La gestione della grafica

Marcello Colozzo

Introduzione

Mathematica gestisce la grafica attraverso opportune combinazioni di *primitive grafiche*. Con tale denominazione ci si riferisce a enti geometrici elementari come ad esempio, un poligono, una circonferenza, un cubo, etc. Diversamente, oggetti del tipo *curva* (sia piana che sghemba) e *superficie*, vengono generati numericamente. Nel primo caso ci si riferisce, infatti, al diagramma cartesiano di una funzione reale di una variabile reale o ad un luogo geometrico in rappresentazione parametrica, mentre il secondo caso riguarda il diagramma cartesiano di una funzione reale di due variabili reali, considerando la possibilità di disegnare una superficie per via parametrica.

Primitive bidimensionali

La più semplice primitiva grafica è il punto. Nel caso bidimensionale, un punto viene definito dalla coppia delle sue coordinate cartesiane, per cui l'istruzione è **`Point[]`**, che accetta come argomento le predette coordinate rappresentate da una lista. Ad esempio:

```
Point[{2, 1}]
```

```
Point[{2, 1}]
```

Per stampare a video una qualunque primitiva, occorre processarla attraverso l'istruzione **`Graphics`**

```
Graphics[Point[{2, 1}]]
```


equivalente a

```
Point[{2, 1}] // Graphics
```


Per maggiore chiarezza, è preferibile indentare il codice (i commenti sono racchiusti tra (* *)):

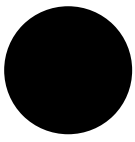
```
Graphics[
 (*primitiva grafica*)
 Point[{2, 0}]
]
```


L'istruzione **Graphics** ha l'attributo **Listable** nel senso che ignora eventuali parentesi graffe. Detto in altro modo, l'argomento di **Graphics** può essere una lista, cosicché

```
Graphics[
 {
  Point[{2, 0}]
 }
]
```

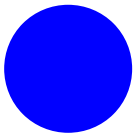

L'attributo **Listable** è vitale, in quanto spesso è necessario inserire alcune opzioni per le primitive grafiche. Stiamo parlando delle cosiddette *direttive grafiche.* Ad esempio, se vogliamo aumentare la dimensione del punto, utilizziamo il seguente codice:

```
Graphics[
 {
  PointSize[0.2],
  Point[{2, 0}]
 }
]
```



Se vogliamo un punto non di colore nero, ma blue:

```
Graphics[
 {
  Blue,
  PointSize[0.2],
  Point[{2, 0}]
 }
]
```



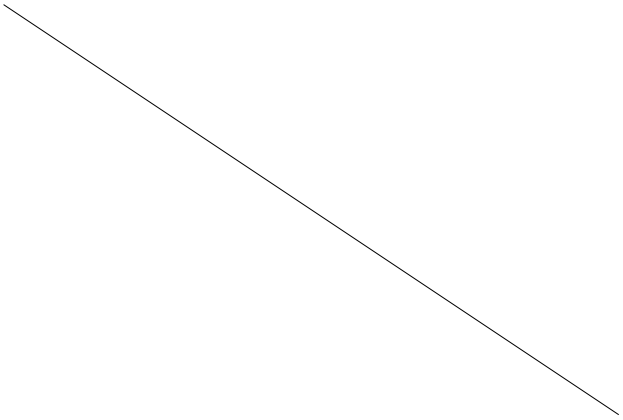
Dopo il punto, l'ente geometrico più semplice è il segmento, che è implementato dal comando **Line**. Tale funzione accetta come argomento una matrice quadrata di ordine 2, le cui righe sono le coordinate cartesiane dei vettori. Ad esempio, se vogliamo tracciare il segmento che unisce il punto $A(-1, 1)$ al punto $B\left(\frac{1}{2}, 0\right)$, scriviamo:

```
segmentoAB = Line[{{-1, 1}, {1/2, 0}}]
```

```
Line[{{-1, 1}, {1/2, 0}}]
```

