



**Wolfram Mathematica**<sup>®</sup>

*Il software di riferimento per la Didattica, la Ricerca e lo Sviluppo*

---

WebSeminar  
Mathematica



## Lezione 6

# Interattività e dinamicità

*Crescenzo Gallo – Università di Foggia*

*crescenzo.gallo@unifg.it*

*Note:*

- Il materiale visualizzato durante questo seminario è disponibile per il download all'indirizzo <http://www.crescenziogallo.it/unifg/seminario-mathematica-2014/>
- Per una migliore visione ingrandire lo schermo mediante il pulsante in alto a destra "Schermo intero"

## Agenda

Come nasce il concetto di computazione dinamica e interattiva

I principali controller

Tutto si può rendere dinamico: esempi di base

Alcuni trucchi

Demonstrations (<http://demonstrations.wolfram.com>): esempi avanzati



## Come nasce il concetto di computazione dinamica e interattiva

*Mathematica* ha rivoluzionato il concetto di computazione interattiva e dinamica, introducendo funzioni dinamiche che istantaneamente creano interfacce intuitive e interattive. Le computazioni sottostanti vengono eseguite in run-time

**Integrate**[1 / (x<sup>3</sup> + 1), x]

$$\frac{\text{ArcTan}\left[\frac{-1+2x}{\sqrt{3}}\right]}{\sqrt{3}} + \frac{1}{3} \text{Log}[1+x] - \frac{1}{6} \text{Log}[1-x+x^2]$$



## Come nasce il concetto di computazione dinamica e interattiva

Alla base di **Manipulate** c'è la funzione **Dynamic**

```
Grid[{{"Statico", Solve[a x^2 + b x + c == 0, x]},
      {"Dinamico", Dynamic[Solve[a x^2 + b x + c == 0, x]]}}, Alignment -> Left, Dividers -> All]
```

|          |   |
|----------|---|
| Statico  | $\left\{ \left\{ x \rightarrow \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \left\{ x \rightarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right\} \right\}$ |
| Dinamico | <code>Solve[0.389376 a + 0.624 b + c == 0, 0.624]</code>  |



Assegniamo ora dei valori ai parametri a, b, c

```
{a = 0, b = 1, c = -2}
```

```
{0, 1, -2}
```

Si può creare un oggetto di tipo **SetterBar** per modificare più semplicemente i valori di a, b e c

```
SetterBar[Dynamic[a], {-1, 0, 1}]
```

|    |   |   |
|----|---|---|
| -1 | 0 | 1 |
|----|---|---|

```
Panel [  
  Column [  
    {Row[{"Imposta a ", SetterBar[Dynamic[a], Range[-5, 5]]}],  
      Row[{"Imposta b ", SetterBar[Dynamic[b], Range[-5, 5]]}],  
      Row[{"Imposta c ", SetterBar[Dynamic[c], Range[-5, 5]]}]}}]
```

|           |    |    |    |    |    |   |   |   |   |   |   |
|-----------|----|----|----|----|----|---|---|---|---|---|---|
| Imposta a | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 |
| Imposta b | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 |
| Imposta c | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 |



## I principali controller

This image displays a variety of interactive user interface (UI) components:

- Sliders:** A horizontal slider at the top left with a value of 0.316317 in an adjacent text box. A second slider is located at the bottom left, with a text box containing the number 41 below it.
- Form Elements:** A text box containing the name "Alice" with a dropdown arrow. A "click" button is shown with two radio buttons, one of which is selected. A "Browse..." button is accompanied by the text "(no file)". A "To Do" button is positioned at the bottom right.
- Checkboxes and Radio Buttons:** Three checkboxes are labeled "apples", "oranges", and "bananas", with "apples" and "bananas" checked. Three radio buttons are labeled "yes", "no", and "maybe", with "no" selected.
- Color and Navigation:** A color selection bar is present, along with a set of navigation icons (back, forward, home, search, refresh, etc.).
- Graphs and Lists:** A graph in the middle left shows a sine wave on a coordinate plane with x-axis values from 0 to 10 and y-axis values from -1.0 to 1.0. A list of five numbered buttons (1-5) is shown in the middle left, and another similar list is in the middle right.
- Other Elements:** A hand cursor icon is pointing at the second button of the middle-right list. A circular radial pattern is located at the bottom center.



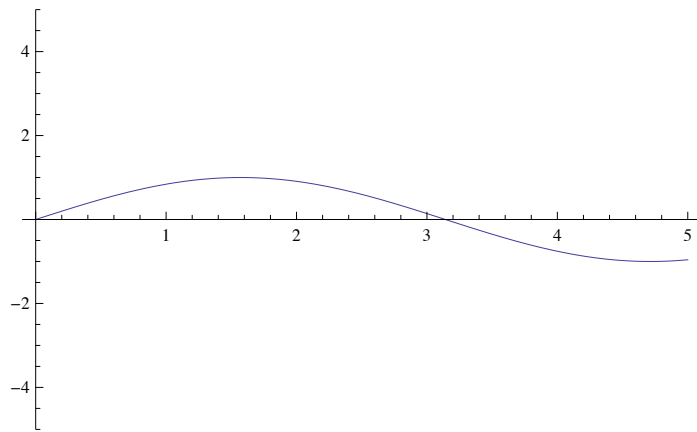


## Tutto si può rendere dinamico: esempi di base

Grazie sempre alla filosofia sintetizzata dallo slogan “Everything is an expression” anche gli oggetti dinamici possono utilizzare qualsiasi espressione nel loro modulo, dunque qualsiasi cosa in *Mathematica* si può rendere dinamico.

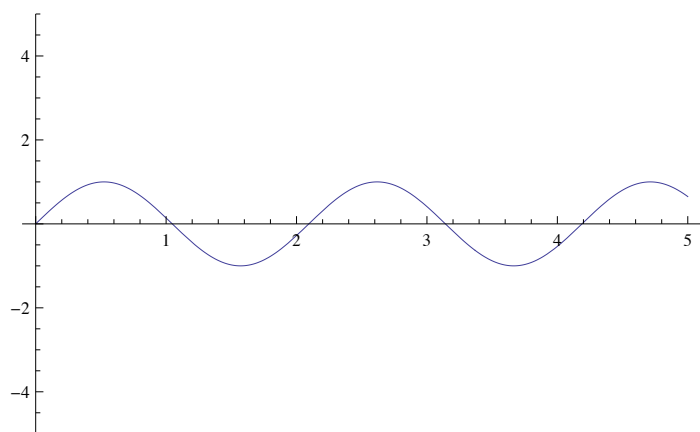
Esempio 1: semplificare la comprensione di un concetto

```
Plot[Sin[x], {x, 0, 5}, PlotRange → {-5, 5}]
```



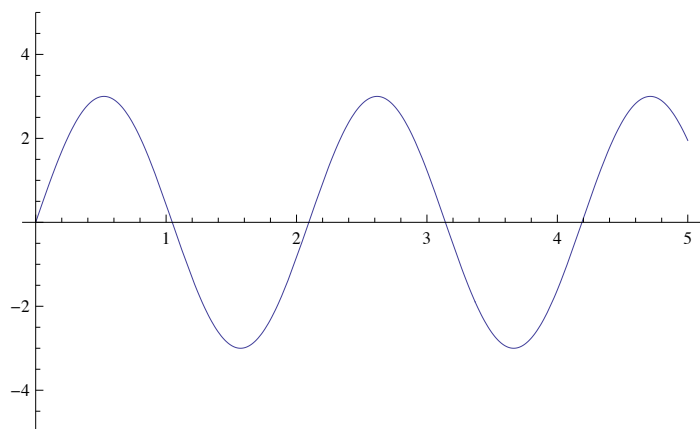
Voglio introdurre il concetto di frequenza

```
Plot[Sin[3 x], {x, 0, 5}, PlotRange → {-5, 5}]
```



e ampiezza

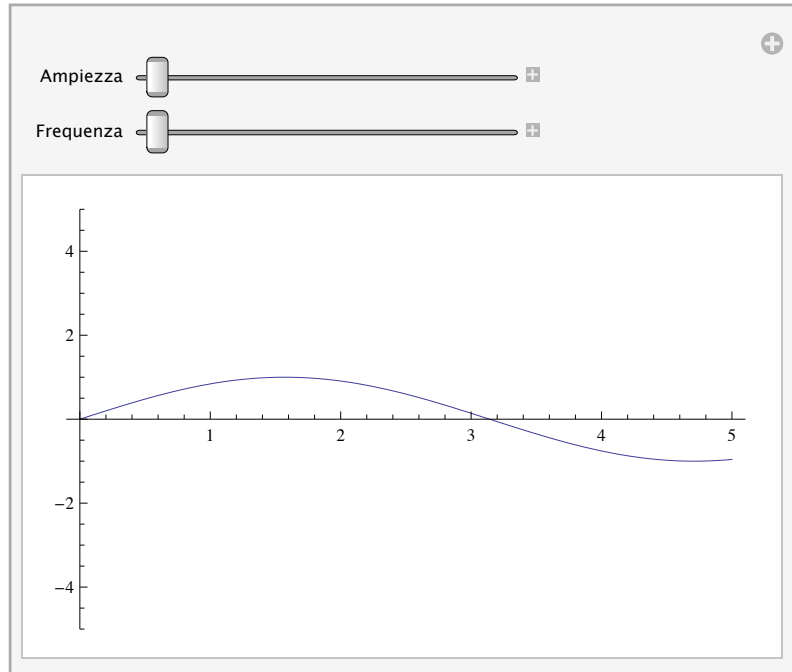
`Plot[3 Sin[3 x], {x, 0, 5}, PlotRange → {-5, 5}]`



