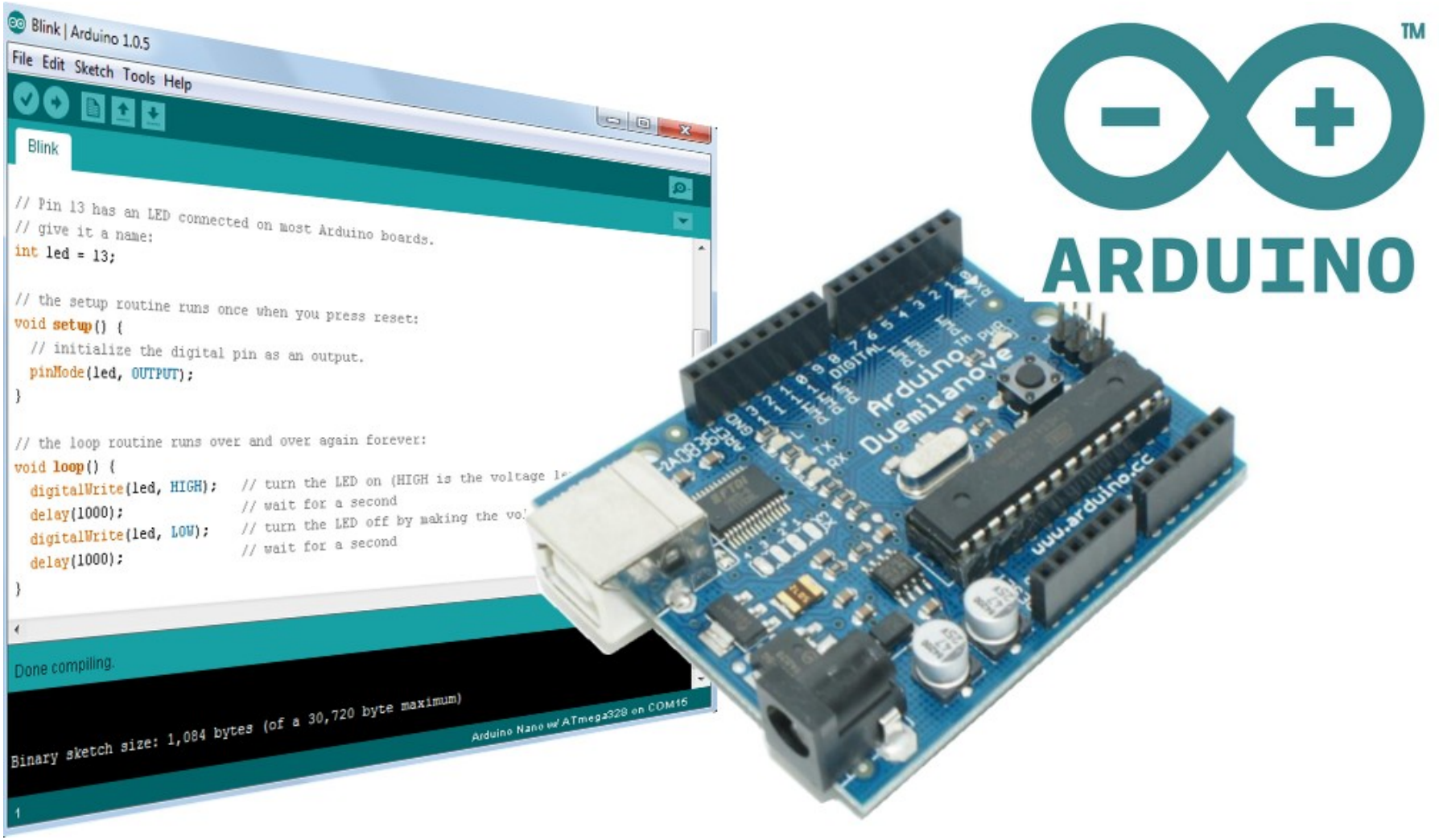
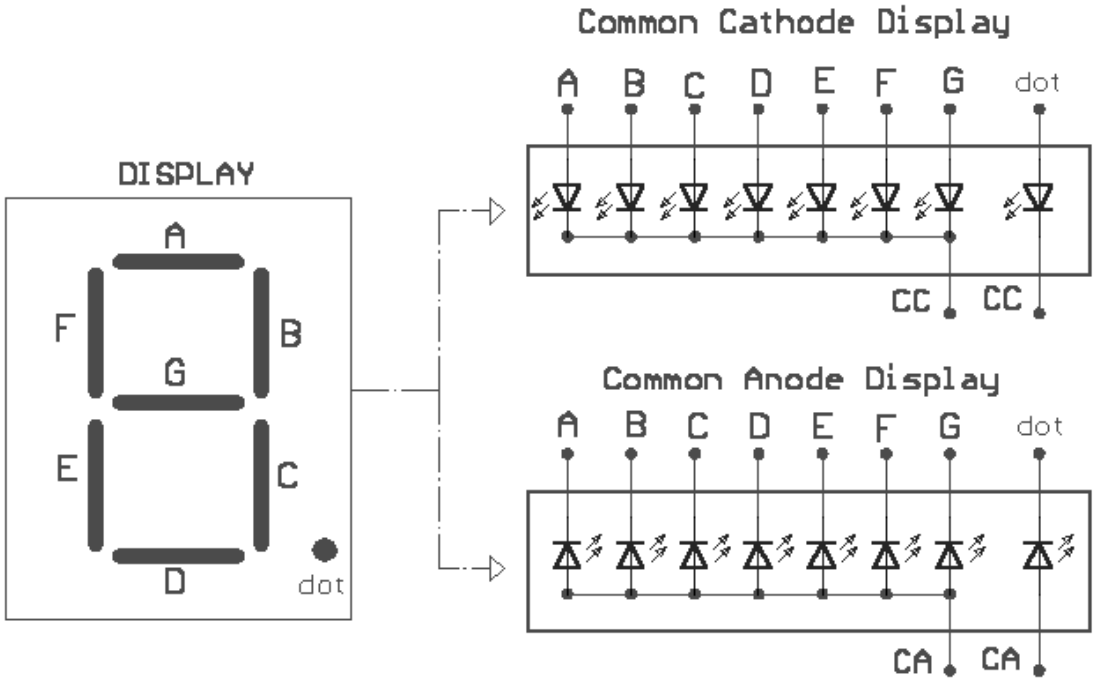
Bevezetés az elektronikába