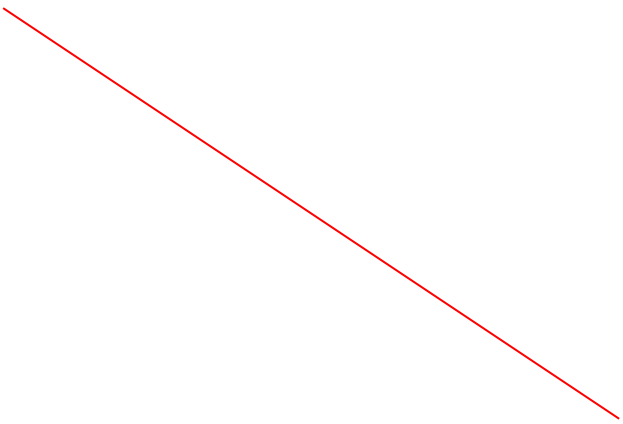
Al solito, per stampare a video dobbiamo processare attraverso **Graphics** la primitiva appena dichiarata:

```
Graphics[
 segmentoAB
]
```



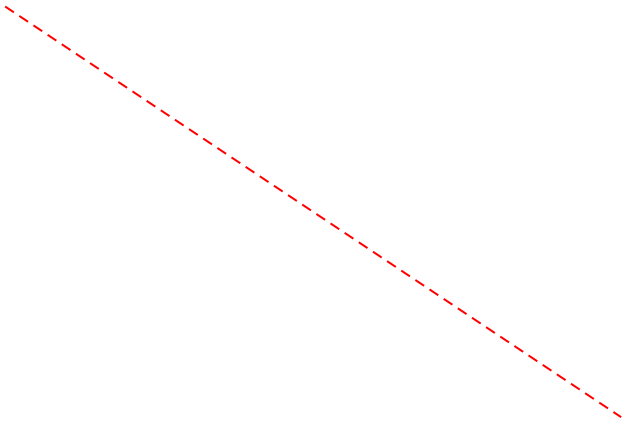
Nel caso del punto avevamo il controllo sulla dimensione e sul colore. Qui, invece, possiamo controllare lo spessore e il colore:

```
Graphics[
 {
  Thickness[0.003],
  Red,
  segmentoAB
 }
]
```



Se vogliamo tratteggiare il segmento utilizziamo **Dashed**

```
Graphics[
 {
  Thickness[0.003],
  Red,
  Dashed,
  segmentoAB
 }
]
```



Il tratteggio può essere modulato; in tal caso si utilizza al posto di **Dashed**, la direttiva **Dashing[]** come si può vedere dall'help in linea:

```
? Dashing
```

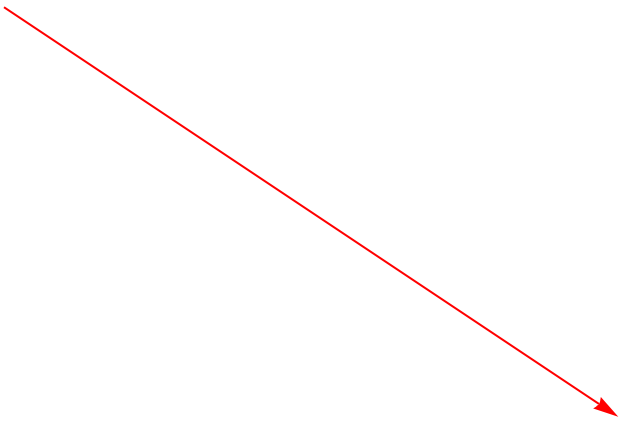
Dashing[{r_1, r_2, …}] is a two-dimensional graphics directive which specifies that lines which follow are to be drawn dashed, with successive segments of lengths r_1, r_2, … (repeated cyclically). The r_i are given as a fraction of the total width of the graph.
Dashing[r] is equivalent to Dashing[{r, r}]. ≫

