda, Anda akan memerlukan sebuah adaptor USB-to-serial tambahan, karena mereka tidak terhubung ke adaptor Mega USB-to-serial. Untuk menggunakannya untuk berkomunikasi dengan TTL perangkat serial eksternal, menghubungkan pin TX untuk perangkat RX pin, RX untuk perangkat TX pin, dan ground Mega ke ground perangkat Anda.

Arduino memiliki tiga tambahan 3.3V TTL serial port: serial1 pada pin 19 (RX) dan 18 (TX); Serial2 pada pin 17 (RX) dan 16 (TX), Serial3 pada pin 15 (RX) dan 14 (TX). Pin 0 dan 1 juga terhubung ke yang sesuai pin dari ATmega16U2 USB-to-TTL Serial chip, yang terhubung ke port USB debug. Selain itu, ada port USB-serial asli pada chip SAM3X, SerialUSB '.

Papan Arduino Leonardo menggunakan serial1 untuk berkomunikasi melalui TTL (5V) serial pada pin 0 (RX) dan 1 (TX). Serial dicadangkan untuk USB CDC communication.

**Stream**

Adalah kelas dasar untuk karakter dan dasar biner stream. Hal ini tidak dipanggil secara langsung, tapi dipanggil setiap kali Anda menggunakan fungsi yang bergantung pada itu.

Stream mendefinisikan fungsi membaca di Arduino. Ketika menggunakan fungsionalitas inti yang menggunakan read() atau metode yang sama, Anda dapat dengan aman berasumsi itu dipanggil pada kelas Stream. Untuk fungsi seperti print(), Turunan stream dari Print class.

Beberapa library yang mengandalkan Stream:

[Serial](https://www.arduino.cc/en/Reference/Serial)

[Wire](https://www.arduino.cc/en/Reference/Wire)

[Ethernet Client](https://www.arduino.cc/en/Reference/Ethernet)

[Ethernet Server](https://www.arduino.cc/en/Reference/Ethernet)

[SD](https://www.arduino.cc/en/Reference/SD)

Functions

**available()**

available() mendapat jumlah byte yang tersedia pada stream. Ini hanya untuk byte yang telah tiba.

Fungsi ini merupakan bagian dari kelas Stream, dan disebut oleh kelas yang mewarisi dari itu (Wire, Serial, dll). Lihat halaman utama kelas Streaming untuk informasi lebih lanjut.

**read()**

read() membaca karakter dari stream yang masuk ke buffer.

**flush()**

flush() membersihkan buffer ketika semua karakter telah dikirim.

**find()**

find() membaca data dari stream sampai target string ditemukan dengan waktu yang ditentukan Fungsi returns true jika target string ditemukan, false jika timed out.

**findUntil()**

findUntil() membaca data dari stream sampai target string ditemukan dengan waktu yang ditentukan terminator string ditemukan.

**peek()**

Membaca byte dari file tanpa melanjutkan ke tahap berikutnya. Artinya, panggilan berturut-turut untuk peek() akan mengembalikan nilai yang sama, seperti yang akan panggilan berikutnya untuk read().

**readBytes()**

ReadBytes() membaca karakter dari stream ke dalam buffer. Fungsi berakhir jika panjang bytes telah terbaca, atau time out (lihat setTimeout()).

readBytes () mengembalikan jumlah byte ditempatkan dalam buffer. angka 0 berarti tidak ada data valid yang ditemukan.

**readBytesUntil()**   
readBytesUntil() membaca karakter dari stream ke dalam buffer. Fungsi berakhir jika karakter terminator terdeteksi, panjang bytes telah terbaca, atau time out (lihat setTimeout()). readBytesUntil() mengembalikan jumlah byte ditempatkan dalam buffer. Angka 0 berarti tidak ada data valid yang ditemukan.

**readString()**

ReadString() membaca karakter dari stream ke dalam string. fungsi berakhir jika time out (lihat setTimeout()).

**readStringUntil()**

readStringUntil() membaca karakter dari stream ke dalam string. Fungsi berakhir jika karakter terminator terdeteksi atau time out (lihat setTimeout()).