Se voglio rendere più veloce il cambiamento al fine di poter concentrare poi la spiegazione sui due concetti, posso usare una **Manipulate** che mi permette di gestire in automatico i due parametri ampiezza e frequenza

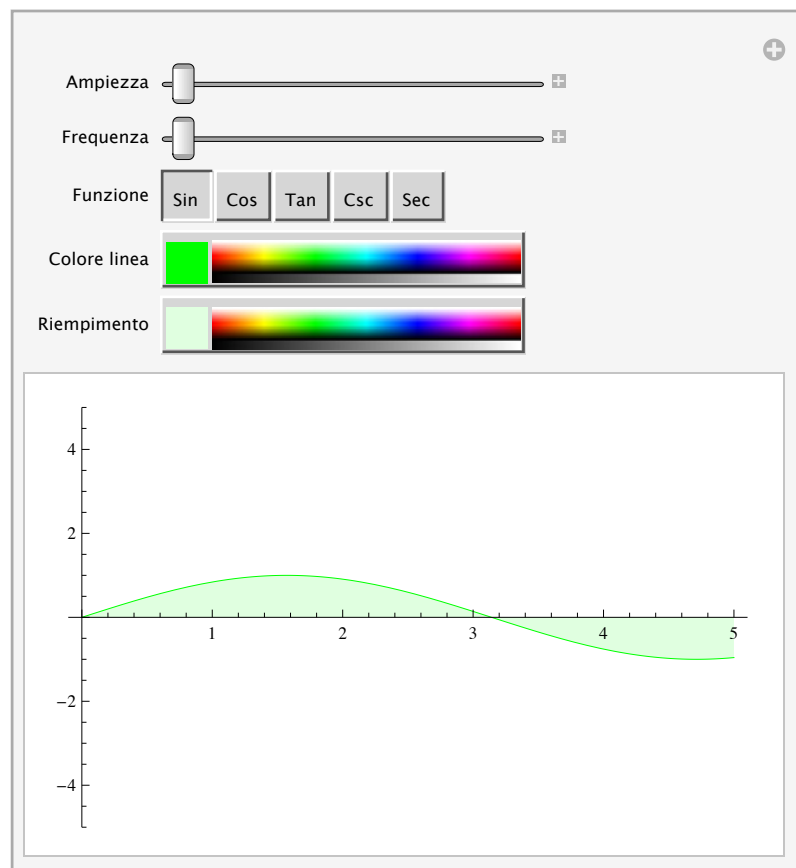
**Manipulate[**

```
Plot[amp Sin[freq x], {x, 0, 5}, PlotRange → {-5, 5}],  
{{amp, 1, "Ampiezza"}, 1, 5},  
{{freq, 1, "Frequenza"}, 1, 5}]
```



Qualche ulteriore abbellimento

```
Manipulate[Plot[amp funzione[freq x], {x, 0, 5},  
PlotRange → {-5, 5}, Filling → Axis, PlotStyle → pcol, FillingStyle → fcol],  
{{amp, 1, "Ampiezza"}, 1, 5},  
{{freq, 1, "Frequenza"}, 1, 5}, {{funzione, Sin, "Funzione"}, {Sin, Cos, Tan, Csc, Sec}},  
{{pcol, Green, "Colore linea"}, Red}, {{fcol, LightGreen, "Riempimento"}, LightRed}]
```



## Tutto si può rendere dinamico: esempi di base

Esempio 2: un semplice esercizio

```
Clear[a, b, c, x];
Manipulate[
  Panel[Grid[{{{"Equazione di partenza ",  $a x^2 + b x + c$ },
    {"Discriminante ( $b^2 - 4 a c$ )",  $b^2 - 4 a c$ },
    {"Soluzioni ",  $\text{Replace}[\text{Reduce}[a x^2 + b x + c == 0, x, \text{Reals}], \text{False} \rightarrow \text{"Nessuna"}]}$ }},
    Alignment  $\rightarrow$  Left]],
  {a, -5, 5, 1},
  {b, -5, 5, 1},
  {c, -5, 5, 1}]
```

The screenshot shows a Mathematica Manipulate interface. At the top, there are three sliders labeled 'a', 'b', and 'c', each with a range from -5 to 5 and a step of 1. Below the sliders, a panel displays the results of the quadratic equation:

|                                 |  |
|---------------------------------|--|
| Equazione di partenza           | -10.0669   |
| Discriminante ( $b^2 - 4 a c$ ) | -75  |
| Soluzioni                       | $\text{Reduce}[\text{False}, 0.624, \text{Reals}]$ |

Reduce::ivar : 0.624` is not a valid variable. >>

Volendo posso impostare un valore di partenza per ciascun parametro/slider

```
Clear[a, b, c, x];
```

```
Manipulate[
```

```
  Panel[Grid[{{{"Equazione di partenza ", a x^2 + b x + c },
    {"Discriminante (b^2-4 a c)", b^2 - 4 a c},
    {"Soluzioni ", Replace[Reduce[a x^2 + b x + c == 0, x, Reals], False -> "Nessuna"]}},
    Alignment -> Left]],
  {{a, -1}, -5, 5, 1},
  {{b, 1}, -5, 5, 1},
  {{c, -3}, -5, 5, 1}]
```

The screenshot shows a Mathematica Manipulate interface. At the top, there are three sliders for parameters a, b, and c. Below the sliders is a display box containing the following information:

|                                       |                             |
|---------------------------------------|-----------------------------|
| Equazione di partenza                 | -2.76538                    |
| Discriminante (b <sup>2</sup> -4 a c) | -11                         |
| Soluzioni                             | Reduce[False, 0.624, Reals] |

Reduce::ivar : 0.624` is not a valid variable. >>

## Tutto si può rendere dinamico: esempi di base

Esempio 3: un calcolo racchiuso in una tabella

**Manipulate**[

```
Grid[Prepend[Table[{i, m, im}, {i, 1, n}], {"n", "m", "nm"}]],
{n, 1, 20, 1}, {m, 1, 100, 1}]
```

The screenshot shows a Mathematica Manipulate interface. At the top, there are two sliders: one for 'n' and one for 'm'. The 'n' slider is currently set to 6, and the 'm' slider is set to 2. Below each slider is a small input field and a set of control buttons (minus, plus, up, down, and reset). Below the sliders is a table with three columns: 'n', 'm', and 'n<sup>m</sup>'. The table contains the following data:

| n | m | n <sup>m</sup> |
|---|---|----------------|
| 1 | 2 | 1              |
| 2 | 2 | 4              |
| 3 | 2 | 9              |
| 4 | 2 | 16             |
| 5 | 2 | 25             |
| 6 | 2 | 36             |

Con alcune opzioni aggiuntive possiamo migliorarne anche l'aspetto



```

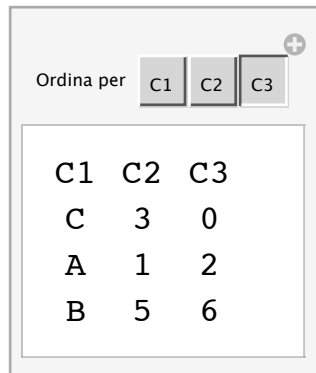
Manipulate[
  Grid[
    Prepend[Table[{i, m, im}, {i, 1, n}],
      {Style["n", Bold, Red], Style["m", Bold, Red], Style["nm", Bold, Red]}],
    Alignment → {{Left, Left, Right}, Automatic}, Frame → All,
    ItemStyle → {Blue, FontFamily → "Helvetica"}, Background → LightGray],
  {{n, 5, "Base"}, 1, 20, 1}, {{m, 50, "Esponente"}, 1, 100, 1}]

```

## Tutto si può rendere dinamico: esempi di base

Esempio 4: modificare l'ordine delle righe in una tabella

```
Manipulate[
  Grid[
    Prepend[
      SortBy[{{"A", 1, 2}, {"C", 3, 0}, {"B", 5, 6}}, (#[[sort]] &)], {"C1", "C2", "C3"}]],
  {{sort, 1, "Ordina per"}, {1 -> "C1", 2 -> "C2", 3 -> "C3"}}]
```



Ordina per  C1  C2  C3

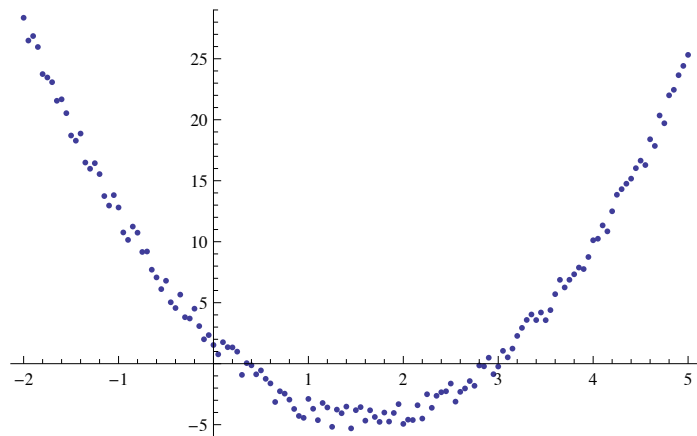
| C1 | C2 | C3 |
|----|----|----|
| C  | 3  | 0  |
| A  | 1  | 2  |
| B  | 5  | 6  |

## Tutto si può rendere dinamico: esempi di base

Esempio 4: un data fitting manuale

```
dati = Table[{x, 2.5 x^2 - 8 x + 1.87 + RandomReal[{-1, 1}]}, {x, -2, 5, 0.05}];
```

```
ListPlot[dati]
```



```
Manipulate[
```

```
  Show[
```

```
    ListPlot[dati],
```

