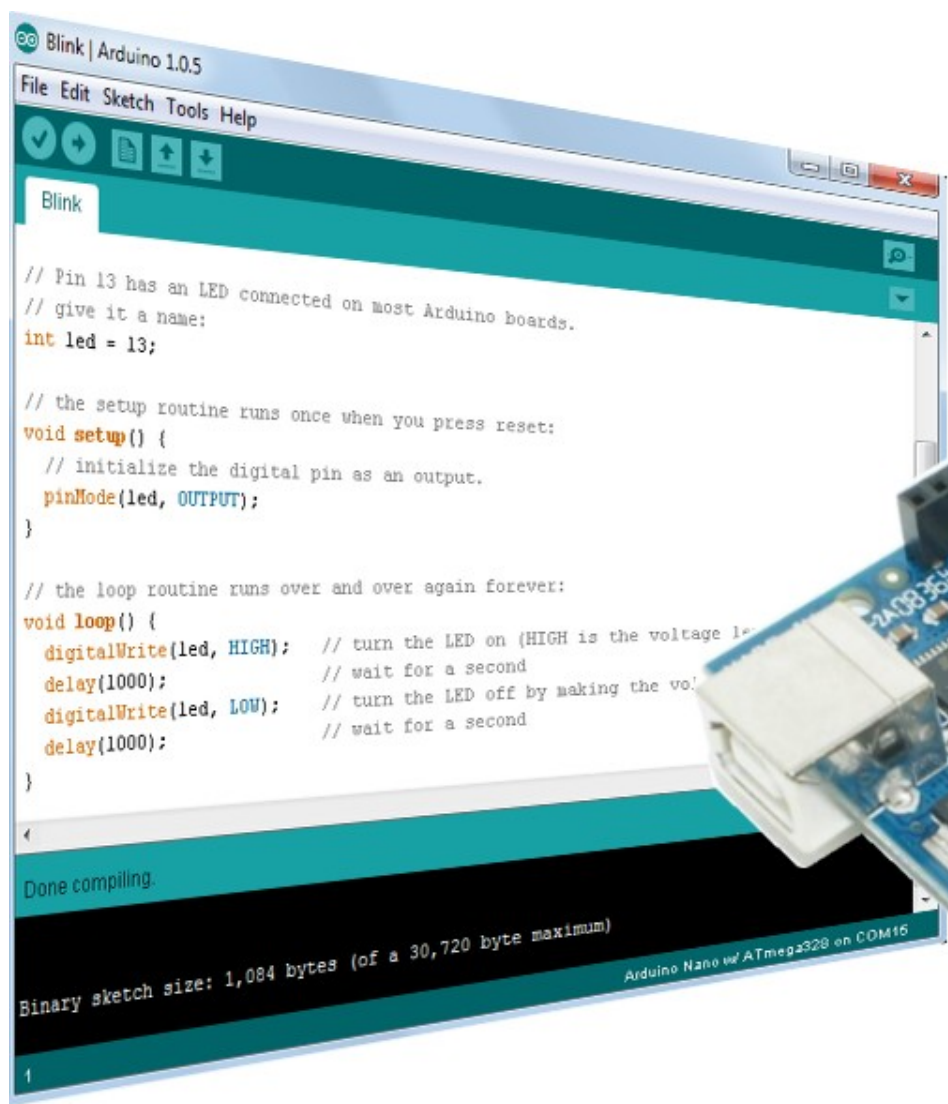


Arduino tanfolyam kezdőknek és haladóknak



2. Változók típusai, feltételvizsgálat, programelágazás

Változók

- Programfutás alatt a mért vagy számított adatokat ideiglenesen el kell valahova helyezni, erre a RAM memóriában kell helyet foglalni
- A lefoglalt helyeket névvel látjuk el, ezzel hivatkozhatunk rájuk, amikor újra elővesszük, vagy felülírjuk az adatot
- A RAM memória csak a kikapcsolásáig őrzi meg az adatokat
- A változókat felfoghatjuk úgy, mint egy sokfiókos doboz (a memória) egy-egy fiókját, amibe egyvalamit beletehetünk, vagy kivehetünk
- A változó neve kötelezően kis- vagy nagybetűvel kezdődjön de tartalmazhat számjegyet vagy aláhúzásjelet is



Változók típusai

- A legfontosabb változó típusok az alábbi táblázatban láthatók

Típus	Méret [bájt]	Min. érték	Max. érték	STDINT típus (ANSI C90)
boolean	1	false (0)	true (1)	
char	1	-128	127	int8_t
unsigned char <i>vagy byte</i>	1	0	255	uint8_t
int	2	-32 768	32 767	int16_t
unsigned int <i>vagy word</i>	2	0	65 535	uint16_t
long	4	-2,147,483,648	2,147,483,647	int32_t
unsigned long	4	0	4,294,967,295	uint32_t
float	4	-3.4028235E+38	3.4028235E+38	
double	erőforrás hiányában ugyanaz, mint a float típus			

Változó deklaráció és értékadás

- Változó deklarálásánál meg kell adni a típusát és a változó nevét például: `int a, b, c;` három előjeles egész változót hoz létre
- Deklaráláskor nem kötelező, de megadhatunk kezdőértéket is például: `int a = 2, b = -3, c = 120;`
A **const** előtag konstans értéket deklarál, pl: `const int RED_LED = 13;`
- Az Arduino C/C++ környezete azt is megengedi, hogy a változó deklarációja ne a program elején, hanem az első használatbavétel helyén legyen, mint például az alábbi programrészletben

```
void setup() {
  pinMode(3, INPUT_PULLUP);    // D3 digitális bemenet, belső felhúzással
}

void loop() {
  boolean sw = digitalRead(3); // Nyomógomb állapotának beolvasása
  int adc = analogRead(A0);    // ADC csatorna beolvasása
  float temp = (adc*4750.0/1024-500)/10.0; //Adatok átszámítása
  ...
}
```