Una primitiva simile a **Line** è **Arrow**:

```
frecciaAB = Arrow[{{-1, 1}, {1/2, 0}}]
```

```
Arrow[{{-1, 1}, {1/2, 0}}]
```

```
Graphics[
 {
  Thickness[0.003],
  Red,
  frecciaAB
 }
]
```



Anche tale primitiva presenta molte direttive grafiche:

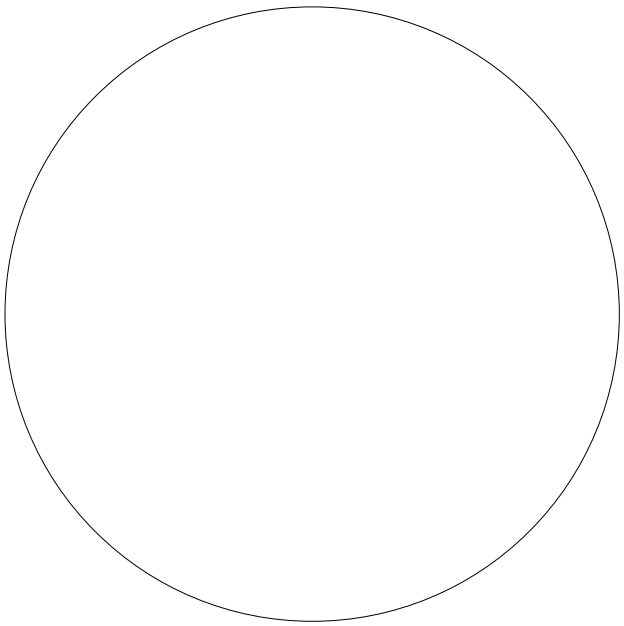
```
? Arrow
```

Arrow[{{x_1, y_1}, {x_2, y_2}}] is a graphics primitive which represents an arrow from (x_1, y_1) to (x_2, y_2).
Arrow[{pt_1, pt_2}, s] represents an arrow with its ends set back from pt_1 and pt_2 by a distance s.
Arrow[{pt_1, pt_2}, {s_1, s_2}] sets back by s_1 from pt_1 and s_2 from pt_2. ≫

Una circonferenza è rappresentata dalla primitiva **`Circle`** che accetta come argomento le coordinate cartesiane del centro e il raggio.

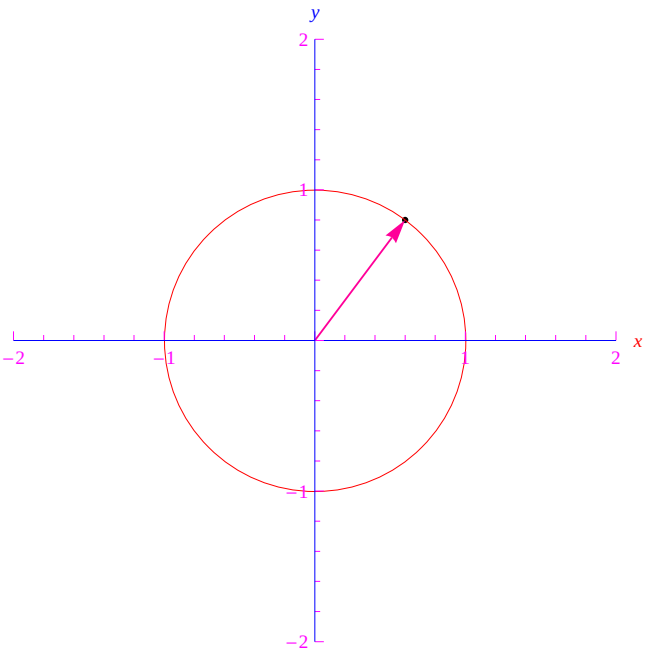
```
centro = {0, 0}; raggio = 1;
```

```
crf = Graphics[
  Circle[centro, raggio]
 ]
```