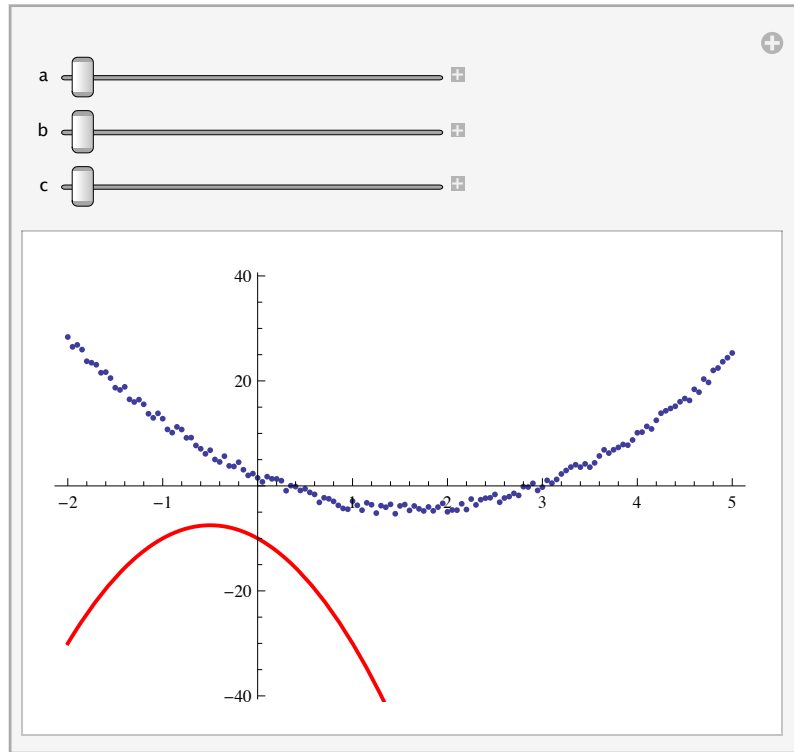
```
    Plot[a x^2 + b x + c, {x, -2, 5}, PlotStyle -> {Thick, Red}],
```

```
    PlotRange -> {{-2, 5}, {-40, 40}},
```

```
  {a, -10, 10},
```

```
  {b, -10, 10},
```

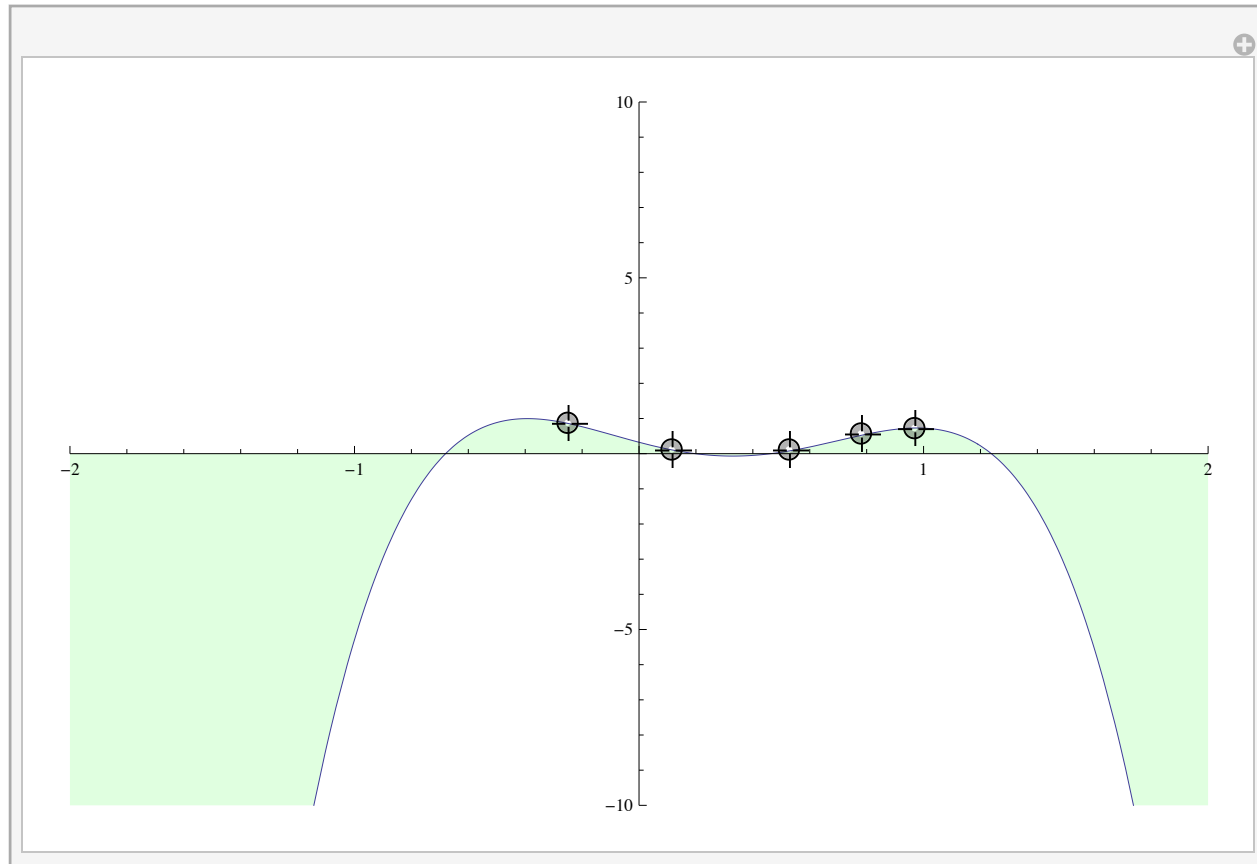
```
  {c, -10, 10}]
```



## Tutto si può rendere dinamico: esempi di base

Esempio 5: una interpolazione manuale

```
Manipulate[  
Plot[InterpolatingPolynomial[pts, x], {x, -2, 2}, PlotRange → {{-2, 2}, {-10, 10}},  
    Filling → Axis, FillingStyle → LightGreen, ImageSize → 600],  
{{pts, {{-0.25, 0.86}, {0.53, 0.08}, {0.97, 0.73}, {0.12, 0.10}, {0.78, 0.52}}},  
  {-2, -10}, {2, 10}, Locator, LocatorAutoCreate → True}]
```



## Tutto si può rendere dinamico: esempi di base

Esempio 6: una texture dinamica

```

clock := Module[{hour, min, sec, ht, mt, st},

  {hour, min, sec} = Take[DateList[], -3]; ht =  $\frac{\pi}{2} - \frac{\text{hour} \pi}{6} - \frac{\text{min} \pi}{360}$ ; mt =  $\frac{\pi}{2} - \frac{\text{min} \pi}{30}$ ;

  st =  $\frac{\pi}{2} - \frac{1}{30} \pi \text{Floor}[\text{sec}]$ ; Graphics[{AbsoluteThickness[5], Arrowheads[Large],

    Arrow[{{0, 0}, 0.6 {Cos[ht], Sin[ht]}]}, Arrow[{{0, 0}, 0.9 {Cos[mt], Sin[mt]}]},

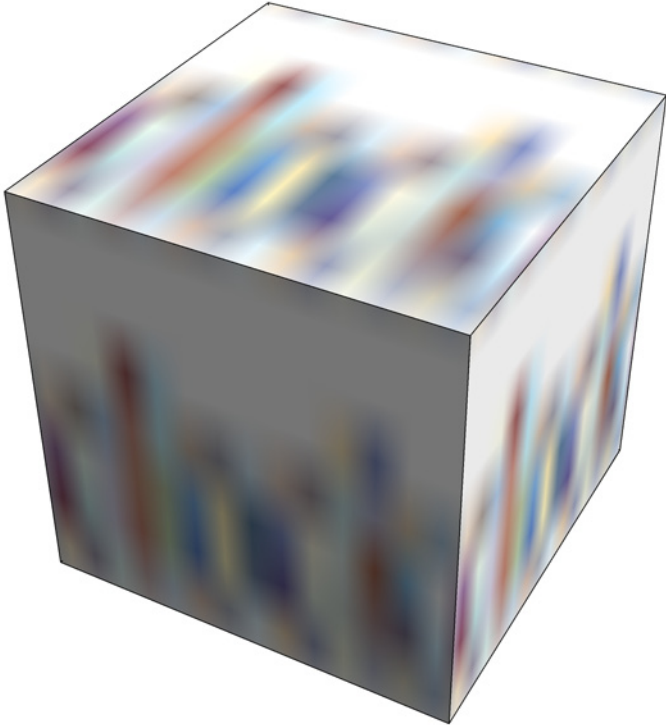
    PointSize[Large], Table[Point[0.9 {Cos[i], Sin[i]}], {i, 0, 2  $\pi$ ,  $\frac{\pi}{6}$ }],

    Point[{0, 0}], Circle[], Red, Line[{{0, 0}, 0.85 {Cos[st], Sin[st]}]}]}]]

Dynamic[Refresh[clock, UpdateInterval → 0]]
clock

vtc = {{0, 0}, {1, 0}, {1, 1}, {0, 1}};
coords =
  {{{0, 0, 0}, {0, 1, 0}, {1, 1, 0}, {1, 0, 0}}, {{0, 0, 0}, {1, 0, 0}, {1, 0, 1}, {0, 0, 1}},
   {{1, 0, 0}, {1, 1, 0}, {1, 1, 1}, {1, 0, 1}}, {{1, 1, 0}, {0, 1, 0}, {0, 1, 1}, {1, 1, 1}},
   {{0, 1, 0}, {0, 0, 0}, {0, 0, 1}, {0, 1, 1}}, {{0, 0, 1}, {1, 0, 1}, {1, 1, 1}, {0, 1, 1}}};
Dynamic[Graphics3D[{Dynamic[Texture[clock], UpdateInterval → 1], Polygon[coords,
  VertexTextureCoordinates → Table[vtc, {6}]]}, Lighting → "Neutral", Boxed → False]]