Túlcsordulás, számábrázolási pontosság

- **Túlcsordulás** akkor következik be, ha a változóban maximálisan ábrázolható számot meghaladja a művelet eredménye

```
byte a = 255;
```

```
a = a + 1; //nem 256, hanem 0 lesz!
```

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ \hline \end{array} = 255_{10}$$

$$+ \begin{array}{r} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\ \hline \end{array} = 1$$

$$\begin{array}{r} \textcircled{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ \hline \end{array} = 0 \text{ (8 biten, mivel a 9. bit elveszett)}$$

- Lebegőpontos (float) típusú változóknál a **számábrázolási pontosságot** két tényező korlátozza:

- ❖ Az értékes jegyek száma kb. 6-7, így például

`float a = 3.1234567E10;` esetén `a` és `a+1` nem lesz különböző

- ❖ Bizonyos számok (például 0.1) kettes számrendszerben nem ábrázolható véges számú számjeggyel – ahogyan pl. `1/3` vagy `1/7` sem ábrázolhatók véges tizedestörttel



Aritmetikai műveletek

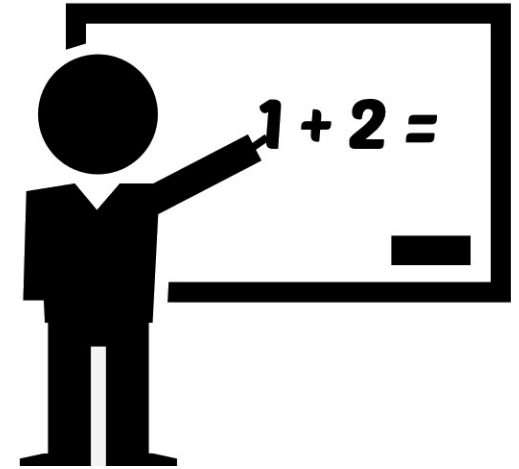
- Számítási (aritmetikai) műveletek

`a = a + 5; // összeadás`

`a = c - b; // kivonás`

`a = a * 7; // szorzás`

`a = b / 2; // osztás`



- Fontos, hogy milyen típusú változót használunk, mert egész típusoknál $10 / 4$ eredménye nem 2.5, hanem 2 lesz!

- **Osztási maradék** előállítása a `%` művelettel történik

Legyen pl. $a = 15$, akkor az $a/6$ osztás eredménye 2 lesz, az $a\%6$ művelet eredménye pedig 3 lesz (mivel $15 = (2 * 6) + 3$)

- Összevonások (rövidebb jelölésmód)

