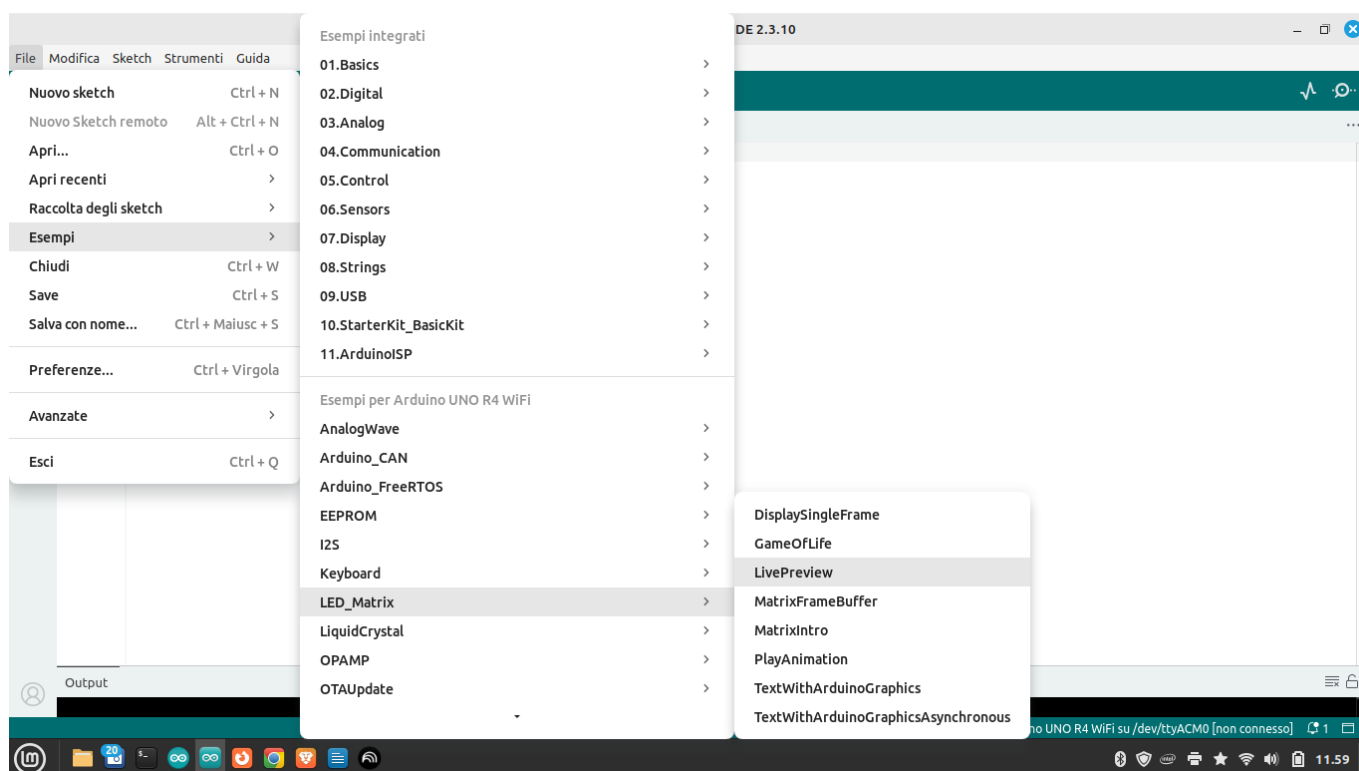


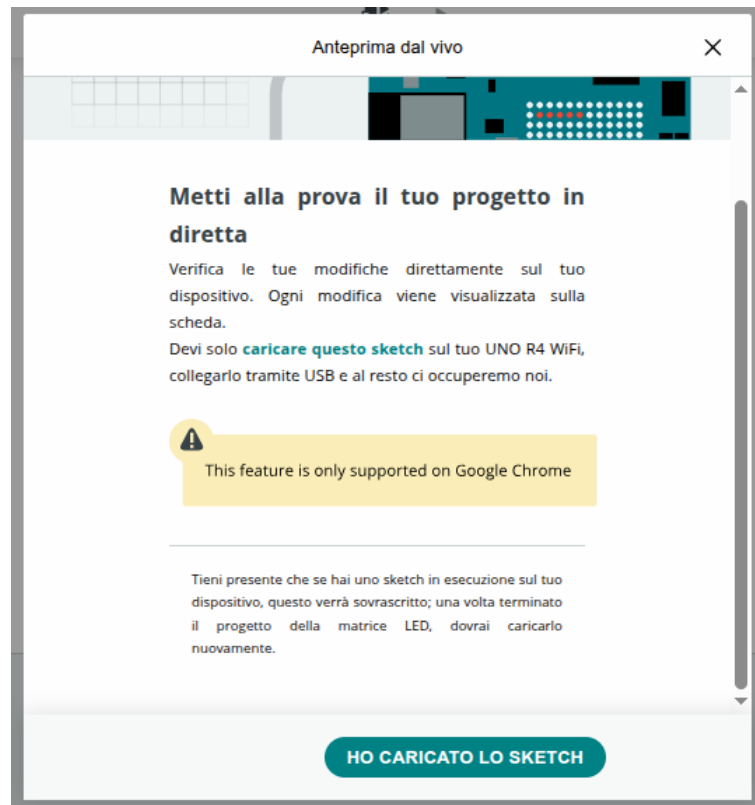
LivePreview

Per prendere dimestichezza con la matrice è utile vedere in diretta cosa si può disegnare sulla MatriceLED. Per far ciò è necessario fare interagire Arduino e **LED Matrix Editor** esclusivamente attraverso il browser Google Chrome. Si tratta di un **software sperimentale** in fase di pre-rilascio al momento della stesura del presente tutorial, per cui non è escluso qualche problemino!



Si procede nel seguente modo:

- Si collega Arduino al computer tramite porta USB
- Si apre l'IDE di Arduino
- Nel menu *File* / *Esempi* si va sulla sezione *Esempi per Arduino WiFi R4* ed in particolare su *LED_MATRIX*, e si carica il programma **LivePreview**
- Si apre **Google** Chrome all'indirizzo <https://ledmatrix-editor.arduino.cc/>
- Sulla schermata dell'Editor si clicca sull'icona del dischetto posta in alto al centro e si seguono le istruzioni:
 - si conferma di aver caricato lo sketch
 - si connette il proprio dispositivo collegandolo alla porta USB accoppiata
 - accertarsi che **non ci siano altre comunicazioni seriali attive**
 - si inizia a disegnare cliccando sugli elementi della matrice e si vedrà accendere o spegnere i LED sulla MatriceLED



Bisogna dar merito al piccolo **A. D.** che ha dato luogo ad una animazione... “a modo suo”:

- ha creato prima più fotogrammi di un pupazzetto in successione di varie posizioni;
- ha cliccato sull'icona posta in alto al centro dell'editor per creare la successione dei fotogrammi;
- poiché la permanenza di ogni fotogramma risultava ridottissima, ha duplicato più volte ogni singolo fotogramma per rallentare il movimento dell'animazione.

Sicuramente avrebbe agito diversamente se avesse atteso di essere informato che nella parte sottostante al singolo fotogramma compare il ritardo temporale prima di passare al fotogramma successivo; forse sarebbe stato meno creativo!