**parseInt()**

parseInt() mengembalikan valid pertama (panjang) nomor integer dari serial buffer. Karakter yang tidak integer (atau tanda minus) yang dilewati.

**parsefloat()**

parseFloat() mengembalikan valid pertama nomor floating point dari posisi saat ini. karakter awal yang tidak integer (atau tanda minus) dilewati. parseFloat() diakhiri oleh karakter pertama yang bukan angka floating point.

**setTimeout()**

setTimeout() menetapkan milidetik maksimum untuk menunggu stream data, standarnya ke 1000 milidetik. Fungsi ini merupakan bagian dari Stream class, dan dipanggil oleh kelas yang mewarisi dari itu (Wire, Serial, dll). Lihat halaman utama kelas Streaming untuk informasi lebih lanjut.

**USB (32u4 based boards and Due/Zero only)**

**Mouse dan keyboard Library**

Inti dari libraries ini mengijinkan papan dasar 32u4 atau papan Zero tampil sebagai mouse atau sebagai keyboard asli ke komputer yang terhubung.  
Hati-hati menggunakan Mouse dan Keyboard Library: jika Mouse atau Keyboard Library terus berjalan, akan sulit untuk memrogram papan Anda. Fungsi seperti Mouse.move() dan Keyboard.print() akan memindahkan kursor atau mengirim keystrokes ke komputer yang terhubung dan seharusnya hanya dipanggil ketika Anda siap untuk menangani mereka. Disarankan untuk menggunakan sistem kontrol untuk menggunakan fungsi ini, seperti saklar fisik atau hanya menanggapi masukan tertentu dari Anda yang dapat dikontrol.

Bila menggunakan Mouse atau Keyboard Library, mungkin yang terbaik untuk menguji output pertama menggunakan Serial.print(). Dengan cara ini, Anda dapat yakin Anda tahu apa nilai yang dilaporkan. Mengacu pada Mouse dan Keyboard contoh untuk beberapa cara untuk menangani hal ini.

**Mouse**

Fungsi mouse memungkinkan Leonardo, Micro, atau Due untuk mengontrol gerakan kursor pada komputer yang terhubung. Ketika memperbarui posisi kursor, itu selalu relatif terhadap lokasi kursor sebelumnya.

**Mouse.begin()**

Dimulai meniru mouse yang terhubung ke komputer. Begin() harus dipanggil sebelum mengendalikan komputer. Untuk mengakhiri kontrol, menggunakan Mouse.end ().

**Mouse.click()**

Mengirim klik sesaat ke komputer di lokasi kursor. Ini sama dengan menekan dan segera melepaskan tombol mouse.

Mouse.click() default untuk tombol kiri mouse.

PERINGATAN: Bila Anda menggunakan Mouse.click () perintah, Arduino mengambil alih mouse Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Sebuah tombol tekan untuk mengaktifkan kontrol negara mouse efektif.

**Mouse.end()**

Untuk mengakhiri control pada mouse

**Mouse.move()**

Memindahkan kursor pada komputer yang terhubung. Gerakan pada layar selalu relatif terhadap lokasi kursor saat ini. Sebelum menggunakan Mouse.move() Anda harus memanggil Mouse.begin()

PERINGATAN: Bila Anda menggunakan Mouse.move () perintah, Arduino mengambil alih mouse Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Sebuah tombol tekan untuk mengaktifkan kontrol negara mouse efektif.

**Mouse.press()**

Mengirim sebuah tombol yang ditekan dari komputer yang terhubung. Satu kali press setara dengan mengklik dan terus memegang tombol mouse. Sebuah tekan dibatalkan dengan Mouse.release(). Sebelum menggunakan Mouse.press(), Anda harus mulai komunikasi dengan Mouse.begin(). Mouse.press() default untuk tombol tekan kiri.

PERINGATAN: Bila Anda menggunakan perintah Mouse.press (), Arduino mengambil alih mouse Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Sebuah tombol tekan untuk mengaktifkan kontrol negara mouse efektif.

**Mouse.release()**

Mengirimkan pesan bahwa tombol sebelumnya ditekan (dipanggil melalui Mouse.press()) dilepaskan. Mouse.release() default untuk tombol kiri.