`x++ // ugyanaz, mint: x = x+1, x-- // ugyanaz, mint: x = x - 1`

`x += y // ugyanaz, mint: x = x + y, x -= y // ugyanaz, mint: x = x - y`

`x *= y // ugyanaz, mint: x = x * y, x /= y // ugyanaz, mint: x = x / y`

valtozok.ino

- Az alábbi programban egyszerű példákat mutatunk a változók használatára

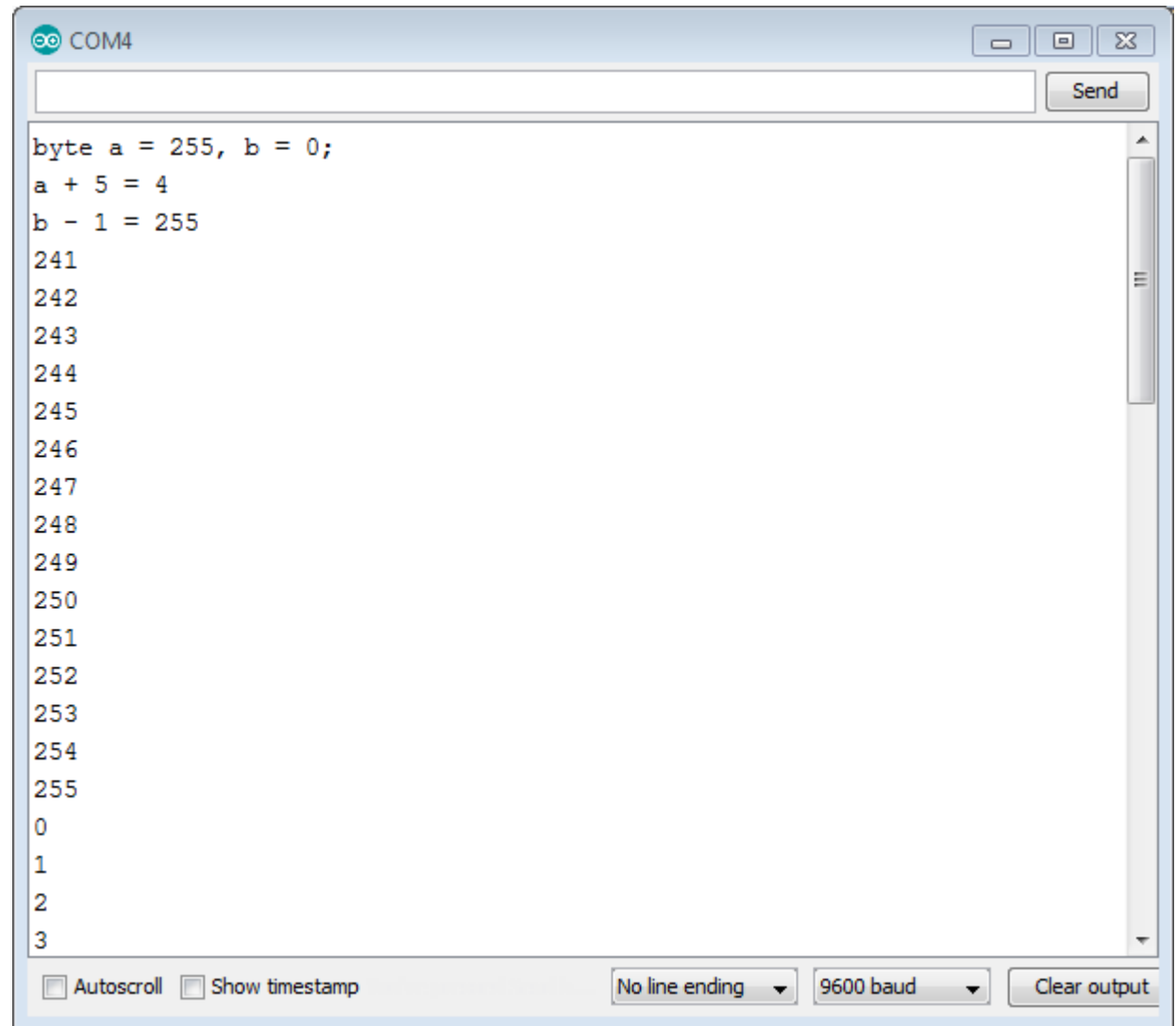
```
byte szamlalo = 240;           // szamlalo változó deklarálása kezdőértéke 240
byte a = 255, b = 0;

void setup() {
  Serial.begin(9600);         // A soros porton fogunk kiíratni
  Serial.println("byte a = 255, b = 0;");
  Serial.print("a + 5 = ");
  a = a + 5;                  // Túlcsordulás: 255 + 5 = 4 lesz (260 - 256)
  Serial.println(a);
  Serial.print("b - 1 = ");
  b = b - 1;                  // Alulcsordulás: 0 - 1 = 255 lesz (256 - 1)
  Serial.println(b);
}

void loop() {
  szamlalo++;                 // Egy hozzáadása minden ciklusban
  Serial.println(szamlalo);   // Kiíratjuk szamlalo aktuális tartalmát
}
```

szamlalo.ino futási eredménye

- A program eredményének megtekintéséhez meg kell nyitnunk a terminál ablakot
- **a+5** túlcsorduláshoz vezetett
- **b-1** alulcsorduláshoz vezetett
- A **szamlalo** változó növelgetése 255 elérése után túlcsordulást okoz



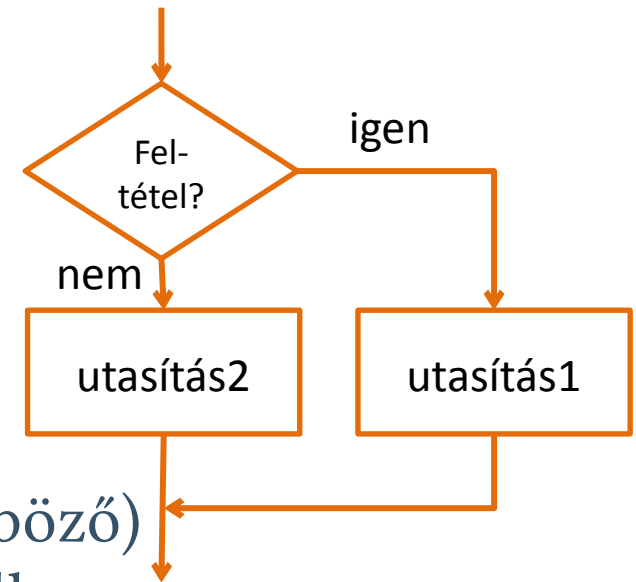
```
COM4
byte a = 255, b = 0;
a + 5 = 4
b - 1 = 255
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
0
1
2
3
```


Feltételes elágazás: if – else utasítás

- Megvizsgál egy feltételt, s ezt követően a végrehajtás menete a feltételvizsgálat eredményétől függ

```
if(feltétel) utasítás1  
else utasítás2      (az else ág elhagyható)
```

- A **feltétel** egy kifejezés, ami aritmetikai és relációs operátorokat is tartalmazhat. A kiértékelés eredménye
igaz: ha a kifejezés értéke **true** (0-tól különböző)
hamis: ha a kifejezés értéke **false**, vagy nulla.



- Ha a **feltétel teljesül**, akkor **utasítás1** kerül végrehajtásra, **egyébként** pedig **utasítás2** lesz végrehajtva. Például:

Ha PUSH2 nincs lenyomva

```
if(digitalRead(PUSH2)) { digitalWrite(RED_LED,LOW); }  
else { digitalWrite(RED_LED,HIGH); } //Ha a gomb le van nyomva, a LED világít
```

Ha PUSH2 le van nyomva

PUSH2 olvasása **1**, ha nincs lenyomva, **0**, ha le van nyomva

Relációs és logikai operátorok

- Logikai kifejezésekben, illetve feltételvizsgálatoknál az alábbi operátorok (műveletek) állnak rendelkezésre:

Relációs operátorok

`==` egyenlő pl. `x == y`
`!=` nem egyenlő pl. `x != y`
`<` kisebb mint pl. `x < y`
`>` nagyobb mint pl. `x > y`
`<=` kisebb, vagy egyenlő
`>=` nagyobb, vagy egyenlő