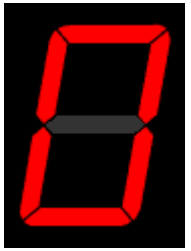


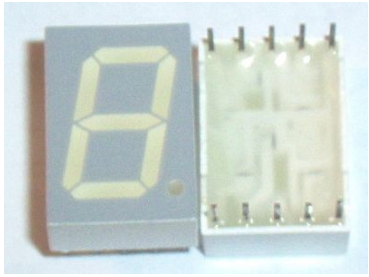
14. Arduino programozás – hétszegmenses kijelzők

A hétszegmenses LED kijelző

- A hétszegmenses kijelzők 7 db LED-et vagy LED csoportot tartalmaznak, olyan elrendezésben, hogy a 0…9 arab számjegyeket ki lehessen jelezni.
- A hét szegmenshez nyolcadikként többnyire egy tizedespont is járul.
- Kivitel: Közös anódú vagy közös katódú tokozás



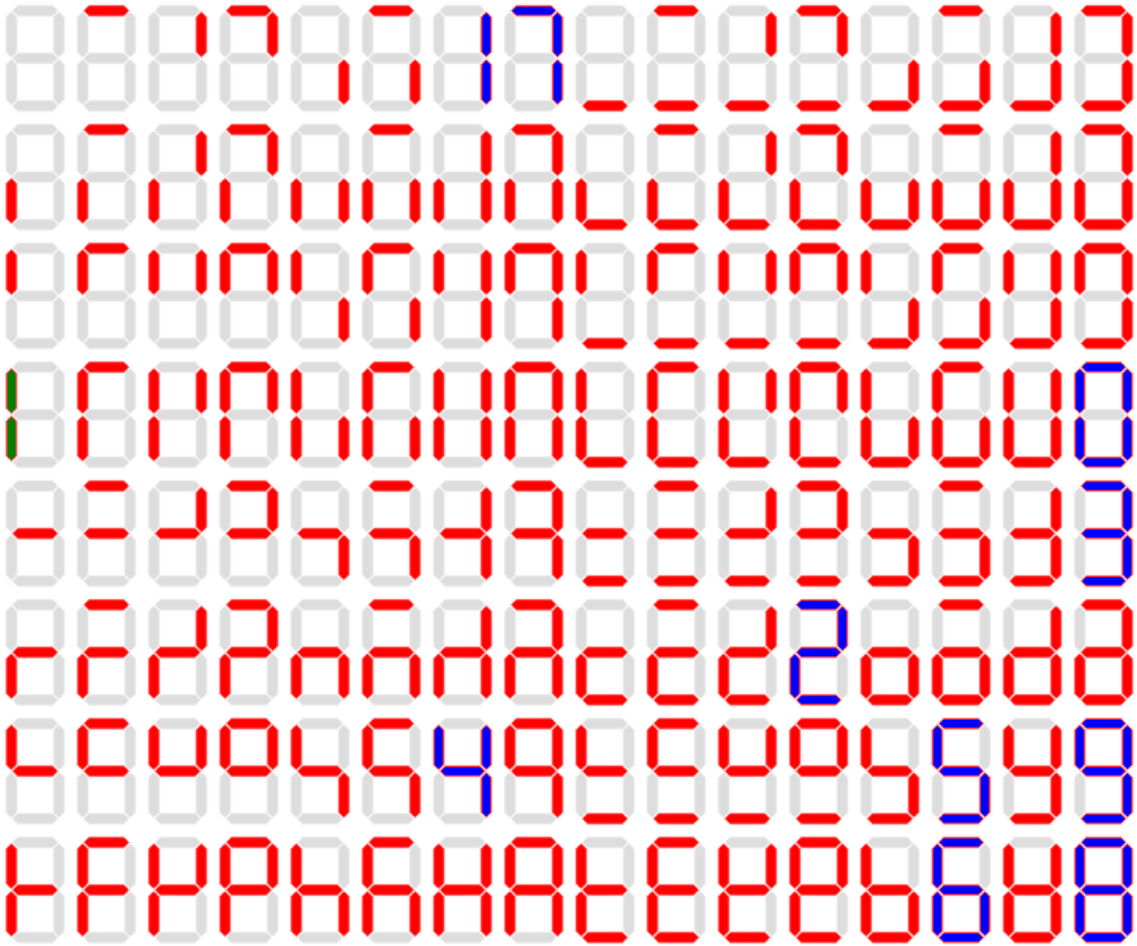




Hogyan működik a kijelzés?

- A 7 db szegmens mindegyike lehet ki- vagy bekapcsolt állapotban.

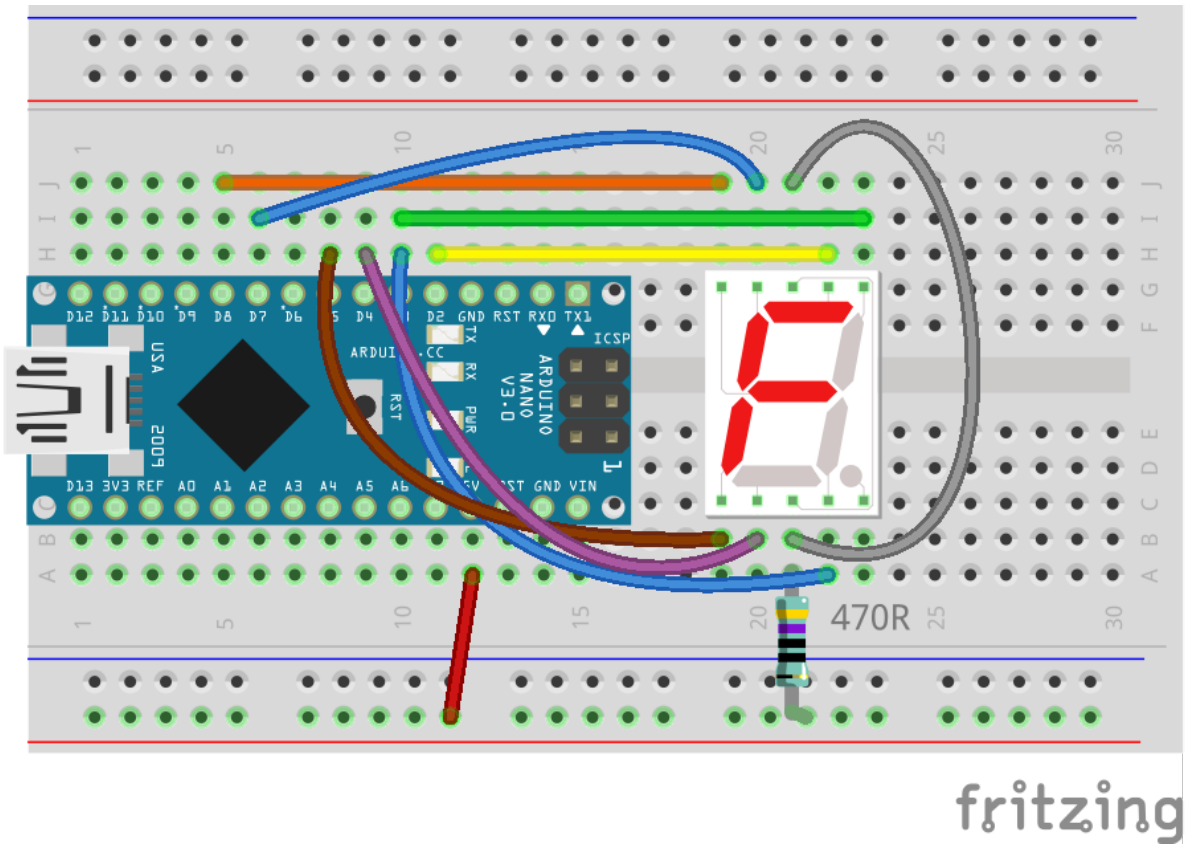
- A lehetséges állapotok száma: $2^7 = 128$
- Ezek közül nem mindegyik értelmes kombináció.