```

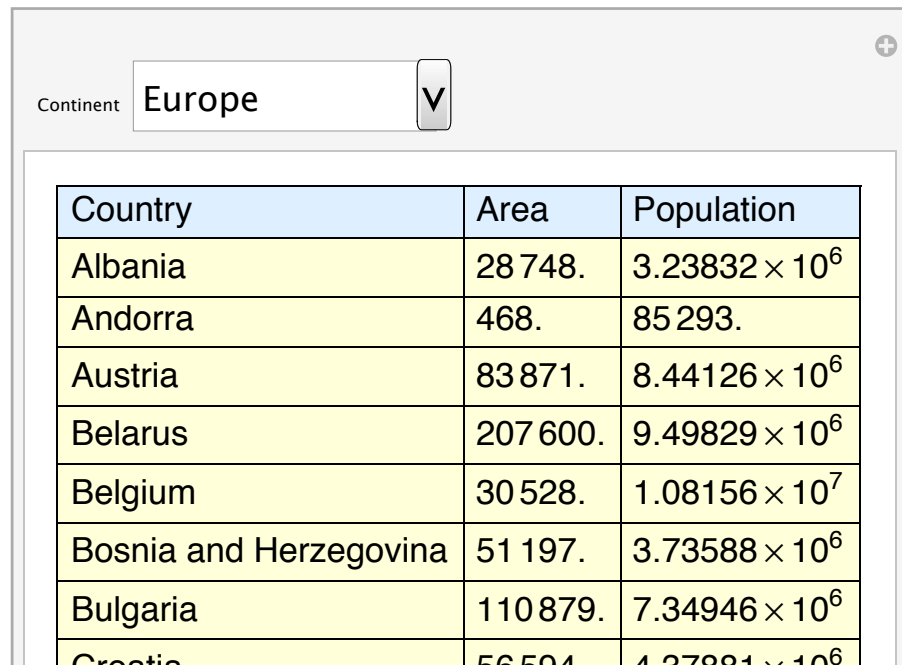




## Tutto si può rendere dinamico: esempi di base

Esempio 6: integrazione con le banche dati

```
Manipulate[
  Grid[Prepend[Outer[CountryData[#1, #2] &, CountryData[continent],
    {"Name", "Area", "Population"}], {"Country", "Area", "Population"}],
    Alignment → Left, BaseStyle → {FontFamily → "Helvetica"}, Frame → All,
    Background → {None, {LightBlue, {LightYellow}}}],
  {{continent, "SouthAmerica", "Continent"},
  DeleteCases[CountryData["Continents"], "Antarctica"]}]
```



| Country                | Area     | Population            |
|------------------------|----------|-----------------------|
| Albania                | 28 748.  | $3.23832 \times 10^6$ |
| Andorra                | 468.     | 85 293.               |
| Austria                | 83 871.  | $8.44126 \times 10^6$ |
| Belarus                | 207 600. | $9.49829 \times 10^6$ |
| Belgium                | 30 528.  | $1.08156 \times 10^7$ |
| Bosnia and Herzegovina | 51 197.  | $3.73588 \times 10^6$ |
| Bulgaria               | 110 879. | $7.34946 \times 10^6$ |
| Croatia                | 56 534.  | $4.27881 \times 10^6$ |

|                |          |                       |
|----------------|----------|-----------------------|
| Croatia        | 50 594.  | $4.57001 \times 10^5$ |
| Cyprus         | 9251.    | $1.14145 \times 10^6$ |
| Czech Republic | 78 867.  | $1.05898 \times 10^7$ |
| Denmark        | 43 094.  | $5.61119 \times 10^6$ |
| Estonia        | 45 228.  | $1.33883 \times 10^6$ |
| Faroe Islands  | 1393.    | 49 709.               |
| Finland        | 338 145. | $5.41876 \times 10^6$ |
| France         | 547 030. | $6.3783 \times 10^7$  |
| Germany        | 357 022. | $8.18042 \times 10^7$ |
| Gibraltar      | 6.5      | 29 111.               |
| Greece         | 131 940. | $1.14457 \times 10^7$ |
| Guernsey       | 78.      | 65 605.               |
| Hungary        | 93 028.  | $9.9338 \times 10^6$  |
| Iceland        | 103 000. | 331 996.              |
| Ireland        | 70 273.  | $4.63139 \times 10^6$ |
| Isle of Man    | 572.     | 86 159.               |
| Italy          | 301 340. | $6.10874 \times 10^7$ |
| Jersey         | 116.     | 95 732.               |
| Kosovo         | 10 887.  | $1.84771 \times 10^6$ |
| Latvia         | 64 589.  | $2.22626 \times 10^6$ |
| Liechtenstein  | 160.     | 37 009.               |
| Lithuania      | 65 300.  | $3.27834 \times 10^6$ |

|                |          |                       |
|----------------|----------|-----------------------|
| Luxembourg     | 2586.    | 530 018.              |
| Macedonia      | 25 713.  | $2.06922 \times 10^6$ |
| Malta          | 316.     | 420 557.              |
| Moldova        | 33 851.  | $3.49566 \times 10^6$ |
| Monaco         | 1.95     | 30 500.               |
| Montenegro     | 13 812.  | 633 167.              |
| Netherlands    | 41 543.  | $1.67616 \times 10^7$ |
| Norway         | 323 802. | $4.992 \times 10^6$   |
| Poland         | 312 685. | $3.83318 \times 10^7$ |
| Portugal       | 92 090.  | $1.07045 \times 10^7$ |
| Romania        | 238 391. | $2.13388 \times 10^7$ |
| San Marino     | 61.      | 32 448.               |
| Serbia         | 77 474.  | $9.83501 \times 10^6$ |
| Slovakia       | 49 035.  | $5.48878 \times 10^6$ |
| Slovenia       | 20 273.  | $2.04477 \times 10^6$ |
| Spain          | 504 782. | $4.70426 \times 10^7$ |
| Svalbard       | 62 045.  | 2495.                 |
| Sweden         | 450 295. | $9.54582 \times 10^6$ |
| Switzerland    | 41 277.  | $7.76182 \times 10^6$ |
| Ukraine        | 603 550. | $4.46967 \times 10^7$ |
| United Kingdom | 243 610. | $6.31774 \times 10^7$ |
| Vatican City   | 0.44     | 826                   |

|              |      |      |
|--------------|------|------|
| vatican city | 0.44 | 020. |
|--------------|------|------|



## Tutto si può rendere dinamico: esempi di base

Esempio 7: dinamicità ed interattività non significa solo Manipulate & Dynamic

Ci sono molte altre funzionalità che contribuiscono a rendere *Mathematica* un ambiente estremamente flessibile e versatile per la creazione di applicazioni user-friendly.

Vediamo alcuni esempi

» OpenerView

```
OpenerView[{Dynamic[DateString[], UpdateInterval -> 1], Dynamic[clock, UpdateInterval -> 1]}]
```

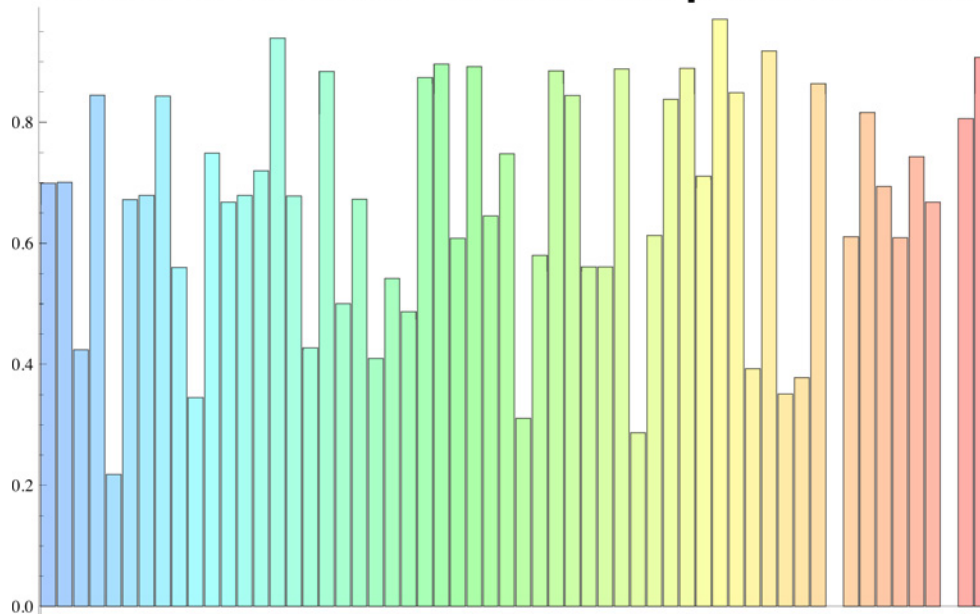
```
▼ Sun 2 Mar 2014 20:12:42
```

```
clock
```