Logikai operátorok

Logikai változókon vagy logikai kifejezéseken végzett műveletekhez használhatók

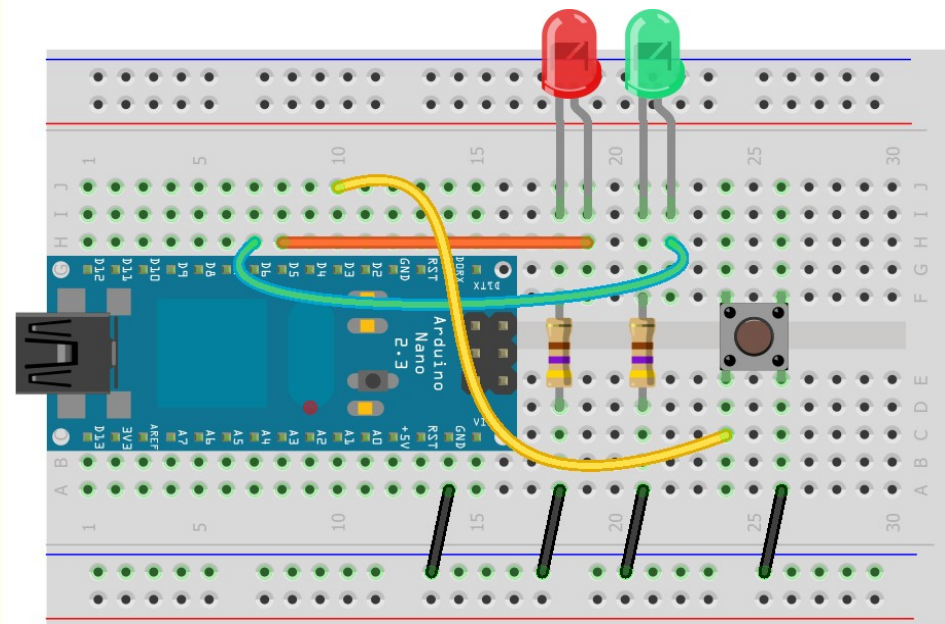
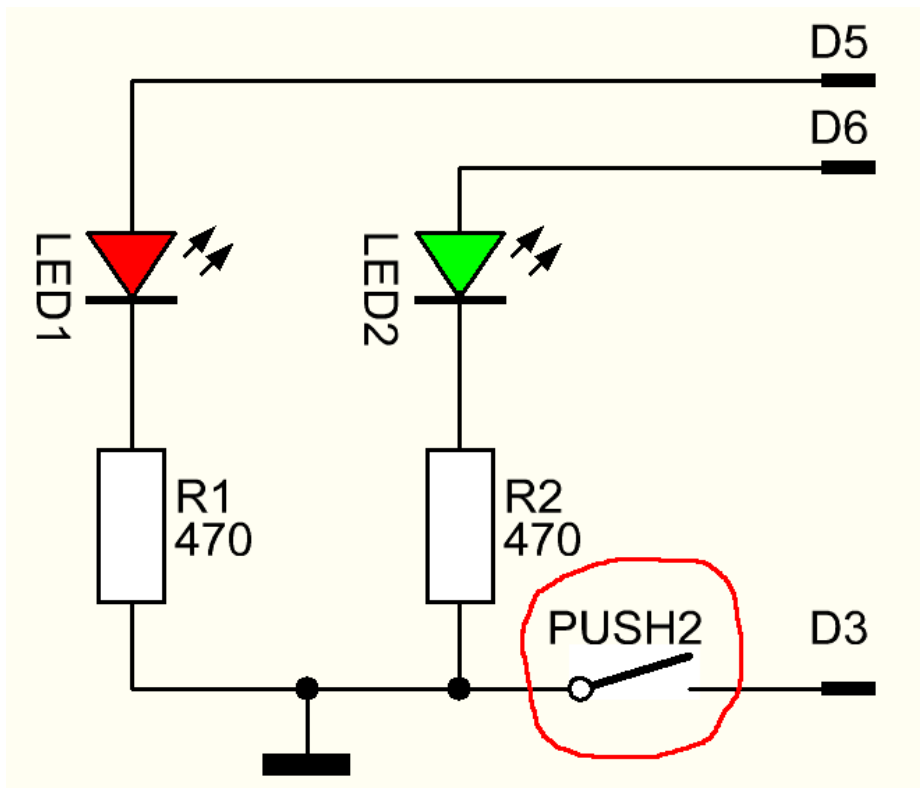
`!` Tagadás (negáció) pl. `!a`
`&&` ÉS művelet pl. `a && b`
`||` VAGY művelet pl. `a || b`

- **Megjegyzések:**

- ❖ A relációs, illetve a logikai műveletek eredménye egy logikai (boolean) mennyiség, ami csak 0 (ha hamis) vagy 1 (ha igaz) értéket vehet fel.
- ❖ A C fordítók kiterjesztett értelmezéssel bármilyen 0-tól különböző számot is elfogadnak igaz értéknek. Így például `if(x != 0)` helyett `if(x)` -et is írhatunk.
- ❖ Ne keverjük össze a logikai operátorokat a bitenként logikai operátorokkal!
Bitenkénti műveletek: `~`, `&`, `|`, `^` Logikai műveletek: `!`, `&&`, `||`

blink_control.ino

- Villogtassunk két LED-et felváltva, de a villogás frekvenciája (az átkapcsolások közötti időtartam) a nyomógomb állapotától függjön:
 - ❖ Ha a gombot elengedjük, 500 ms-onként váltunk (1 Hz villogás)
 - ❖ Ha a gombot lenyomjuk, 100 ms-onként váltunk (5 Hz villogás)
- A kapcsolás legyen ugyanaz, mint az előző foglalkozáson!