Aggiungiamo alcuni elementi, come ad esempio gli assi coordinati:

```
Graphics[
 {
  Point[{0.6, 0.8}],
  Red,
  Circle[centro, raggio],
  Hue[0.9],
  Thickness[0.003],
  Arrow[{{0, 0}, {0.6, 0.8}}]
 },
 PlotRange → {{-2, 2}, {-2, 2}},
 Axes → True,
 AxesStyle → Blue,
 AxesLabel → {
   Style["x", Small, Red, Italic],
   Style["y", Small, Blue, Italic]
  },
 TicksStyle → Magenta
]
```



È facile visualizzare a video il moto di un punto nel piano cartesiano, a patto di conoscere le funzioni $x(t)$ e $y(t)$ che definiscono la posizione del punto in funzione del tempo. Ad esempio, consideriamo un punto che si muove lungo la circonferenza centrata nell'origine e di raggio unitario, e con legge oraria $x(t) = \cos(\omega t)$, $y(t) = \sin(\omega t)$:

```
ω = π/5;
```

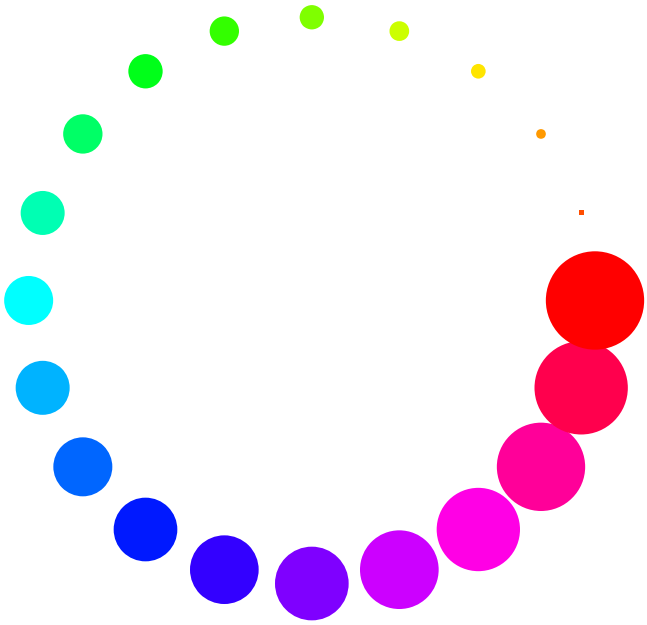
```
(punto[t_] := Point[{Cos[ω * t], Sin[ω * t]}];
 dimensione[t_] := PointSize[t / 60]; colore[t_] := Hue[t / 10])
```

Generiamo una lista di posizioni del punto per t variabile da 0 a 10 e con passo 0.5:

```
lista = Table[
   Graphics[
    {
     colore[t],
     dimensione[t],
     punto[t]
    }
   ],
   {t, 0, 10, 0.5}
  ];
```

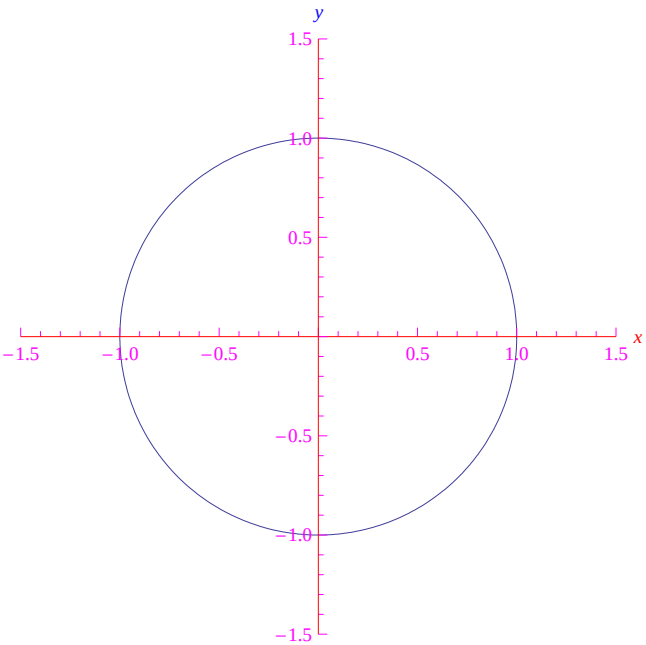
Per visualizzare le posizioni ottenute utilizziamo il comando **Show**

```
Show[lista]
```