- **Házi feladat:**
  Vadásszunk betűkre!
  A mellékelt ábrából keressünk olyan kombinációkat, amelyek kis- vagy nagybetűt mutatnak!

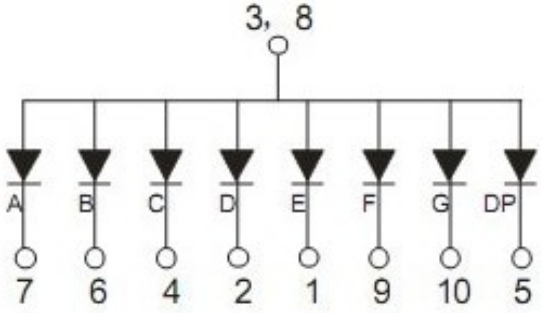
A „lusta kapcsolás” és hátrányai

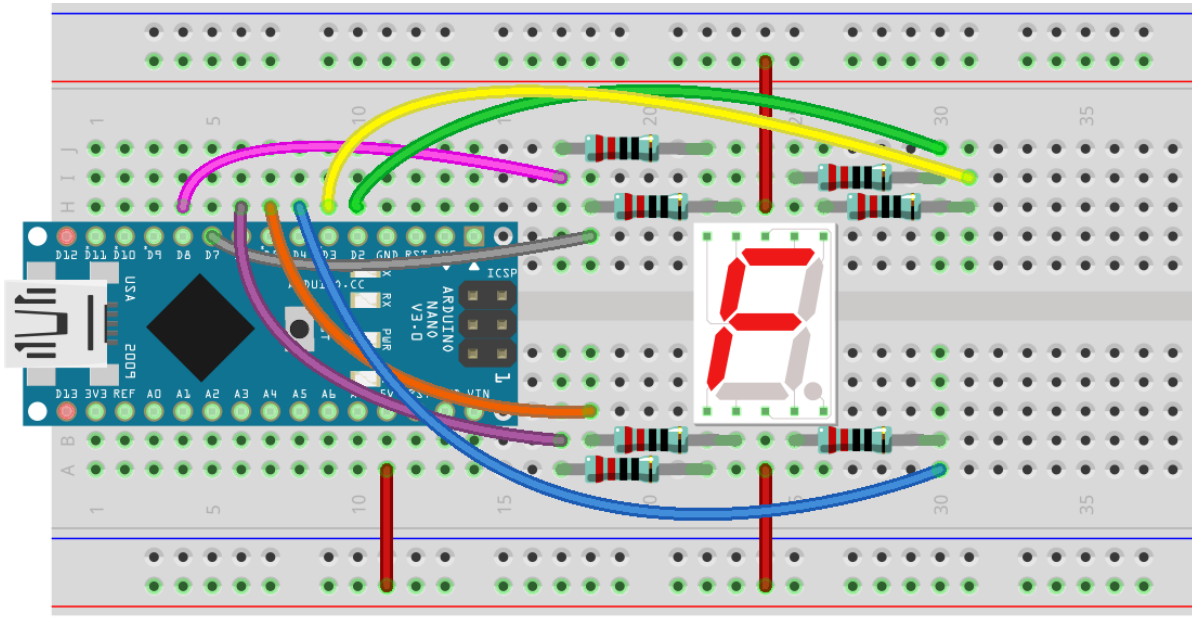
- A „lusta kapcsolás” itt abban áll, hogy csak a közös ágban használunk áramkorlátozó ellenállást
- Ennek a megoldásnak fő hátrányai:
  - Mivel változó számú szegmenst gyújtunk ki, nem oldható meg, hogy ne változzon a szegmensek fényereje
  - A párhuzamosan kapcsolt LED-ek nyitófeszültsége nem pontosan azonos, így az árammegosztás nem lesz egyenletes. Szélsőséges esetben egyes szegmensek nem is gyulladnak ki.