Made with  Fritzing.org

blink_control.ino

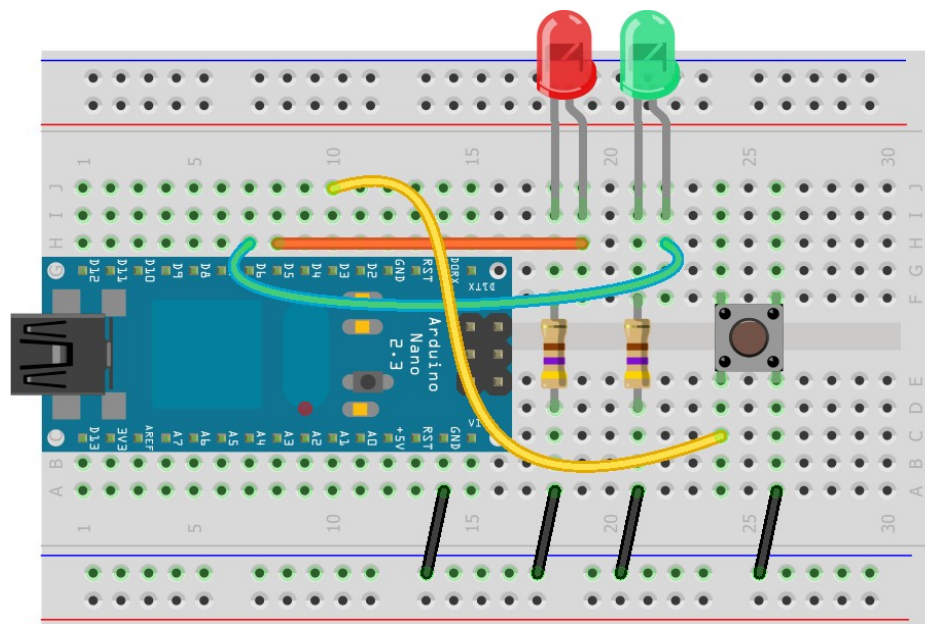
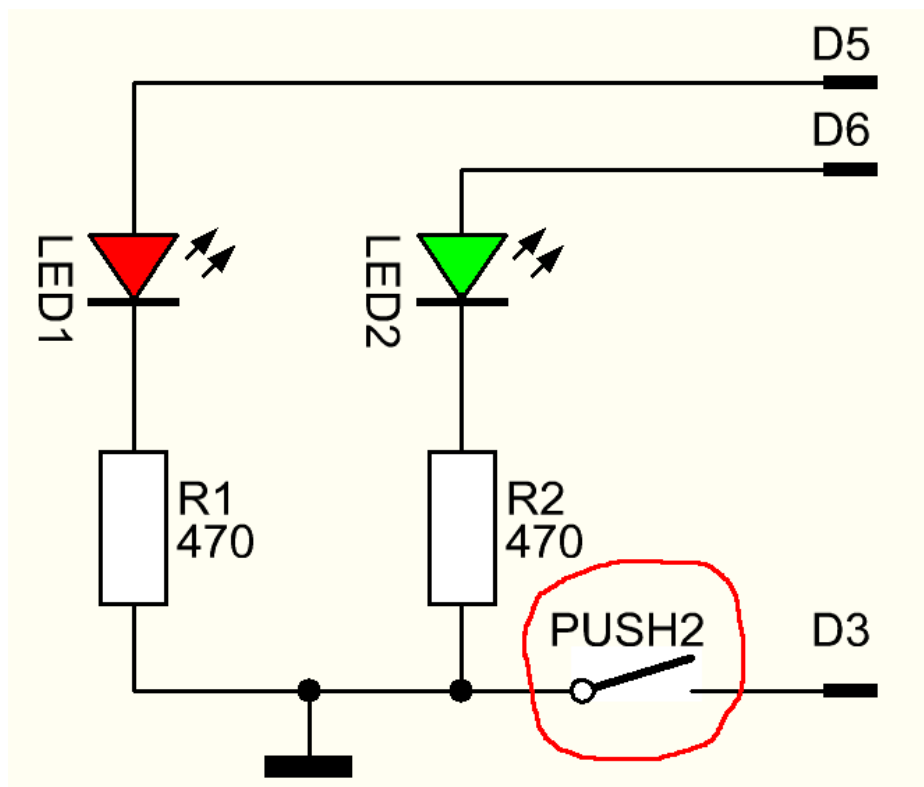
```
#define RED_LED    5
#define GREEN_LED 6
#define PUSH2     3

void setup() {
    pinMode(RED_LED,OUTPUT);    // legyen kimenet
    pinMode(GREEN_LED,OUTPUT); // legyen kimenet
    pinMode(PUSH2,INPUT_PULLUP); // legyen bemenet, belső felhúzással
}

void loop() {
    boolean sw;                // Ebben tároljuk a nyomógomb állapotát
    sw = digitalRead(PUSH2);   // Beolvassuk a nyomógomb állapotát
    digitalWrite(RED_LED,HIGH); // A piros LED világít
    digitalWrite(GREEN_LED,LOW); // A zöld LED nem világít
    if(sw == HIGH) delay(400); // Felengedett gomb esetén HIGH állapot van
    delay(100);                // A 100 ms késleltetés mindig kell
    digitalWrite(RED_LED,LOW); // A piros LED nem világít
    digitalWrite(GREEN_LED,HIGH); // A zöld LED világít
    if(sw == HIGH) delay(400); // Felengedett gomb esetén több késleltetés
    delay(100);                // 100 ms késleltetés mindig kell
}
```