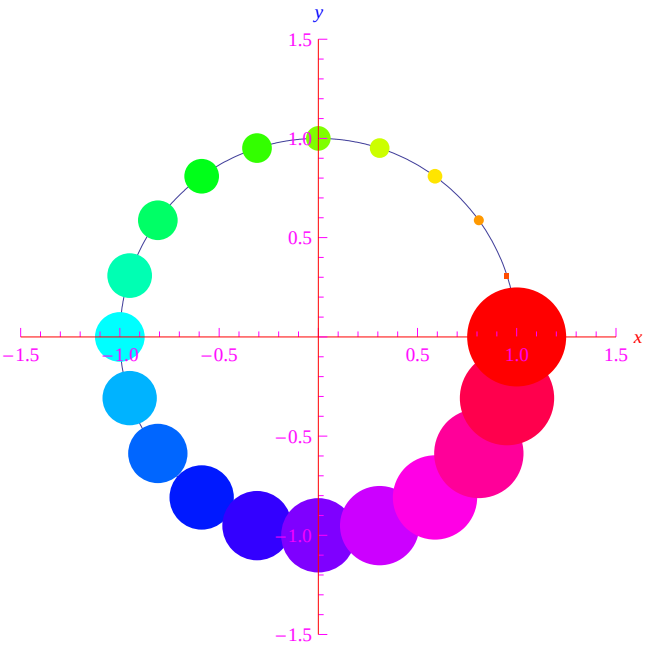


Proviamo a tracciare la traiettoria:

```
traiettoria = ParametricPlot[
  {Cos[2 t * π/10], Sin[2 t * π/10]},
  {t, 0, 10},
  PlotRange → {{-1.5, 1.5}, {-1.5, 1.5}},
  Axes → True,
  AxesStyle → Red,
  AxesLabel → {
    Style["x", Small, Red, Italic],
    Style["y", Small, Blue, Italic]
   },
  TicksStyle → Magenta
 ]
```



```
Show[
 {
  traiettoria,
  lista
 }
]
```

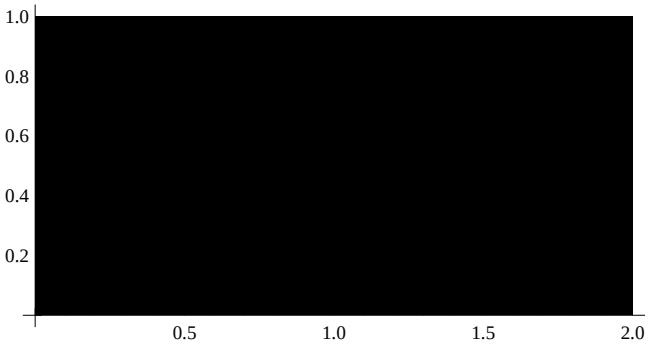


Un'altra primitiva grafica bidimensionale è il rettangolo, definito dalla funzione **Rectangle[]** che accetta come argomento le coordinate cartesiane dei vertici "estremi", cioè aventi rispettivamente le minime e massime coordinate.

```
Rectangle[{0, 0}, {2, 1}] // Graphics
```