» Tooltip

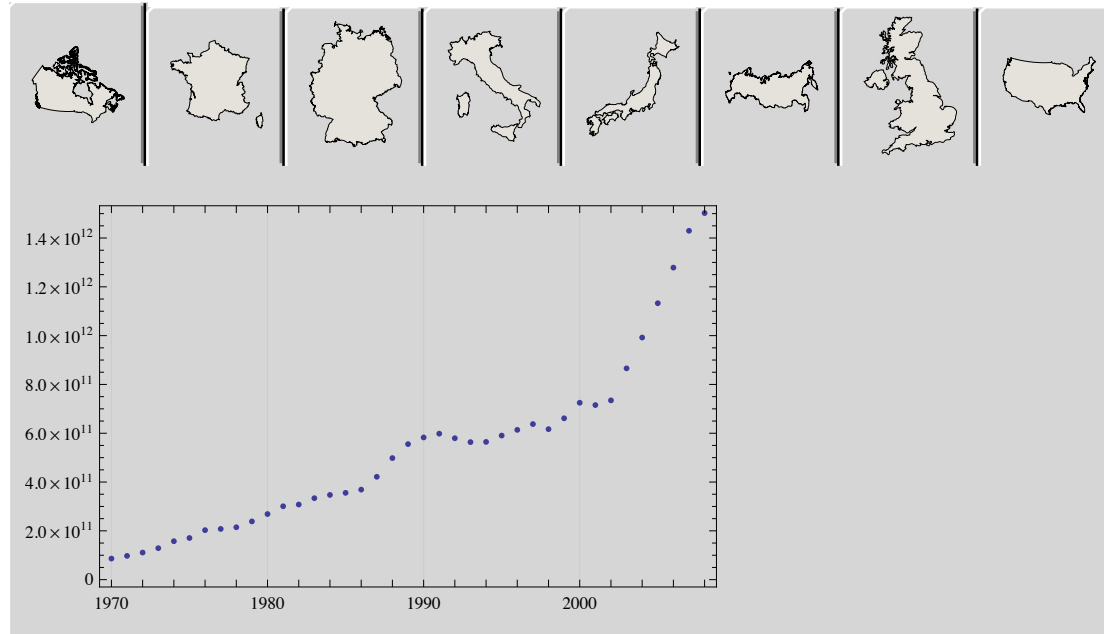
```
data = Tooltip[CountryData[#, "LiteracyFraction"], Framed@
  Column[{
    Style[CountryData[#, "Name"], Bold],
    CountryData[#, "Flag"]}]] & /@ CountryData["Africa"];
BarChart[{data}, PlotLabel →
  Style["Tasso di alfabetizzazione nei paesi Africani", FontFamily → "Arial", Bold, 24]]
```

### Tasso di alfabetizzazione nei paesi Africani



» Visualizzatori come TabView e FlipView

```
TabView[Map[Tooltip[Show[CountryData[#, "Shape"], ImageSize -> 50], #] ->
  DateListPlot[CountryData[#, {"GDP"}, {1970, 2010}]]] &, CountryData["GroupOf8"]]
```



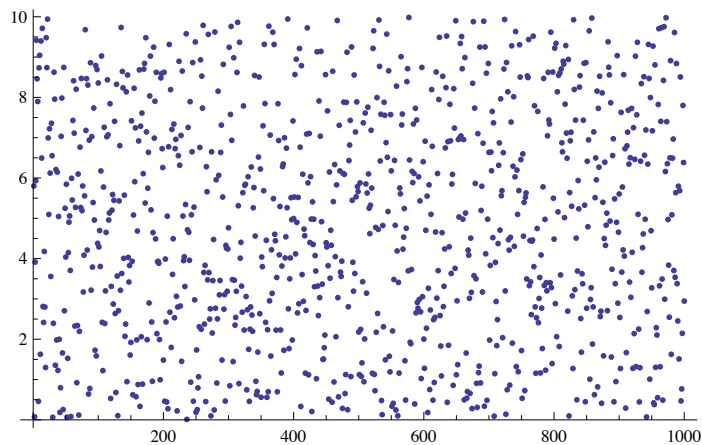
```
FlipView[Row[#, "\t", Rasterize@CountryData[#, "Shape"]]] & /@ CountryData["Europe"]]
```

BosniaHerzegovina



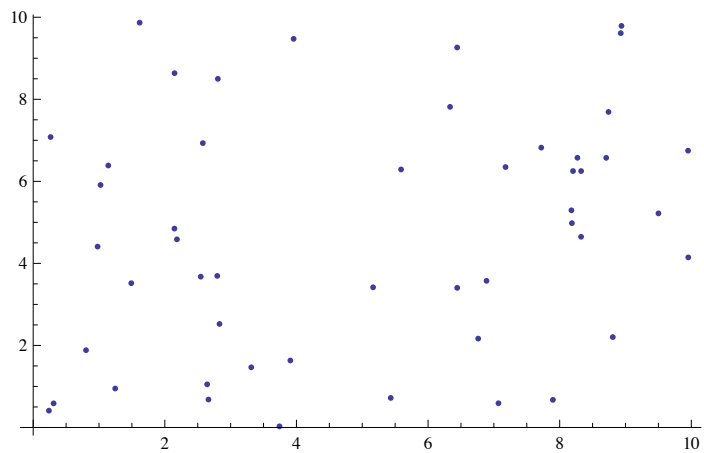
» Mouseover

```
data = RandomReal[{0, 10}, 1000];  
Mouseover[ListPlot[data], BarChart[BinCounts[data, {0, 10, 1}]]]
```



```
data = RandomReal[10, {50, 2}];  
Mouseover[ListPlot[data],  
  Tooltip[ListLinePlot[data[[Last@FindShortestTour[data]]]], "Percorso minimo"]]
```





```
OpenerView[{Dynamic[DateString[], UpdateInterval -> 1], Dynamic[clock, UpdateInterval -> 1]}]
```

```
▼ Sun 2 Mar 2014 20:12:42
```

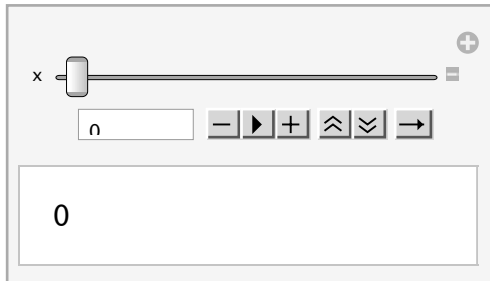
```
clock
```

## Alcuni trucchi

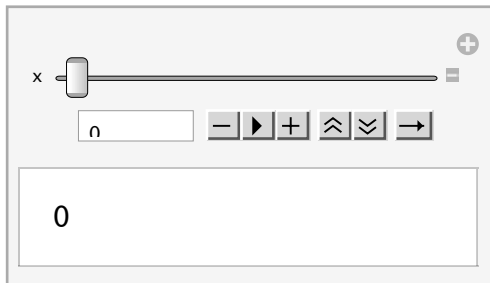
### » Personalizzazione della posizione dei controllers

In alcuni casi il modo in cui **Manipulate** dispone i controller potrebbe non soddisfare le esigenze dell'utente. Per ovviare a questo problema si può usare il costrutto `Control` che ci permette di personalizzare in maniera più flessibile i controller che governano le variabili. Ovviamente bisogna scrivere del codice in più e comporre manualmente ogni singolo pezzo dell'applicazione.

**Manipulate**[*x*, {*x*, 0, 1}]

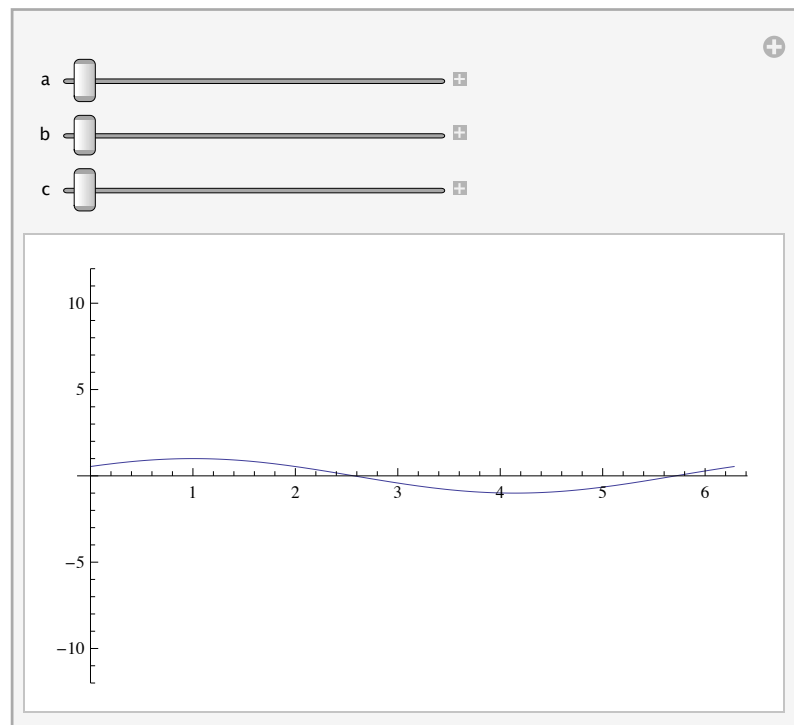


**Manipulate**[*x*, `Control`[{*x*, 0, 1}]]



**Manipulate** ha delle opzioni che ci permettono di dire che tipo di controller utilizzare e dove collocarli rispetto al riquadro principale (Left, Right, Top, Bottom) ma non ci permette di modificarne altre disposizioni. Facciamo alcuni esempi.

```
Manipulate[
  Plot[a Cos[b (t - c)], {t, 0, 2 π}, PlotRange -> 12],
  {a, 1, 12},
  {b, 1, 12},
  {c, 1, 12}]
```