Nyomógomb vizsgálata: button_blinker.ino

- Vizsgáljuk a **D3**-ra kötött nyomógomb állapotát **if()** utasítással!
- Villogtassuk a **D5**-re kötött LED-et, amikor a gomb le van nyomva!
Az állapotváltások közötti idő legyen 100 ms!
- A kapcsolás lehet ugyanaz, mint a 11. oldalon, de a zöld LED-et most nem használjuk



Made with  Fritzing.org

Nyomógomb vizsgálata: `button_blinker.ino`

- Mi az **else** ág szerepe ebben a programban?
- Mi történik, ha elhagyjuk az **else** ágot?

```
#define PUSH2      3           // a nyomógomb bemenet sorszáma
#define RED_LED    5           // a piros LED kimenet sorszáma
boolean ledState = LOW, buttonState; // Állapotjelzők tárolói

void setup() {
  pinMode(RED_LED, OUTPUT); // RED_LED legyen kimenet
  pinMode(PUSH2, INPUT_PULLUP); // Bemenet, belső felhúzással
}

void loop() {
  buttonState = digitalRead(PUSH2); // Nyomógomb állapot beolvasása
  if (buttonState == LOW) { // Ha a gomb le van nyomva
    ledState = !ledState; // LED állapot átbillentése
  } else { // különben (ha a gomb nincs lenyomva)
    ledState = LOW; // LED kikapcsolása
  }
  digitalWrite(RED_LED, ledState); // LED állapot megjelenítése
  delay(100);
}
```

A kapcsolás ugyanaz, mint a 11. oldalon

if utasítások egymásba ágyazása

Egymásba ágyazott **if-else** szerkezeteknél, hiányzó **else** ágak esetén nem világos, hogy a meglévő **else** ág melyik **if** utasításhoz tartozik. Például az

```
if (n > 0)
    if (a > b) z = a;
    else z = b;
```

programrészletben az **else** a belső **if** utasításhoz tartozik.

Általános szabály: az **else** mindig a hozzá legközelebb eső, **else** ág nélküli

if utasításhoz tartozik. Ha nem így szeretnénk, akkor használjunk kapcsos zárójeleket! Például:

```
if (n > 0) {
    if (a > b) z = a;
}
else z = b;
```

A nem egyértelmű helyzet különösen zavaró az olyan szerkezetekben, mint az alábbi:

```
if (n >= 0)
    for (i = 0; i < n; i++)
        if (s[i] > 0) {
            Serial.println("...");
        }
else /* ez így hibás */
    Serial.println("n értéke negatív");
```

A tagolás ellenére a fordító itt az **else** ágat a belső **if** utasításhoz kapcsolja.

Az ilyen, nehezen felderíthető hibák megelőzésére használjunk kapcsos zárójeleket a második **if** utasítás körül!

Az else – if szerkezet

Az **else if** szerkezet adja a többszörös elágazások programozásának egyik legáltalánosabb lehetőségét.

A szerkezet úgy működik, hogy a program sorra kiértékeli a *kifejezéseket* és ha bármelyik ezek közül igaz, akkor végrehajtja a megfelelő *utasítást*, majd befejezi az egész vizsgáló láncot.

Itt is, mint bárhol hasonló esetben, az *utasítás* helyén kapcsos zárójelek között elhelyezett blokk is állhat.

Az utolsó **else** ág alapértelmezés szerint a „*fentiek közül egyik sem*” esetet kezeli. Ha ilyenkor semmit sem kell csinálni, ez az ág elhagyható.