Durante le esercitazioni è stata riscontrata qualche difficoltà da parte di alcuni partecipanti; allorché si scollegava l'Arduino UNO R4 dal computer e lo si collegava nuovamente, veniva generata una diversa porta USB virtuale, ma nell'Editor l'Arduino risultava ancora collegato alla porta precedente: in alcuni casi è bastato chiudere e riavviare l'Editor...

◆Esercitazione

Creare nel **LED EDITOR MATRIX** un'animazione con i seguenti fotogrammi:



con un tempo di transizione di 600 ms.

Per creare nell'editor un'animazione continua, tenere premuti contemporaneamente il tasto *Maiuscolo* e la *Barra spaziatrice*

Tipi di dati

Un dato può essere variabile o costante:

- **variabile**
se il suo valore può cambiare nel corso del programma;
- **costante**
se il suo valore non può cambiare mai, può essere usato solo in modalità di lettura.

Strutturalmente può essere semplice o un array:

- **semplice**
costituito da un solo valore
- **array**
insieme di dati omogenei identificati da un solo nome

Dichiarazione di variabili e costanti

Sintassi per una **variabile**:

```
tipo_dato nome_variabile;
```

opzionalmente, inizializzando il valore della variabile,

```
tipo_dato nome_variabile = valore;
```

In Informatica il simbolo "=" ha un significato diverso da quello che noi diamo in Matematica, poiché indica **assegnazione** e precisamente: quello che compare a destra di "=" va messo nella zona di memoria denominata con il nome indicato alla sua sinistra, anche il risultato di una elaborazione.

Per le **costanti** si procede allo stesso modo facendo precedere il tutto da la parola chiave **const**:

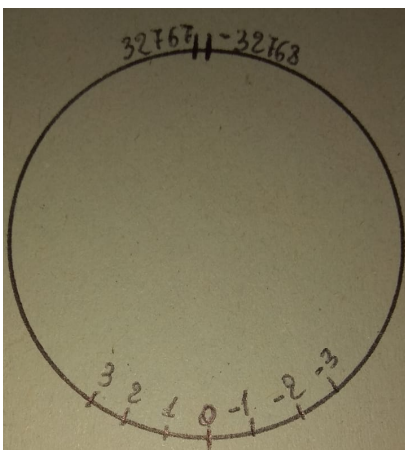
```
const tipo_dato nome_variabile = valore;
```

Ovviamente in questo caso l'assegnazione è obbligatoria.

Numeri interi con segno

Il tipo di dati interi ha diversi modi di memorizzazione.

int



Appartengono a questo tipo i numeri relativi compresi fra -32768 e 32767 cioè da -2^{15} a $(2^{15}-1)$.