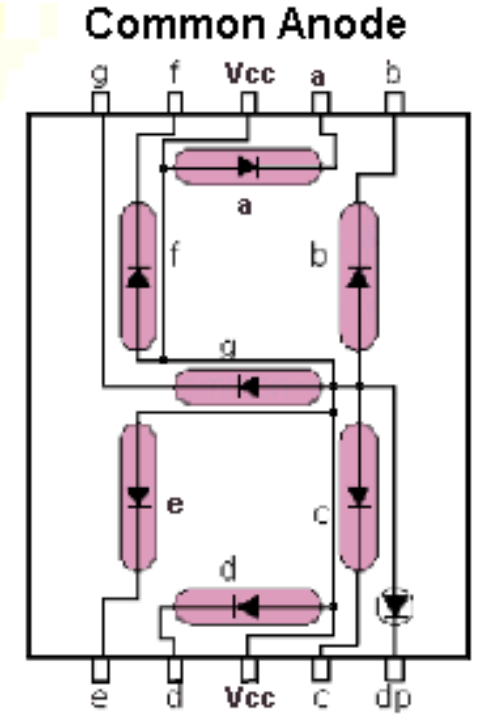


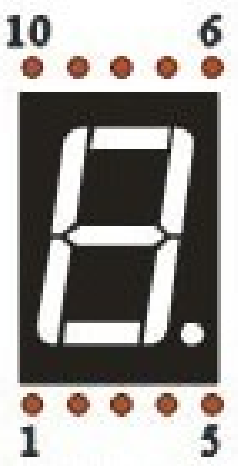
Hétszegmeneses LED szabályos bekötése

- A korrekt kapcsolásban minden szegmenshez külön-külön áramkorlátozó ellenállást használunk (a képen 220 Ω-osakat)
- Az a, b, c, d, e, f, g, DP szegmensek rendre a D2 … D8 kimenetekre csatlakoznak
- A kijelző szegmensek közös anódkivezetései az 5 V-os tápfeszültségre vannak kötve









LED_7seg_simple.ino

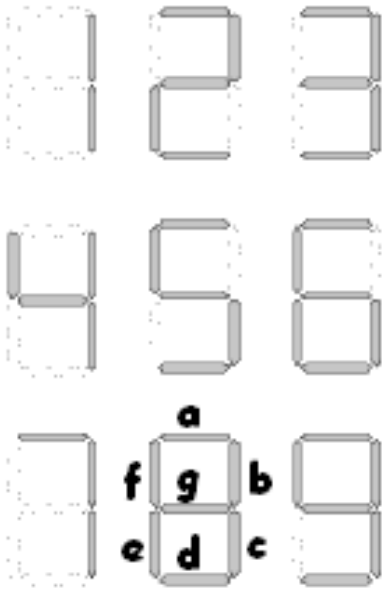
```
// Bekötések definiálása
#define segA  2
#define segB  3
#define segC  4
#define segD  5
#define segE  6
#define segF  7
#define segG  8

// Kimenetek beállítása
void setup() {
  pinMode(segA, OUTPUT);
  pinMode(segB, OUTPUT);
  pinMode(segC, OUTPUT);
  pinMode(segD, OUTPUT);
  pinMode(segE, OUTPUT);
  pinMode(segF, OUTPUT);
  pinMode(segG, OUTPUT);
}
```

```
void clr() { // Minden szegmens lekapcsolása
  digitalWrite(segA, HIGH);
  digitalWrite(segB, HIGH);
  digitalWrite(segC, HIGH);
  digitalWrite(segD, HIGH);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, HIGH);
}

void loop() { // Számjegyek kijelzése
  clr();   delay(1000);
  zero();  delay(1000);
  one();   delay(1000);
  two();   delay(1000);
  three(); delay(1000);
  four();  delay(1000);
  five();  delay(1000);
  six();   delay(1000);
  seven(); delay(1000);
  eight(); delay(1000);
  nine();  delay(1000);
}
```

Folytatás a következő oldalon

LED_7seg_simple.ino

```
void zero() { //Display 0
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, LOW);
  digitalWrite(segE, LOW);
  digitalWrite(segF, LOW);
  digitalWrite(segG, HIGH);
}
void one() {  //Display 1
  digitalWrite(segA, HIGH);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, HIGH);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, HIGH);
}
void two() { //Display 2
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, HIGH);
  digitalWrite(segD, LOW);
  digitalWrite(segE, LOW);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, LOW);
}
```



Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Forrás:www.thelearningpit.com/lp/doc/7seg/7seg.html

Itt most a LOW állapotba kapcsolás gyújtja ki a szegmenst!

```
void three() { //Display 3
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, LOW);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, LOW);
}
```

Folytatás a következő oldalon

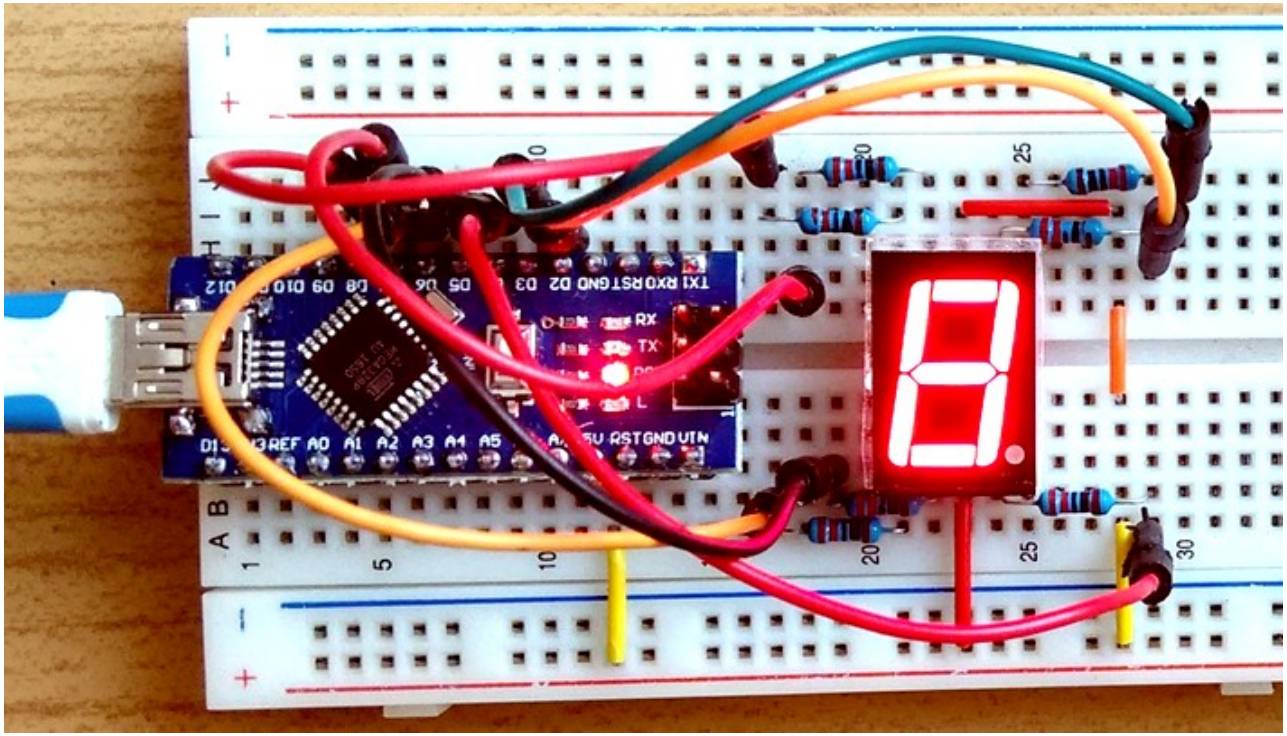
LED_7seg_simple.ino

```
void four() { //Displays 4
  digitalWrite(segA, HIGH);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, HIGH);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, LOW);
  digitalWrite(segG, LOW);
}
void five() { //Displays 5
  digitalWrite(segA, LOW);
  digitalWrite(segB, HIGH);
  digitalWrite(segC, LOW);
  digitalWrite(segD, LOW);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, LOW);
  digitalWrite(segG, LOW);
}
void six() { //Displays 6
  digitalWrite(segA, LOW);
  digitalWrite(segB, HIGH);
  digitalWrite(segC, LOW);
  digitalWrite(segD, LOW);
  digitalWrite(segE, LOW);
  digitalWrite(segF, LOW);
  digitalWrite(segG, LOW);
}
```

```
void seven() { //Display 7
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, HIGH);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, HIGH);
  digitalWrite(segG, HIGH);
}
void eight() { //Display 8
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, LOW);
  digitalWrite(segE, LOW);
  digitalWrite(segF, LOW);
  digitalWrite(segG, LOW);
}
void nine() { //Display 9
  digitalWrite(segA, LOW);
  digitalWrite(segB, LOW);
  digitalWrite(segC, LOW);
  digitalWrite(segD, LOW);
  digitalWrite(segE, HIGH);
  digitalWrite(segF, LOW);
  digitalWrite(segG, LOW);
}
```