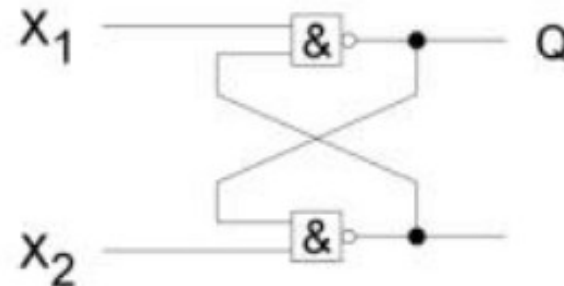
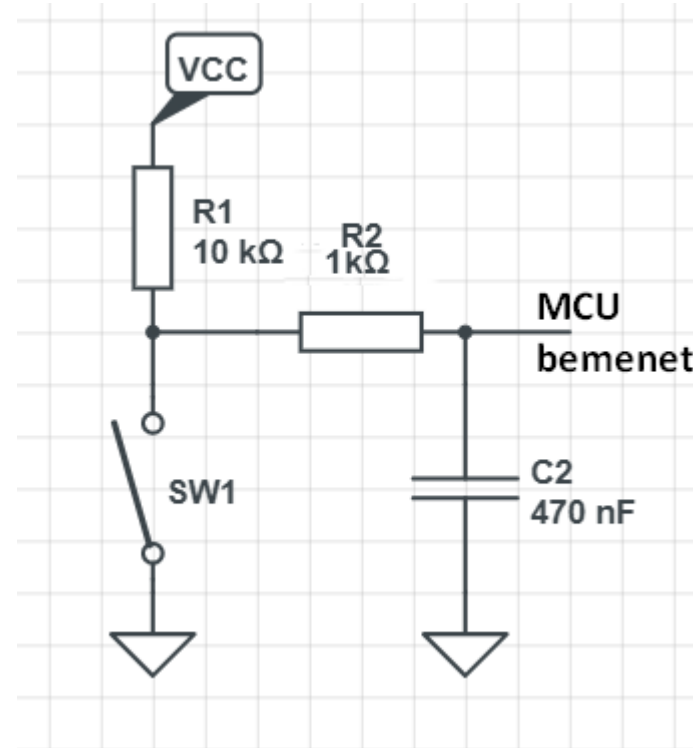
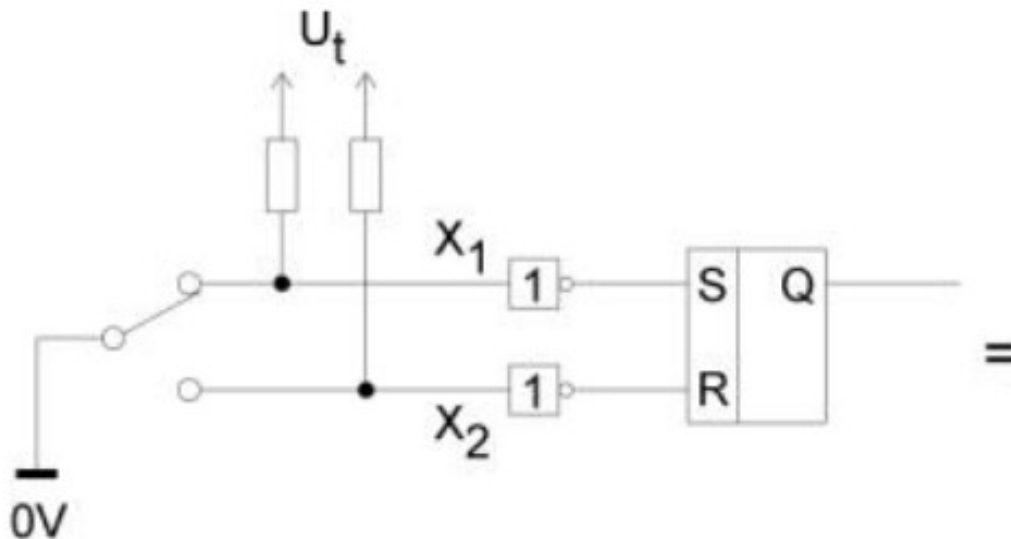
```
if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
.
.
.
else
    utasítás
```


Egy kis elektronika...

■ Nyomógomb, illetve kapcsoló hardveres pergésmentesítése

❖ Analóg módszer:
a jel integrálásával

❖ Digitális módszer:
S – R tárolóval



Időzítés – késleltetés nélkül

- Egy LED villogtatása késleltető függvényhívásokkal könnyen megvalósítható, de a program menetének megakasztása nem mindig elfogadható
- Mit tegyünk, ha több, különböző időzítést kívánó feladatot kell a programnak elvégeznie?
- A **millis()** függvényt fogjuk használni, amely a legutóbbi (újra-) indítástól számított futó idő értékét adja meg ezredmásodpercekben

```
unsigned long time;
```

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

```
void loop() {  
  Serial.print("Time: ");  
  time = millis();  
  Serial.println(time); // kiírja az indítás óta eltelt időt  
  delay(1000);          // tartunk egy kis szünetet, hogy ne írjunk túl sokat  
}
```

Egyszerű példa a **millis()** függvény használatára
(forrás: [Arduino Reference](#))

multitasking.ino

- Két LED-et villogtatunk, különböző frekvenciával
- A piros LED 1000 ms-onként vált állapotot, a zöld LED pedig 5000 ms-onként
- Mindig előre kiszámoljuk, és eltároljuk (redTime, greenTime változók) hogy mikor kerül sorra az adott esemény
- A ciklusban megnézzük, hogy mennyi az idő és összehasonlítjuk azt a feljegyzéseinkkel

```
#define RED_LED 5
#define GREEN_LED 6
unsigned long nowTime, redTime, greenTime;
boolean rState=LOW, gState=LOW;

void setup() {
  pinMode(RED_LED,OUTPUT);
  pinMode(GREEN_LED,OUTPUT);
  nowTime = millis();
  redTime = nowTime + 1000;
  greenTime = nowTime + 5000;
}
```

A kapcsolás ugyanaz, mint a 11. oldalon

```
void loop() {
  nowTime = millis();
  if(nowTime > redTime) {
    rState = !rState;
    digitalWrite(RED_LED,rState);
    redTime += 1000;
  }
  if(nowTime > greenTime) {
    gState = !gState;
    digitalWrite(GREEN_LED,gState);
    greenTime += 5000;
  }
}
```

Nyomógomb vezérelt LED villogtatás

- Bonyolítsuk az előző programot azzal, hogy a LED-ek villogási sebességét egy nyomógombbal lehessen szabályozni:
- Minden lenyomás felezze meg az átbillentési időket (az egyszerűség kedvéért egyformán mindkettőét)!
- Ha a piros LED átbillenési ideje már kisebb, mint 200 ms, akkor a következő gombnyomáskor állítsuk vissza a kiindulási átbillenési időket!
- Gondoljunk a nyomógomb pergésére! Egyszerű megoldásként csak 20 ms-onként vizsgáljuk meg a nyomógomb állapotát...

gombvezeret_ledek.ino

```
#define RED_LED 5
#define GREEN_LED 6
#define PUSH2 3
unsigned long nowTime, redTime, greenTime, buttonTime;
unsigned long redInterval = 1000, greenInterval = 5000;
boolean rState = HIGH, gState = HIGH, bState = HIGH;
```

```
void changeSpeed() {
  if (redInterval < 200) {
    greenInterval = 5000;
    redInterval = 1000;
  }
  else {
    greenInterval /= 2;
    redInterval /= 2;
  }
}
```

```
void setup() {
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(PUSH2, INPUT_PULLUP);
  nowTime = millis();
  redTime = nowTime + redInterval;
  greenTime = nowTime + greenInterval;
  buttonTime = nowTime + 20;
}
```