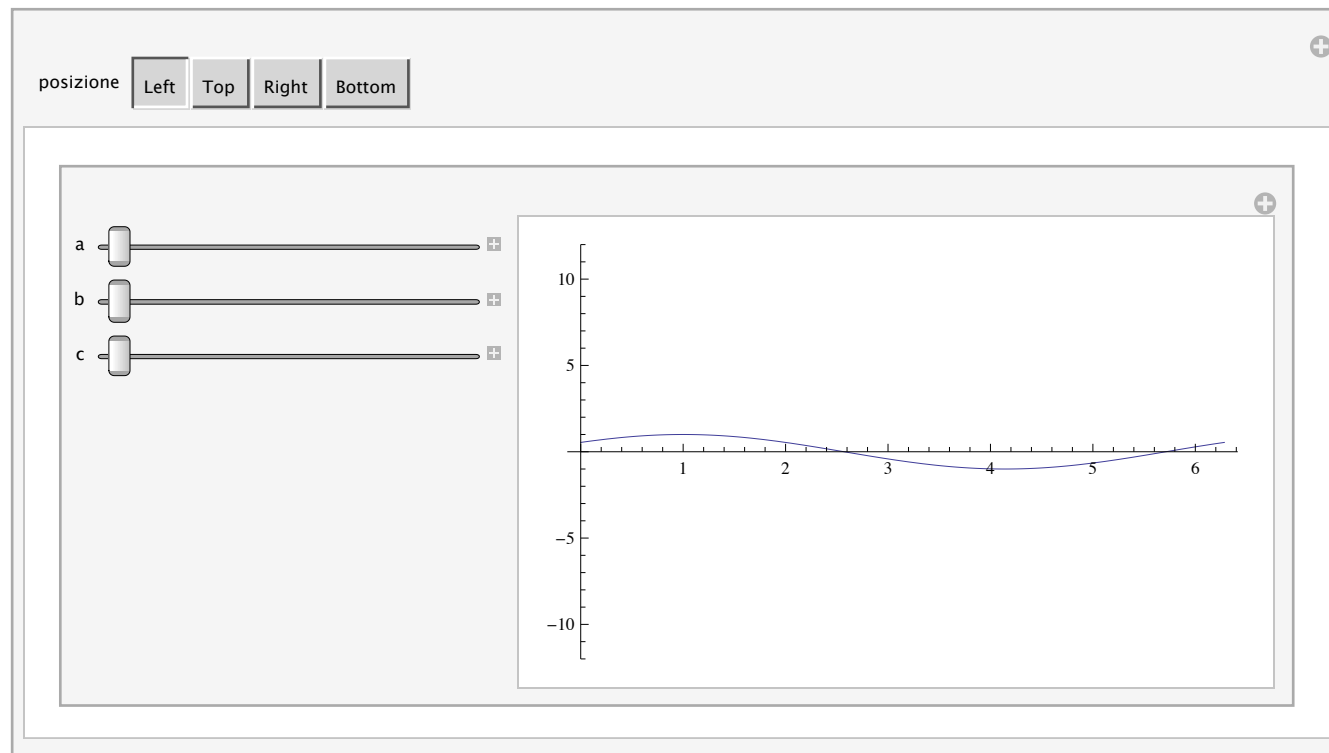


Posso spostare i controller in altri punti. Nota: per semplificarne la visualizzazione uso una Manipulate di Manipulate ;-)

```

Manipulate[
  Manipulate[
    Plot[a Cos[b (t - c)], {t, 0, 2 π}, PlotRange → 12],
    {a, 1, 12},
    {b, 1, 12},
    {c, 1, 12}, ControlPlacement → posizione],
  {posizione, {Left, Top, Right, Bottom}}]

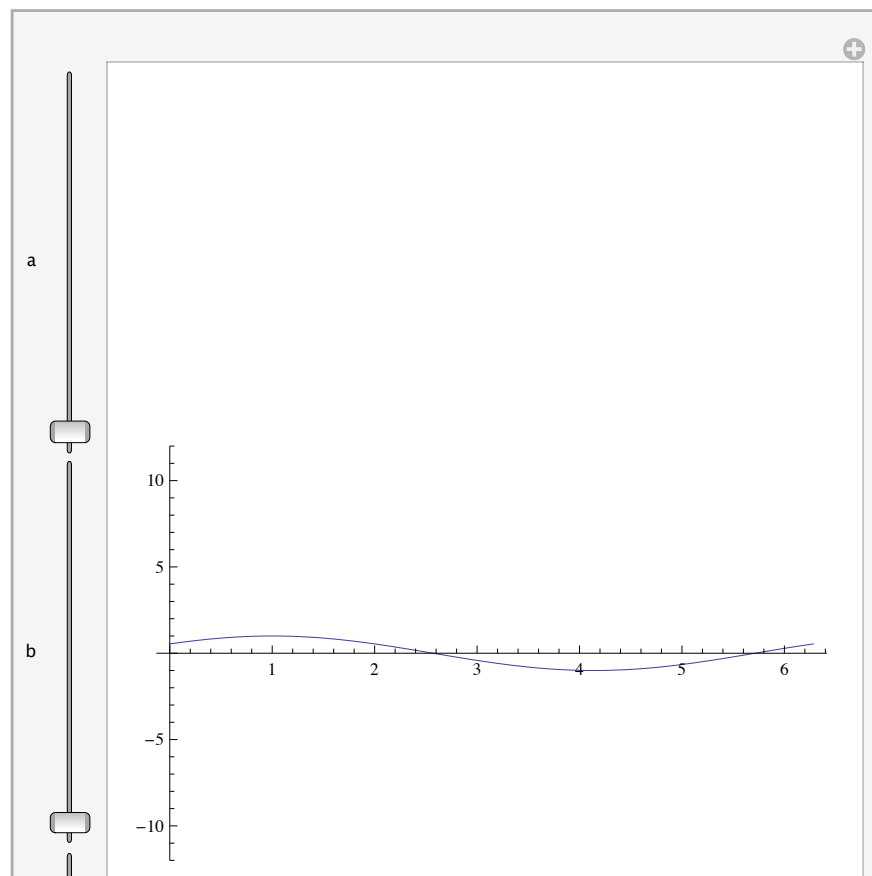
```

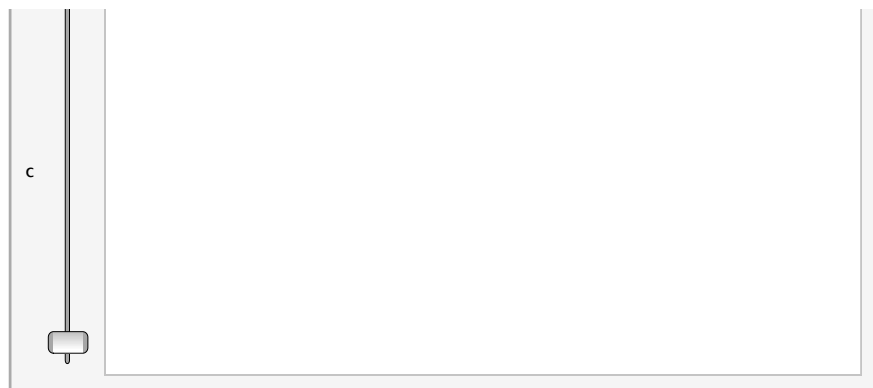


Si noti in particolare come sono “scomode” le posizioni Left e Right.

Proviamo il caso Left e proviamo a specificare che gli slider devono essere verticali, così forse l'apparenza migliora.

```
Manipulate[  
  Plot[a Cos[b (t - c)], {t, 0, 2 π}, PlotRange → 12],  
  {a, 1, 12},  
  {b, 1, 12},  
  {c, 1, 12},  
  ControlPlacement → Left,  
  ControlType → VerticalSlider]
```