Hogyan lehetne egyszerűsíteni?

- Bár a **LED_7seg_simple.ino** program működik, de terjedelmes és a favágás jellegű kivitele világosan mutatja, hogy messze áll az optimálistól
- Nem lehetne egyszerűbben és tömörebben átfogalmazni a programot? De igen, csak ehhez előbb meg kell ismerkednünk néhány új dologgal!
  - Tömbök használata
  - Bitenkénti műveletek
  - Bitműveletek



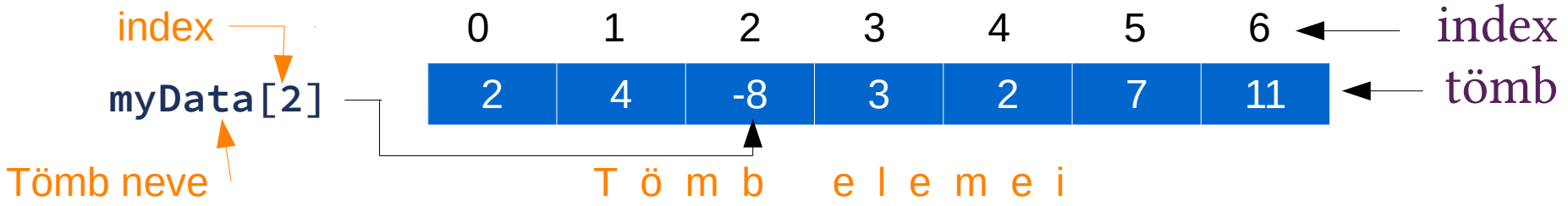
Tömbök használata

- A tömbök segítségével egy változóban több adatot fogunk össze (mint egy fiókos szekrényben). pl. `int x[5] = {8, 3, -2, 5, 0};`
- Az adatok eléréséhez ismerni kell a változó nevét, és az elővenni kívánt adat (a fiók) sorszámát. A sorszámozás 0-val kezdődik! Így `int a = x[3];` ugyanazt eredményezi, mint `int a = 5;`
- Példák tömb deklarálásra:

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int myData[7] = {2, 4, -8, 3, 2, 7, 11};
```

- Példák tömbelem értékadásra, illetve felhasználásra:

```
myInts[4] = 12;
mySenseVals[2] = 3 + myPins[2]
```



array_demo.ino

```
int a[10] = {2, 4, -8, 3, 2, 7, 11, 6, -2, 1};

void setup() {
  Serial.begin(9600);
  long sum = 0;
  float avg;
  Serial.println("\nA tömb elemei:");
  for (int i = 0; i < 10; i++) {
    sum = sum + a[i];
    Serial.print("a[");
    Serial.print(i);
    Serial.print("] = ");
    Serial.println(a[i]);
  }
  avg = sum / 10.0;
  Serial.print("Összeg = ");
  Serial.println(sum);
  Serial.print("Átlag = ");
  Serial.println(avg, 2);
}