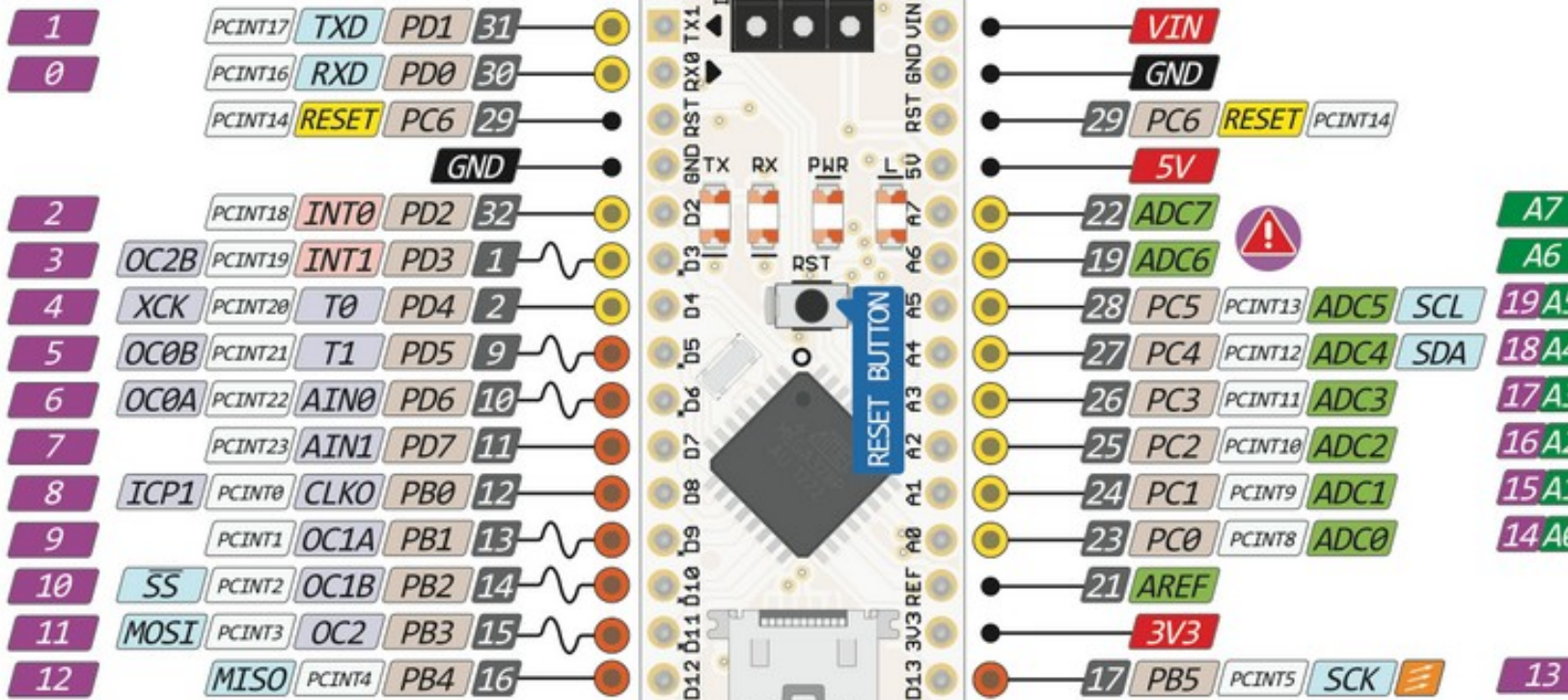
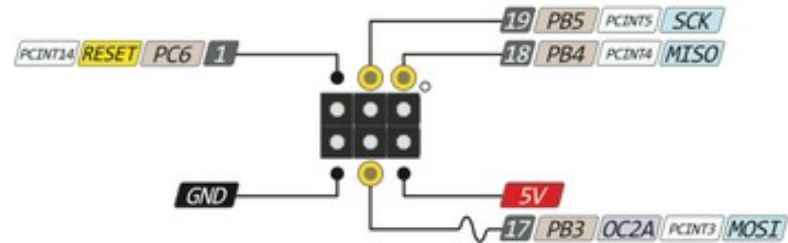
```
void loop() {
  nowTime = millis();
  if (nowTime > redTime) {
    rState = !rState;
    digitalWrite(RED_LED, rState);
    redTime = nowTime + redInterval;
  }
  if (nowTime > greenTime) {
    gState = !gState;
    digitalWrite(GREEN_LED, gState);
    greenTime = nowTime + greenInterval;
  }
  if (nowTime > buttonTime) {
    buttonTime = nowTime + 20;
    if (digitalRead(PUSH2) != bState) {
      bState = !bState;
      if (bState == LOW) changeSpeed();
    }
  }
}
```

Az Arduino nano kártya kivezetései



NANO PINOUT

The power sum for each pin's group should not exceed 100mA



- Power
- GND
- Serial Pin
- Analog Pin
- Control
- INT
- Physical Pin
- Port Pin
- Pin function
- Interrupt Pin
- PWM Pin
- Port Power

Absolute MAX per pin 40mA recommended 20mA

Absolute MAX 200mA for entire package

Analog exclusively Pins

Ellenállás színkódok