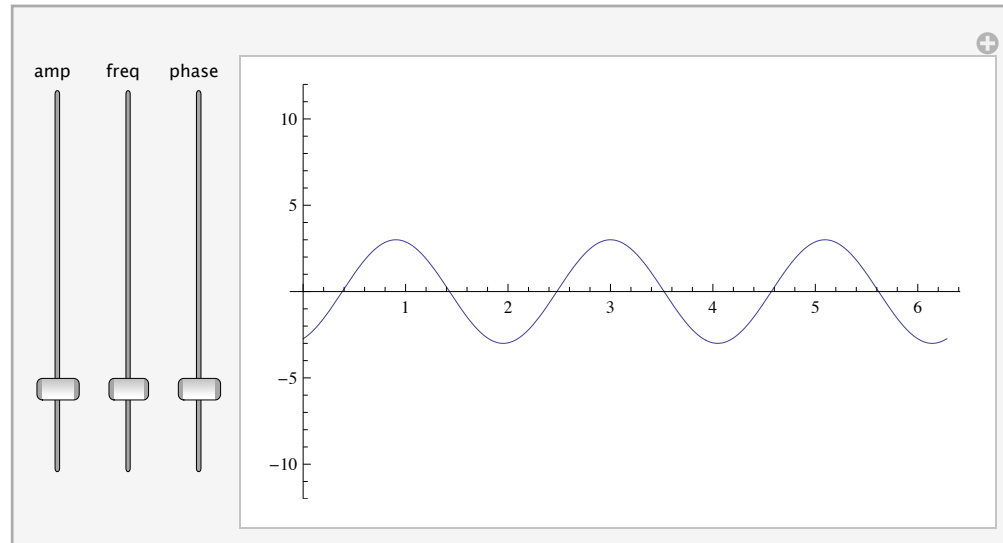


Bene, anzi male: **Manipulate** non ci soddisfa con le impostazioni di default dei controller. Possiamo ricorrere a **Control**.

```

Manipulate[Plot[a Cos[b (t - c)], {t, 0, 2 π}, PlotRange → 12],
  Grid[{{"amp", "freq", "phase"},
    {Control[{{a, 3, Null}, 1, 12, ControlType → VerticalSlider}},
    Control[{{b, 3, Null}, 1, 12, ControlType → VerticalSlider}},
    Control[{{c, 3, Null}, 1, 12, ControlType → VerticalSlider}}]},
  ControlPlacement → Left]

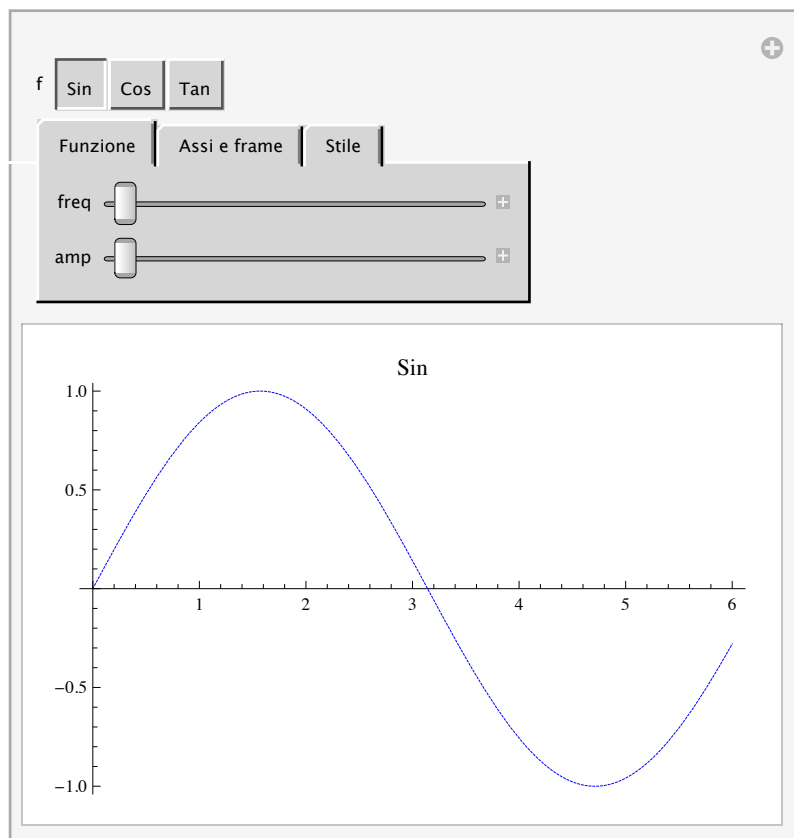
```



Un esempio leggermente più complesso. In questo caso è fondamentale non solo la posizione dei controller ma anche il loro raggruppamento in diverse sezioni, altrimenti l'utente potrebbe vedere troppi bottoni in una sola volta e fare confusione.

```
Manipulate[
  Plot[amp f[freq x], {x, 0, 6}, PlotStyle → {color, Dashing[dashing], Thickness[thickness]},
    Axes → axes, Frame → frame, AxesOrigin → axesorigin, PlotLabel → f],
  {f, {Sin, Cos, Tan}},
  TabView[{
    "Funzione" → Column[{Control[{freq, 1, 5}], Control[{amp, 1, 5}]}],
    "Assi e frame" → Column[{Control[{axes, {True, False}]},
      Control[{frame, {False, True}}], Control[{axesorigin, {0, 0}, {6, 1}}]}],
    "Stile" → Column[{Control[{color, Blue}], Control[{dashing, 0, 0.1}],
      Control[{thickness, 0.001, 0.1}]}]}, ImageSize → Automatic]
```





## Alcuni trucchi

Salvare le definizioni che intercorrono nella **Manipulate**.

Consideriamo un esempio di **Manipulate** dove abbiamo bisogno di definire una nostra funzione per poter eseguire dei calcoli (in questo esempio il calcolo dell'area di un triangolo dati i tre vertici).

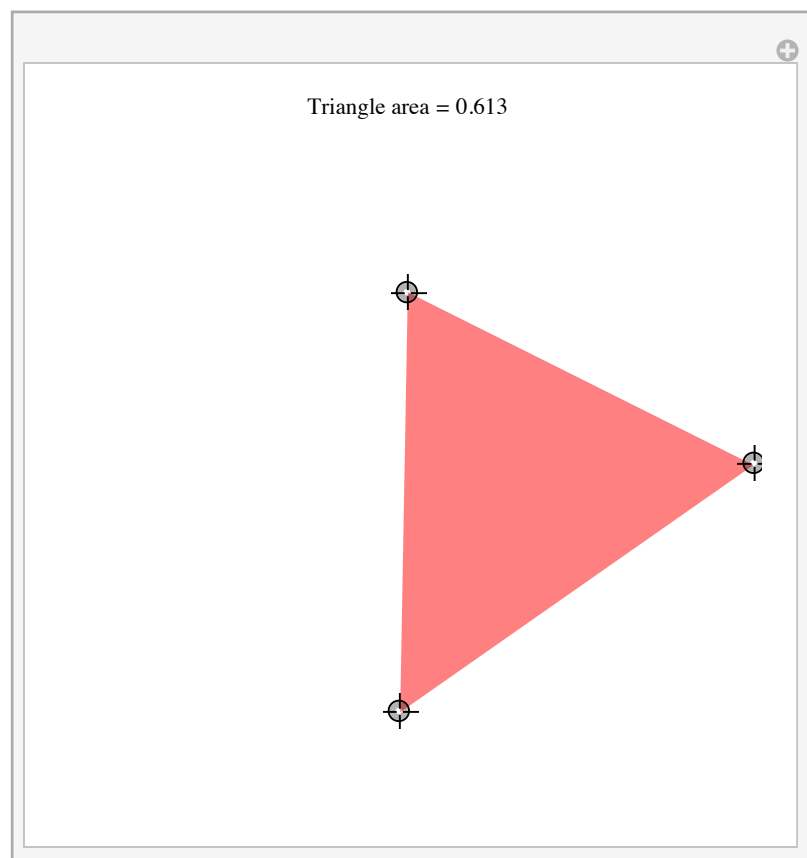
```
TriangleArea[{v1_, v2_, v3_}] := Abs[Det[Join[Transpose[{v1, v2, v3}], {{1, 1, 1}}]]] / 2
```

```
TriangleArea[{{x1, y1}, {x2, y2}, {x3, y3}}] // TraditionalForm
```

$$\frac{1}{2} |-x_2 y_1 + x_3 y_1 + x_1 y_2 - x_3 y_2 - x_1 y_3 + x_2 y_3|$$

Ora con la **Manipulate** vogliamo rendere dinamico il calcolo facendo variare all'utente la posizione dei vertici.

```
Manipulate[  
  Graphics[{Pink, Polygon[pts]}, PlotRange -> 1,  
    PlotLabel -> StringForm["Triangle area = `1`", TriangleArea[pts]]],  
  {{pts, {{0, 0}, {1, 0}, {0, 1/2}}}, Locator}]
```



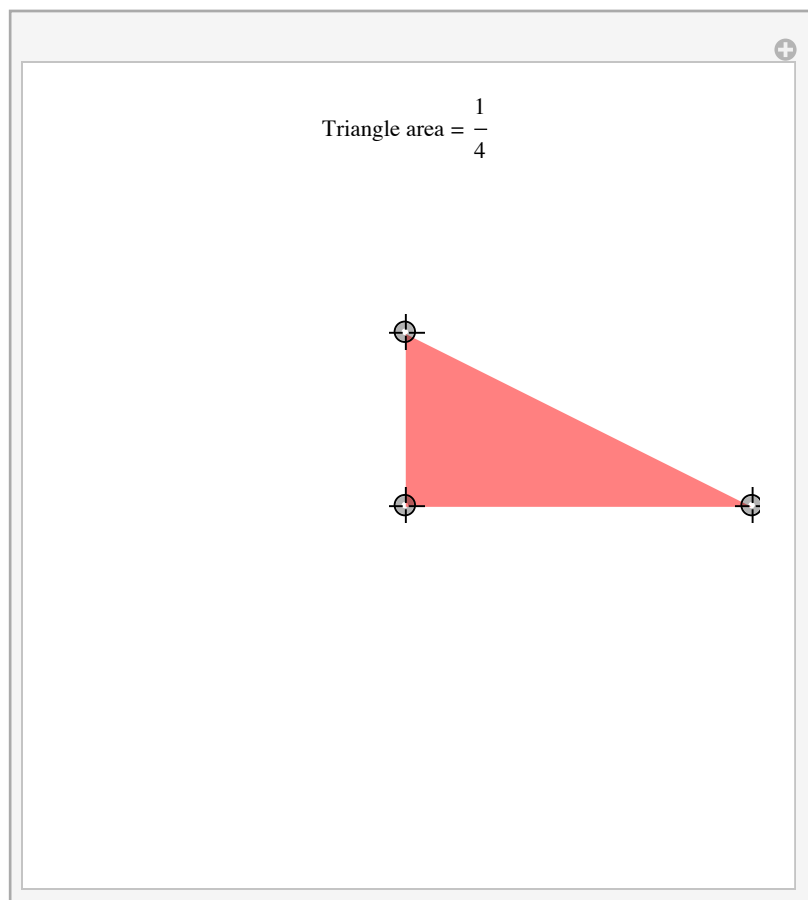
Cosa succede se chiudo il kernel?

```
Quit[]
```