void loop() {
}
```

Az adattömb elemek száma = 10

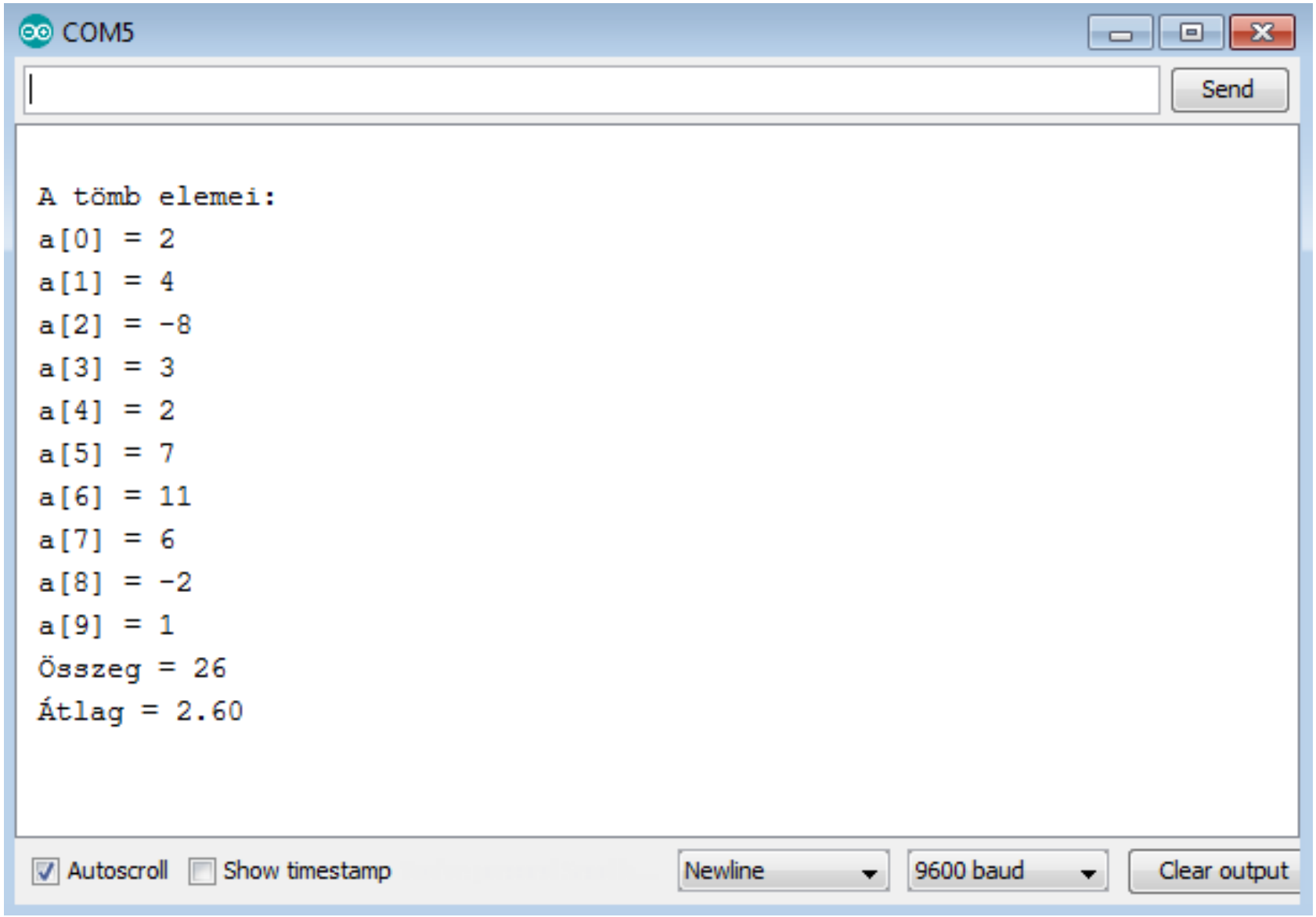
Összegzés

Tömbelem kiíratása

Átlagszámítás

Eredmények kiíratása

array_demo.ino futási eredménye

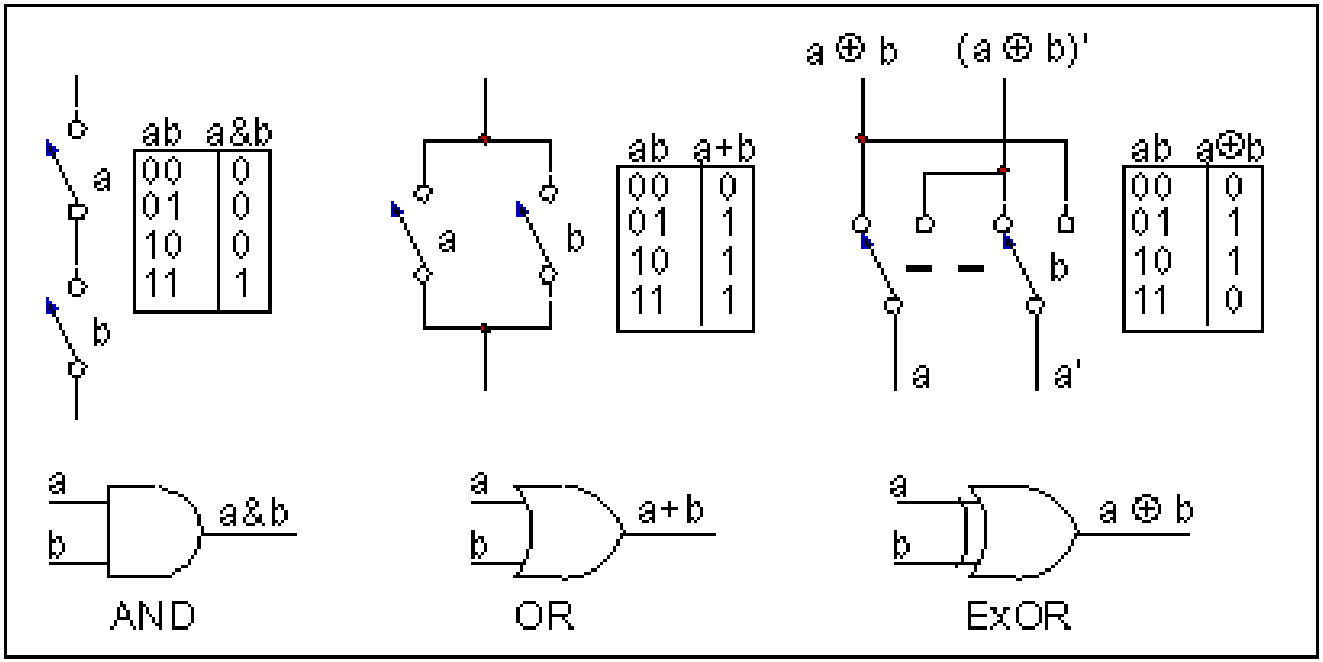


Bitenkénti műveletek a C nyelvben

- A C hardverközeli nyelv, ezért fontos szerepe van a bitenkénti műveletekenek, amikor az azonos helyiértékű bitek között végzünk bitközi logikai műveleteket.
- A jobbra/balra léptetéssel pedig gyorsan és hatékonyan oszthatunk, szorozhatunk 2 hatványaival.

Bitenkénti műveletek

& (bitenkénti ÉS)
| (bitenkénti VAGY)
^ (bitenkénti XOR)
~ (bitenkénti NEM)
<< (léptetés balra)
>> (léptetés jobbra)




```
Y = a & b;   Y = a | b;   Y = a ^ b;
```

Bitenkénti VAGY művelet

- `int A = 0b0010_1100;` (0x2C vagy 44_{10})
- `int B = 0b0100_0001;` (0x41 vagy 65_{10})
- Mi lesz `A = A | B;` eredménye?
- Elv: `x | 1 = 1,  x | 0 = x`
- Példa:

```
0 x 1 0   1 1 0 x   kiindulási érték
0 1 0 0   0 0 0 1   bitmaszk
-------------------------------
0 1 1 0   1 1 0 1   eredmény
```

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

- A bitenkénti **VAGY** műveletet arra is használhatjuk, hogy a maszkban 1-be állított bitek helyiértékén 1-be állítsuk egy változó vagy regiszter bitjeit.
  Például: `PORTD |= 0b00100000; // PORTD_5 legyen HIGH`

Bitenkénti ÉS művelet

- `int A = 0b0110_1101;` (0x6D vagy 109_{10})
- `int B = 0b1011_1110;` (~0x41 azaz 0xBE, 190_{10})
- Mi lesz `A = A & B;` eredménye?
- Elv: `x & 1 = x,  x & 0 = 0`
- Példa:

```
0 x 1 0   1 1 0 x   kiindulási érték
1 0 1 1   1 1 1 0   bitmaszk
-------------------------------
0 0 1 0   1 1 0 0   eredmény
```

A	B	A|B
0	0	0
0	1	0
1	0	0
1	1	1

- A bitenkénti **ÉS** műveletet arra is használhatjuk, hogy a maszkban 0-ra állított bitek helyiértékén 0-ra töröljük egy változó vagy regiszter bitjeit. Például: `PORTD &= 0b10111111;  // PORTD_6 legyen LOW!`

twoled_fastio.ino (csak haladóknak!)

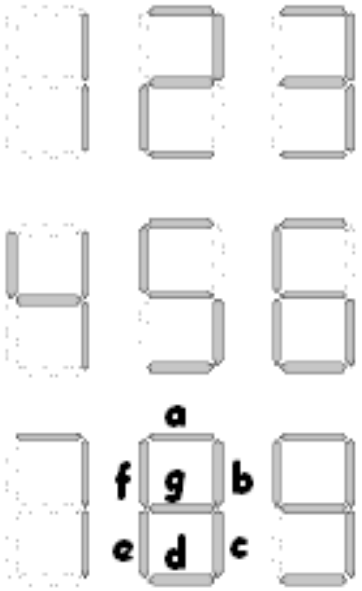
- Két LED-et villogtatunk ellenütemben, közvetlen portkezeléssel, ATmega168, vagy ATmega328 kártyán (Arduino UNO, nano, mini)
- Kapcsolás: a két LED-et (áramkorlátozó ellenálláson keresztül) a **D5** és a **D6** kivezetésekre kössük, amelyek **PORTD** 5. és 6. bitjével vannak összekötve
- Az adatáramlási irányt a **DDRD** regiszterben állíthatjuk be

```
void setup() {
  //Kezdeti beállítások
  DDRD |= 0b01100000;      //PORTD 5 és 6 bitje legyen kimenet!
  PORTD |= 0b00100000;     //Kezdetben PORTD_5 legyen HIGH!
  PORTD &= 0b10111111;     //Kezdetben PORTD_6 legyen LOW!
}

// a loop függvény újra és újra ismétlődik a végtelenségig
void loop() {
  PORTD ^= 0b01100000;     // a LED-ek állapotát átbillentjük (XOR művelettel)
  delay(1000);             // várunk egy másodpercig
}
```

LED_7seg_array.ino

Térjünk vissza a számkijelző vezérléséhez!
Definiáljunk egy tömböt a karakterképekkel!

```
#define segA  2
#define segB  3
#define segC  4
#define segD  5
#define segE  6
#define segF  7
#define segG  8
#define commonAnode true
//számjegyek rajzolata
//Sorrend: a b c d e f g DP
const byte digit[10] = {
  0b11111100,   // 0
  0b01100000,   // 1
  0b11011010,   // 2
  0b11110010,   // 3
  0b01100110,   // 4
  0b10110110,   // 5
  0b10111110,   // 6
  0b11100000,   // 7
  0b11111110,   // 8
  0b11110110    // 9
};
```



Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Például digit[4] a 4-es számjegy alakját adja meg (0b01 100 110)

Ha ezt a számot bitekre bontjuk, akkor egyenként beállíthatjuk a szegmenseket

LED_7seg_array.ino

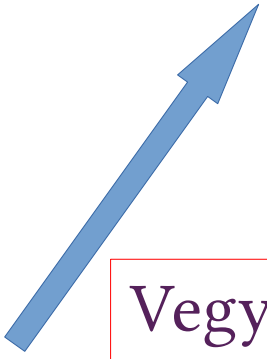
```
//--- Számjegy kiírása (n = 0..9) ------------
void displayNumber(byte n) {
  byte data = digit[n];                    // Számjegy alakja abcdefgDP sorrendben
  if ( commonAnode ) data = ~data;         // Negálni kell, ha közös anódú a kijelző
  digitalWrite(segA, bitRead(data, 7));    // A szegmens beállítása
  digitalWrite(segB, bitRead(data, 6));    // B szegmens beállítása
  digitalWrite(segC, bitRead(data, 5));    // C szegmens beállítása
  digitalWrite(segD, bitRead(data, 4));    // D szegmens beállítása
  digitalWrite(segE, bitRead(data, 3));    // E szegmens beállítása
  digitalWrite(segF, bitRead(data, 2));    // F szegmens beállítása
  digitalWrite(segG, bitRead(data, 1));    // G szegmens beállítása
}
void setup() {
  pinMode(segA, OUTPUT);
  pinMode(segB, OUTPUT);
  pinMode(segC, OUTPUT);
  pinMode(segD, OUTPUT);
  pinMode(segE, OUTPUT);
  pinMode(segF, OUTPUT);
  pinMode(segG, OUTPUT);
}
void loop() {
  for (byte i = 0; i < 10; i++) {          // Visszaszámlálás
    displayNumber(9 - i);                  // Kiíratjuk az aktuális számot
    delay(1000);                           // Várunk egy másodpercet
  }
}
```

- Tudunk még rövidíteni a programon? -
- Igen, lássunk hozzá!

Egyszerűsítsük tovább a program elejét!

- A tömbök segítségével tömörebben leírhatjuk a bekötések definiálását és a kiválasztott kivezetések kimenetre állítását

```
// Bekötések definiálása
#define segA  2
#define segB  3
#define segC  4
#define segD  5
#define segE  6
#define segF  7
#define segG  8


// Kimenetek beállítása
void setup() {
  pinMode(segA, OUTPUT);
  pinMode(segB, OUTPUT);
  pinMode(segC, OUTPUT);
  pinMode(segD, OUTPUT);
  pinMode(segE, OUTPUT);
  pinMode(segF, OUTPUT);
  pinMode(segG, OUTPUT);
}
```



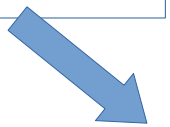
```
// Bekötések definiálása
byte myPins[7] = {2,3,4,5,6,7,8};