Infatti questo tipo di dato impegna 2 byte, cioè 16 bit di memorizzazione per la sua rappresentazione binaria, dei quali uno viene utilizzato per il segno e gli altri quindici per il suo valore (il valore massimo è $2^{15}-1$ e non 2^{15} per via della rappresentazione dello zero).

Per dichiarare che x è una variabile di tipo **int** usiamo la sintassi:

```
int x;
```

se poi la si vuole anche inizializzare

```
int x= valore;
```

Bisogna stare attenti a non eccedere il campo di definizione, poiché rappresentazione di queste variabili ha un andamento "ciclico" e si manifesta un fenomeno conosciuto come "roll over" dovuto all'overflow della memoria; e *questo vale per tutte le variabili!* Per comprendere gli effetti di tale fenomeno, senza entrare nel meccanismo che lo causa, legato alla sua rappresentazione, aiutiamoci riportando su di una circonferenza i valori massimo e minimo possibile per il tipo di dato.

Se la variabile x contiene il valore - 32.768 ed assegniamo ad x il valore attuale diminuito di 1 ($x = x - 1$;) x dopo questa operazione conterrà 32.767 e non -32.768.

Analogamente se x contiene il valore 32.767, l'operazione $x = x + 1$; porterà x ad assumere il valore -32.768.

Nella rappresentazione di un numero intero con segno (signed) il primo bit è 0 se il numero è positivo, 1 se negativo. Per meglio comprendere l'errore cui si va incontro eccedendo l'area di memoria, supponiamo, per semplicità, di voler rappresentare un numero intero con 4 bit di cui il primo da sinistra è da riservare al segno. Ad esempio

- 0000 corrisponde in decimale a $+ 0 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 = 0$,
- 1000 corrisponde in decimale a $- 0 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 = 0$,
- 0111 corrisponde in decimale a $+ 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 = +7$, cioè il valore più grande rappresentabile con questo numero di bit.

binario					decimale	
0	1	1	1	+	7	+
0	0	0	1	=	1	=
1	0	0	0	+	0	+
			1	=	1	=
1	0	0	1		-1	

Per comprendere ancora meglio i danni provocati da un *overflow* basti pensare al display di un contagiri costituito da soli due *digit*; allorché si passa da 99 a 100, compaiono solo i due zeri poiché l'uno andrà fuori campo.

long

Memorizza un numero relativo utilizzando 4 byte (32 bit) per cui copre valori che vanno da -2^{31} a $2^{31}-1$, cioè da -2.147.483.648 a 2.147.483.647.

char

Memorizza un numero relativo utilizzando 1 byte (8 bit) coprendo valori che vanno da -128 a 127 cioè da -2^7 a 2^7-1 .

Numeri interi senza segno

- **unsigned int**
memorizza un intero senza segno in due byte. I valori vanno da 0 a $2^{16} - 1 = 65.535$.

- **unsigned long**
memorizza un numero senza segno utilizzando 4 byte (32 bit), per cui copre valori che vanno da 0 a $(2^{32} - 1) = 4.294.967.295$.
- **unsigned char**
memorizza un numero senza segno utilizzando 1 byte (8 bit), per cui copre valori che vanno da 0 a $(2^8 - 1) = 255$.
- **word**
sinonimo di **unsigned int**.
- **byte**
sinonimo di **unsigned char**.

Riassumendo:

int8_t char	un valore con segno a 8 bit da -2^7 a $+2^7-1$, cioè da -128 a +127	1 byte di RAM
uint8_t byte	un valore senza segno a 8 bit da 0 a 2^8-1 , cioè da 0 a 255	1 byte di RAM
int16_t int	un valore con segno a 16 bit da -2^{15} a $+2^{15}-1$, cioè, da -32.768 a +32.767	2 byte di RAM
uint16_t unsigned int	valore senza segno a 16 bit da 0 a 65.535	2 byte di RAM
int32_t long	valore con segno a 32 bit da -2.147.483.648 a +2.147.483.647	4 byte di RAM
uint32_t unsigned long	valore senza segno a 32 bit da 0 a 4.294.967.295	4 byte di RAM

Nella tabella viene riportata anche la **notazione a larghezza fissa** (o *fixed-width integer type*), in cui ogni parte del nome ha un significato preciso. A titolo di esempio analizziamo e scomponiamo la notazione

uint8_t

- **u** : sta per **unsigned** e significa che il tipo di dato non gestisce numeri negativi, ma solo valori da zero in su;
- **int** : sta per **Integer** ed indica che la variabile può contenere solo numeri interi;
- **8** : indica la **dimensione in bit** dello spazio occupato in memoria;
- **_t** : sta per **Type** (tipo). È un suffisso standard per indicare che non si è in presenza di un *tipo di dato* personalizzato.

Tale notazione è preferita ed ha preso piede perché **le dimensioni** dei tipi classici, come ad esempio int, long, **cambiano a seconda del computer o del compilatore**.

Tale notazione garantisce la **portabilità del codice** poiché un uint8_t occuperà **sempre** 8 bit indipendentemente dal fatto che si stia programmando su un piccolo microcontrollore Arduino o su un computer a 64-bit.