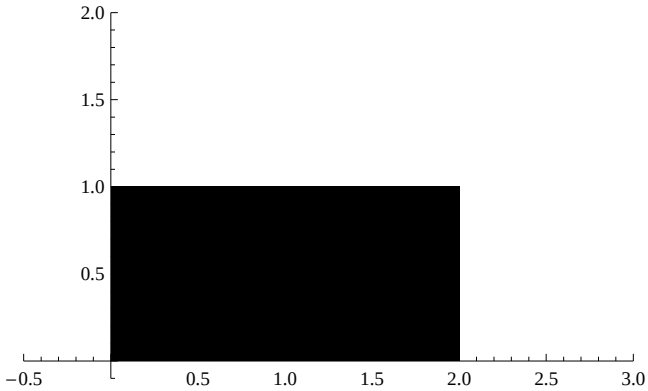
È preferibile assegnare il corrispondente sistema di assi cartesiani ortogonali:

```
Graphics[
 Rectangle[{0, 0}, {2, 1}],
 Axes → True
]
```



Per una migliore visualizzazione utilizziamo l'istruzione **PlotRange**

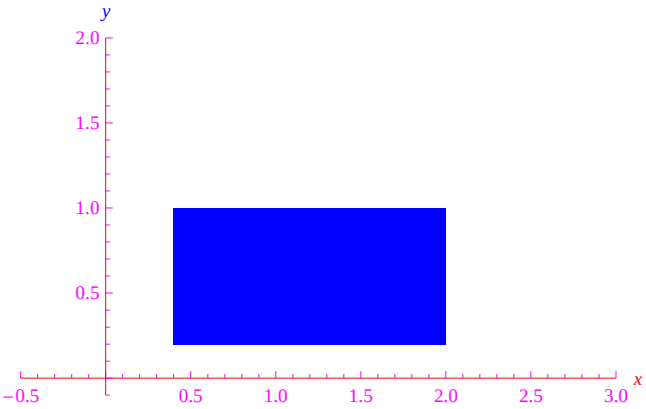
```
Graphics[
 Rectangle[{0, 0}, {2, 1}],
 Axes → True,
 PlotRange → {{-0.5, 3}, {-0.1, 2}}
]
```



Anche qui è possibile colorare l'immagine:

In[2]:=
```
Graphics[
 {
  Blue,
  Rectangle[{0.4, 0.2}, {2, 1}]
 },
 PlotRange → {{-0.5, 3}, {-0.1, 2}},
 Axes → True,
 AxesStyle → Red,
 AxesLabel → {
   Style["x", Small, Red, Italic],
   Style["y", Small, Blue, Italic]
  },
 TicksStyle → Magenta
]
```