// Kimenetek beállítása
void setup() {
  for(int i=0; i<7; i++)
    pinMode(myPins[i], OUTPUT);
}
```

Vegyük észre, hogy nem használtuk ki a bekötéseknél megadott lábszámok folytonosságát! Más kiosztást is megadhatnánk, például így:

```
byte myPins[7] = {5,3,42,11,9,7,8};
```

Számjegykiírás rövidebben

```
// Számjegy kiírása (n = 0..9)
void displayNumber(byte n) {
  byte data = digit[n];
  if ( commonAnode ) data = ~data;
  digitalWrite(segA, bitRead(data, 7));
  digitalWrite(segB, bitRead(data, 6));
  digitalWrite(segC, bitRead(data, 5));
  digitalWrite(segD, bitRead(data, 4));
  digitalWrite(segE, bitRead(data, 3));
  digitalWrite(segF, bitRead(data, 2));
  digitalWrite(segG, bitRead(data, 1));
}
```

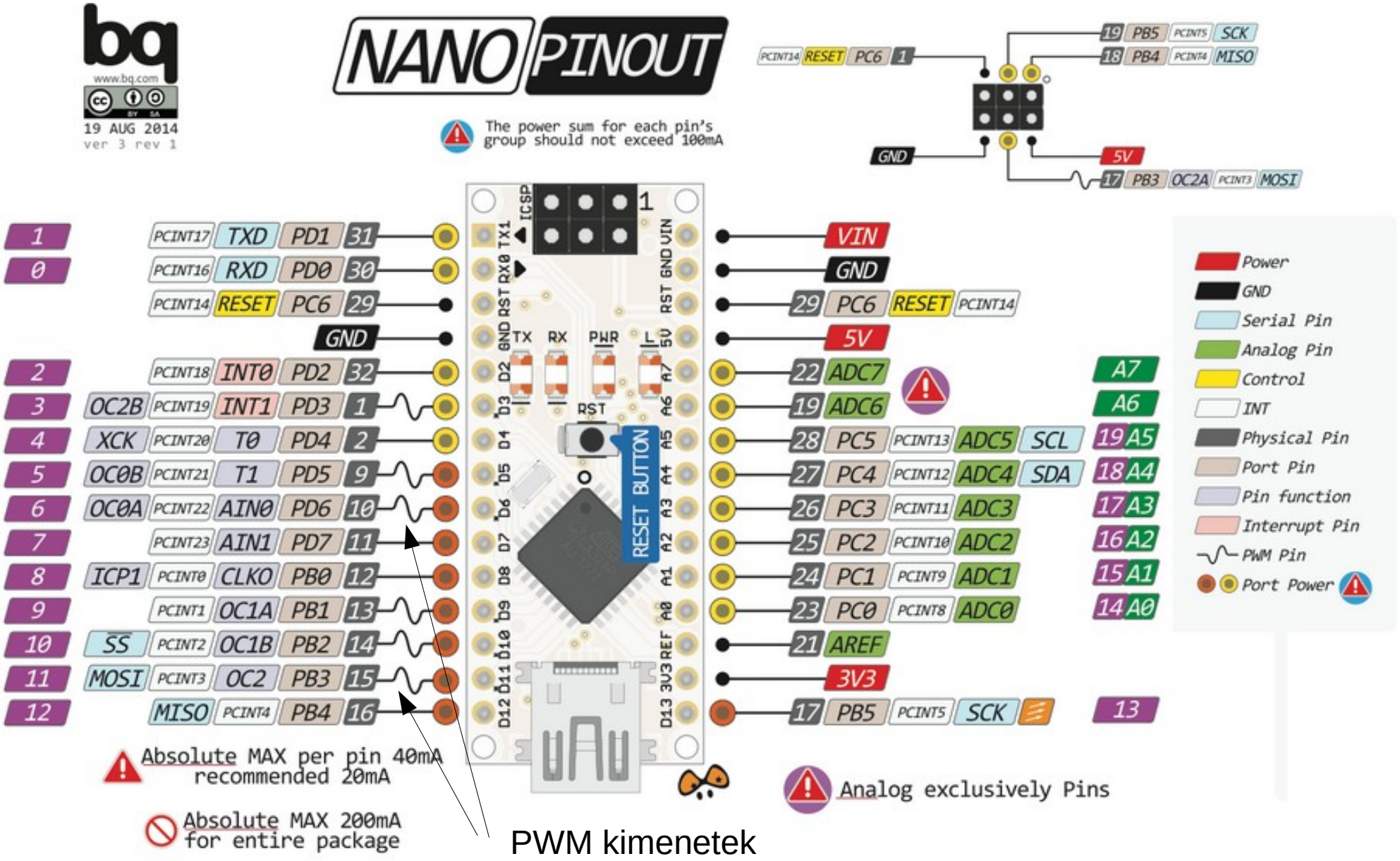


```
// Számjegy kiírása (n = 0..9)
void displayNumber(byte n) {
  byte data = digit[n];
  if( commonAnode ) data = ~data;
  for(i=0; i<7; i++) {
    digitalWrite(myPins[i],bitRead(data,7-i));
}
```

LED_7seg_short.ino

```
//számjegyek rajzolata, sorrend: a b c d e f g DP
const byte digit[10]={0xFC,0x60,0xDA,0xF2,0x66,0xB6,0xBE,0xE0,0xFE, 0xF6};
#define commonAnode true
byte myPins[7] = {2, 3, 4, 5, 6, 7, 8};  // Bekötések definiálása
void displayNumber(byte n) {             // Számjegy kiírása (n = 0..9)
  byte data = digit[n];
  if ( commonAnode ) data = ~data;
  for (int i = 0; i < 7; i++) {
    digitalWrite(myPins[i], bitRead(data, 7 - i));
  }
}

void setup() {                           // Kimenetek beállítása
  for (int i = 0; i < 7; i++)
    pinMode(myPins[i], OUTPUT);
}

void loop() {                            // Ciklikusan ismétlődő rész
  for (int i = 0; i < 10; i++) {         // Visszaszámlálás
    displayNumber(9 - i);                // Kiíratjuk az aktuális számot
    delay(1000);                         // Várunk egy másodpercet
  }
}
```

Az Arduino nano kártya kivezetései



PWM kimenetek