PERINGATAN: Bila Anda menggunakan perintah Mouse.release (), Arduino mengambil alih mouse Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Sebuah tombol tekan untuk mengaktifkan kontrol negara mouse efektif.

**Mouse.isPressed()**

Memeriksa status saat ini dari semua tombol mouse, dan akan ada laporan atau notifikasi jika ada yang ditekan atau tidak.

**Keyboard**

Fungsi Keyboard memungkinkan Leonardo, Micro, atau Due mengirim keystrokes ke komputer yang terhubung.

Catatan: Tidak semua karakter ASCII tersedia, terutama yang non-printing, dapat dikirim dengan Keyboard Library. Perpustakaan mendukung penggunaan tombol pengubah. tombol pengubah mengubah perilaku kunci lain ketika ditekan secara bersamaan. Lihat di sini untuk informasi tambahan mengenai kunci didukung dan penggunaannya.

**Keyboard.begin ()**  
 Ketika digunakan dengan papan Leonardo atau papan Due, Keyboard.begin() mulai meniru keyboard yang terhubung ke komputer. Untuk mengakhiri kontrol, menggunakan Keyboard.end().

**Keyboard.end()**

Menghentikan meniru keyboard pada komputer yang terhubung. Untuk memulai emulasi keyboard, menggunakan Keyboard.begin().

**Keyboard.press()**   
 Ketika dipanggil, Keyboard.press() berfungsi sebagai jika kunci ditekan dan ditahan pada keyboard Anda. Berguna ketika menggunakan tombol pengubah. Untuk mengakhiri tekan tombol, menggunakan Keyboard.release() atau Keyboard.releaseAll(). Hal ini diperlukan untuk memanggil Keyboard.begin() sebelum menggunakan press().

**Keyboard.print ()**

Mengirimkan keystroke ke komputer yang terhubung. Keyboard.print() harus dipanggil setelah diawali Keyboard.begin().

PERINGATAN: Bila Anda menggunakan perintah Keyboard.print(), Arduino mengambil alih keyboard Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Menggunakan aebuah pushbutton untuk mengaktifkan status kontrol keyboard agar lebih efektif.

**Keyboard.println()**

Mengirimkan keystroke ke komputer yang terhubung, diikuti oleh baris dan carriage return. Keyboard.println() harus dipanggil setelah memulai Keyboard.begin().

PERINGATAN: Bila Anda menggunakan perintah Keyboard.println(), Arduino mengambil alih keyboard Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Menggunakan aebuah pushbutton untuk mengaktifkan status kontrol keyboard agar lebih efektif.

**Keyboard.release()**

Ketika dipanggil, Keyboard.release() berfungsi untuk melepas atau mengakhiri dalam menggunakan Keyboard.press().

**Keyboard.releaseAll()**

Ketika dipanggil, Keyboard.release() berfungsi untuk melepas atau mengakhiri semua key yang terpasang dalam menggunakan Keyboard.press().

**Keyboard.write()**

Mengirimkan keystroke ke komputer yang terhubung. Hal ini mirip dengan menekan dan melepaskan tombol pada keyboard Anda. Anda dapat mengirim beberapa karakter ASCII atau pengubah keyboard tambahan dan tombol khusus. Hanya karakter ASCII yang ada di keyboard yang didukung. Misalnya, ASCII 8 (backspace) akan bekerja, tetapi ASCII 25 (Pergantian) tidak.

Saat mengirim huruf, Keyboard.write() mengirimkan perintah pergeseran ditambah karakter yang diinginkan, sama seperti jika mengetik pada keyboard. Jika pengiriman tipe numerik, ia mengirimkannya sebagai karakter ASCII (ex. Keyboard.write (97) akan mengirimkan 'a'). Untuk daftar lengkap dari karakter ASCII, lihat ASCIITable.com.

PERINGATAN: Bila Anda menggunakan perintah Keyboard.write(), Arduino mengambil alih keyboard Anda! Pastikan Anda memiliki kontrol sebelum Anda menggunakan perintah. Menggunakan aebuah pushbutton untuk mengaktifkan status kontrol keyboard agar lebih efektif.

**REFERENSI**

https://www.arduino.cc/en/Reference/HomePage