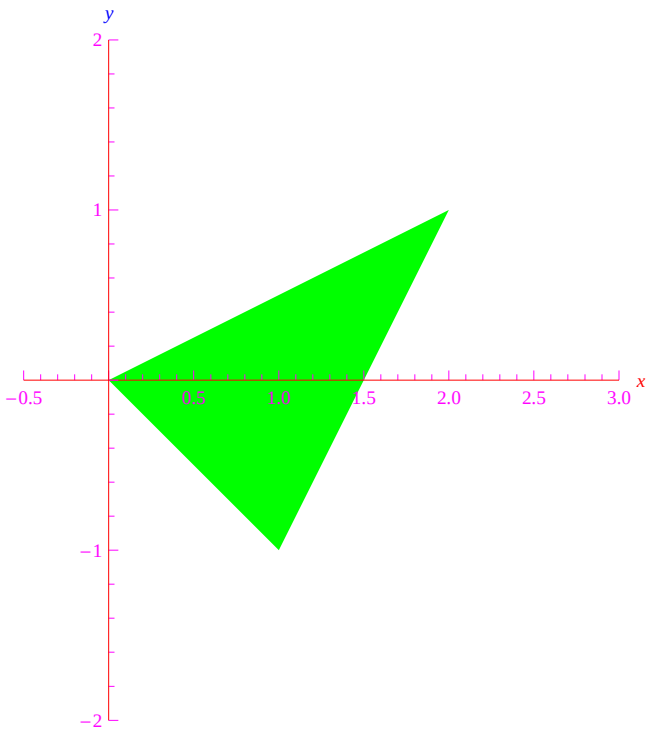
Out[2]=



La primitiva **Polygon[]** accetta come argomento le coordinate dei vertici del poligono che si vuole disegnare. Nel caso particolare di un triangolo:

In[3]:=
```
Graphics[
 {
  Green,
  Polygon[{{0, 0}, {2, 1}, {1, -1}}]
 },
 PlotRange → {{-0.5, 3}, {-2, 2}},
 Axes → True,
 AxesStyle → Red,
 AxesLabel → {
   Style["x", Small, Red, Italic],
   Style["y", Small, Blue, Italic]
  },
 TicksStyle → Magenta
]
```

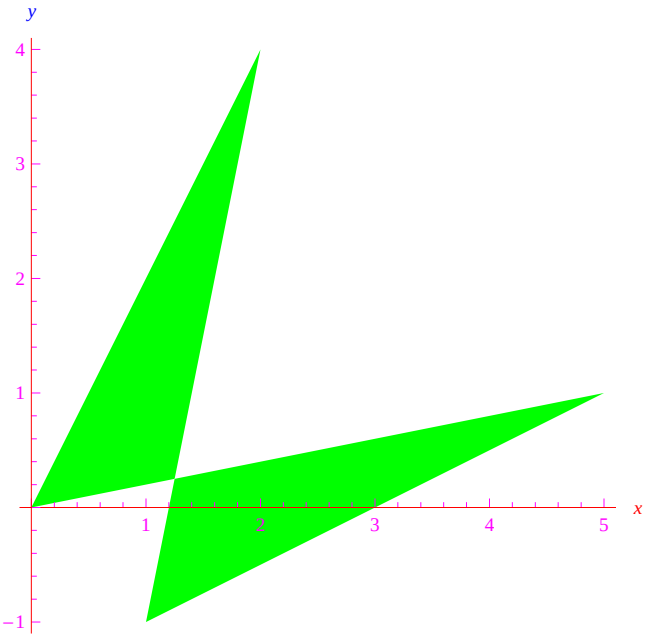
Out[3]=



Più in generale

In[4]:=
```
Graphics[
 {
  Green,
  Polygon[{{0, 0}, {5, 1}, {1, -1}, {2, 4}}]
 },
 PlotRange → All,
 Axes → True,
 AxesStyle → Red,
 AxesLabel → {
   Style["x", Small, Red, Italic],
   Style["y", Small, Blue, Italic]
  },
 TicksStyle → Magenta
]
```

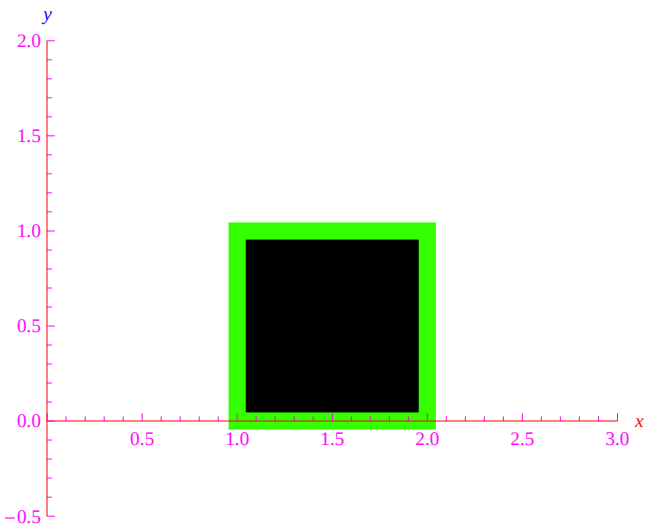
Out[4]=



Il bordo dei poligoni o di altri enti geometrici, può essere evidenziato attraverso la direttiva grafica **`EdgeForm[]`**. Ad esempio:

In[22]:=
```
Graphics[
 {
  EdgeForm[
   {
    (*colora il bordo*)
    Hue[0.3],
    (*definisce lo spessore*)
    Thickness[0.03]
   }
  ],
  Rectangle[{1, 0}, {2, 1}]
 },
 Axes → True,
 AxesStyle → Red,
 AxesLabel → {
   Style["x", Small, Red, Italic],
   Style["y", Small, Blue, Italic]
  },
 TicksStyle → Magenta,
 PlotRange → {{0, 3}, {-0.5, 2}}
]
```

Out[22]=



Utilizzando il generatore di liste **Table[]** è possibile creare una accattivante successione di poligoni. Definiamo le seguenti funzioni:

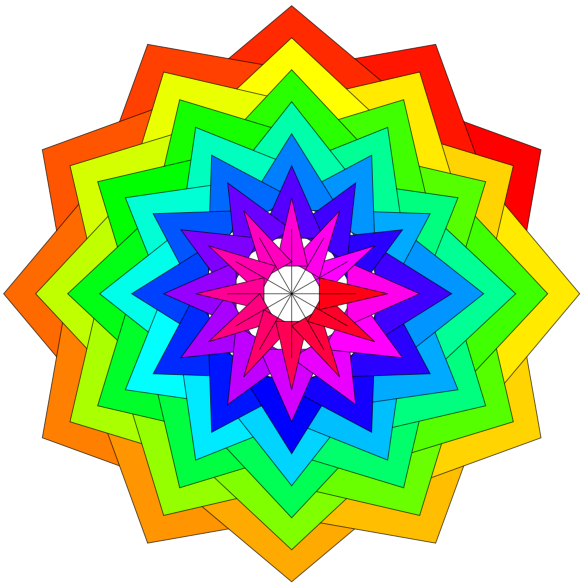
```
(v1[r_, d_, q_] := (8 - r) {Cos[d (q - 1)], Sin[d (q - 1)]};
  v2[r_, d_, q_] := (8 - r) {Cos[d (q + 1)], Sin[d (q + 1)]};
  v3[r_, d_, q_] := (10 - r) {Cos[d q], Sin[d q]});
```

Quindi quest'altra funzione:

In[64]:=
```
f[R_, Q_] := Graphics[
  Table[
   {EdgeForm[Opacity[.6]], Hue[(-11 + q + 10 r) / 72],
    Polygon[{v1[r, π / 6, q], v2[r, π / 6, q], v3[r, π / 6, q]}]},
   {r, R}, {q, Q}
  ],
  PlotRange → {{-10, 10}, {-10, 10}}
 ]
```

In[65]:=
```
f[8, 12]
```

Out[65]=



Per generare una animazione in formato gif, aumentiamo la dimensione dell'immagine:

In[66]:=
```
Clear[f]
```

In[67]:=
```
f[R_, Q_] := Graphics[
  Table[
   {EdgeForm[Opacity[.6]], Hue[(-11 + q + 10 r) / 72],
    Polygon[{v1[r, π / 6, q], v2[r, π / 6, q], v3[r, π / 6, q]}]},
   {r, R}, {q, Q}
  ],
  PlotRange → {{-10, 10}, {-10, 10}},
  ImageSize → {500, 500}
 ]
```

Generiamo i fotogrammi:

In[73]:=
```
data = Table[
   f[8, Q],
   {Q, 1, 13, 0.1}
  ];
```

Esportiamo il file:

In[75]:=
```
Export["animpoly.gif", data]
```

Out[75]=
```
animpoly.gif
```