L'alternativa è data dall'opzione **SaveDefinitions** che permette di salvare tutto il codice necessario al suo funzionamento dentro la **Manipulate** stessa.

```
TriangleArea[{v1_, v2_, v3_}] := Abs[Det[Join[Transpose[{v1, v2, v3}], {{1, 1, 1}}]]] / 2
```

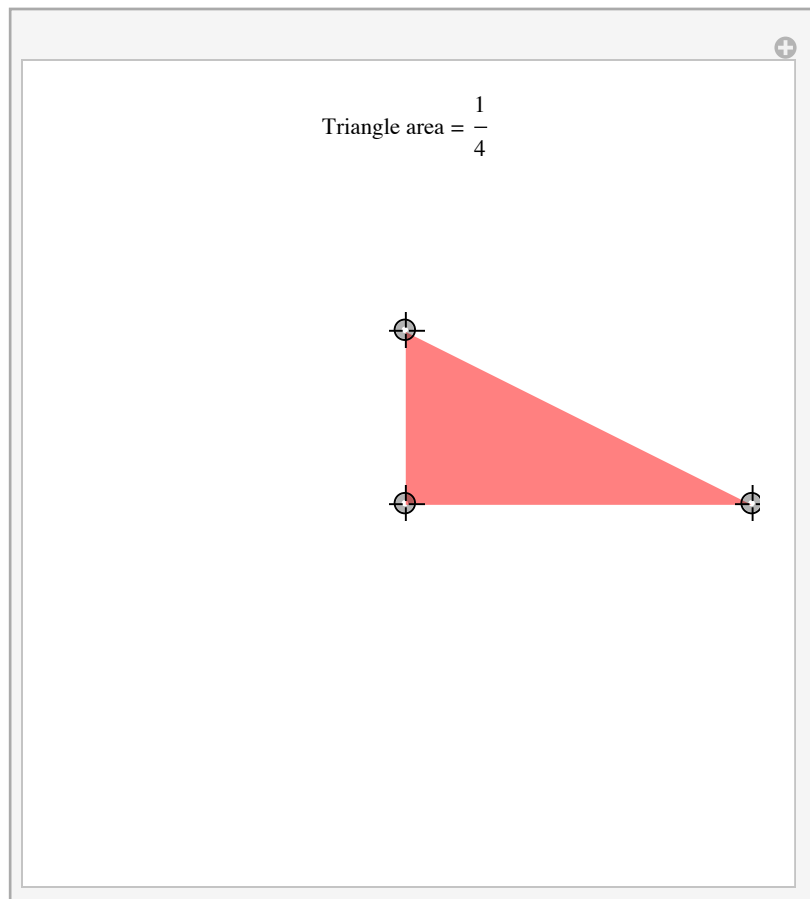
```
Manipulate[  
  Graphics[{Pink, Polygon[pts]}], PlotRange → 1,  
  PlotLabel → StringForm["Triangle area = `1`", TriangleArea[pts]],  
  {{pts, {{0, 0}, {1, 0}, {0, 1/2}}}, Locator},  
  SaveDefinitions → True  
]
```



`Quit[]`

Un'altra possibilità è data dall'opzione **Initialization**

```
Manipulate[
  Graphics[{Pink, Polygon[pts]}, PlotRange → 1,
    PlotLabel → StringForm["Triangle area = `1`", TriangleArea[pts]]],
  {{pts, {{0, 0}, {1, 0}, {0, 1/2}}}, Locator},
  Initialization :>
    (TriangleArea[{v1_, v2_, v3_}] := Abs[Det[Join[Transpose[{v1, v2, v3}], {{1, 1, 1}}]]] / 2)
]
```



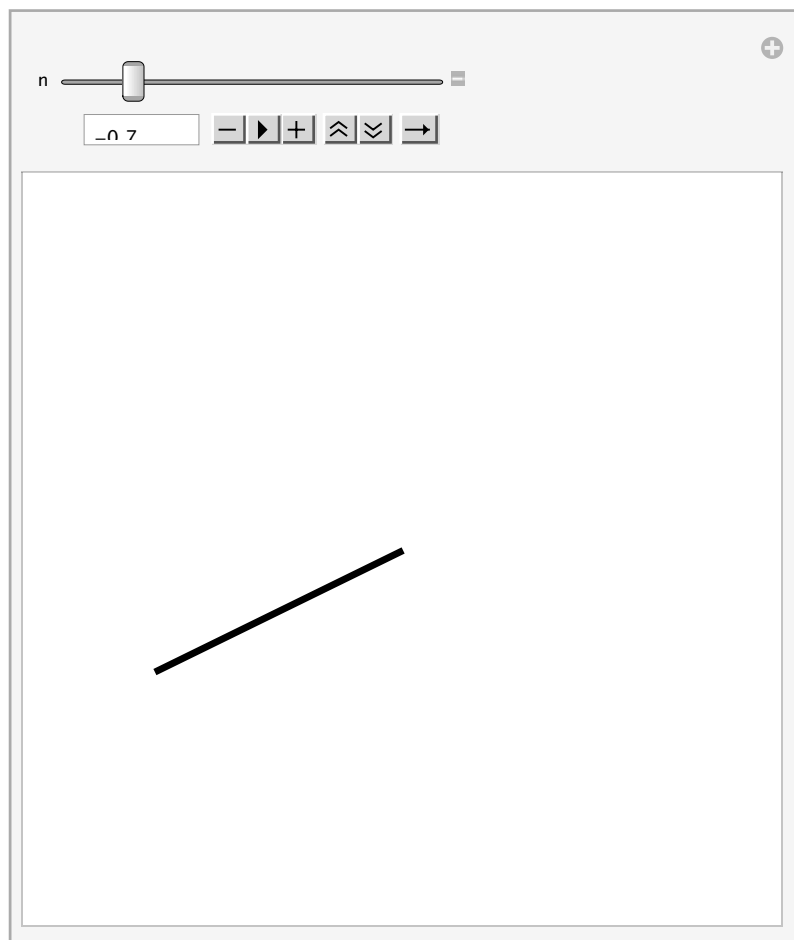
## Alcuni trucchi

Evitare continue ed inutili valutazioni del corpo di una **Manipulate**.

Frequentemente capita che la **Manipulate** si comporta apparentemente in modo strano, ossia valuta costantemente il suo corpo anche quando non dovrebbe perchè nessun controller viene toccato, e questo rallenta tutte le altre operazioni. Vediamo bene cosa accade in questi casi.



```
Manipulate[  
  f[x_] := x^3;  
  Graphics[{Thickness[0.01], Line[{{0, 0}, {n, f[n]}}]}, PlotRange -> 1],  
  {n, -1, 1}]
```



In effetti `Manipulate` si comporta correttamente secondo il suo principio di funzionamento. Infatti, il suo corpo

```
f[x_] := x^3;  
Graphics[{Thickness[0.01], Line[{{0, 0}, {n, f[n]}}]}, PlotRange -> 1]
```

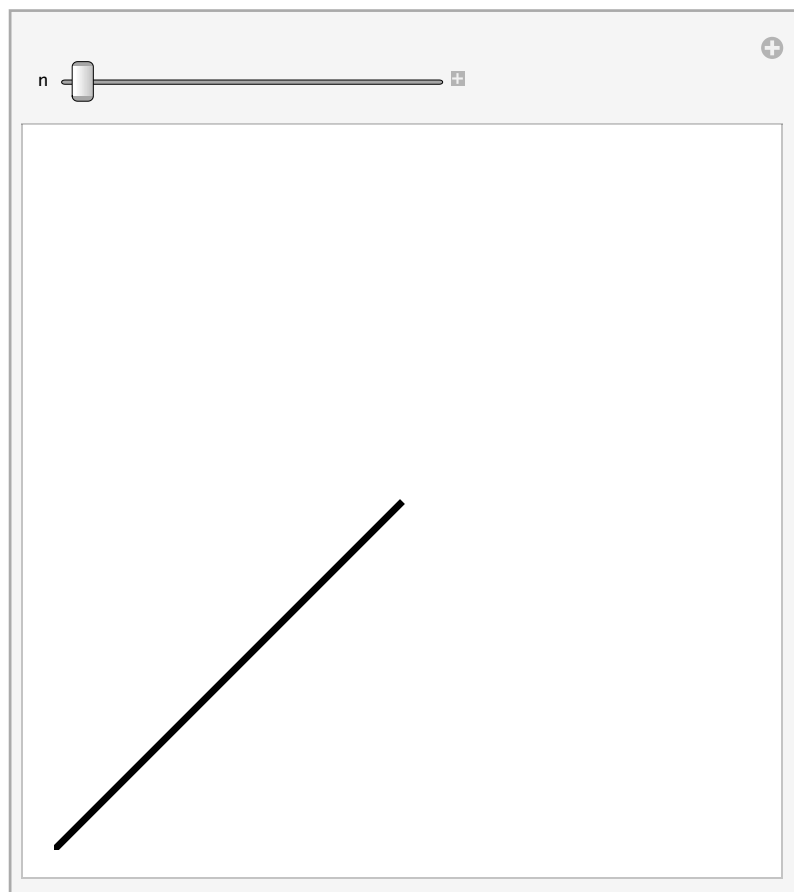
viene rivalutato, ma poi non smette più di essere rivalutato perchè esso modifica se stesso. `Manipulate` esegue il refresh del corpo ogni qualvolta esso subisce una variazione a causa di una qualsiasi modifica eseguita dal kernel o dal front end stesso (i controller). Ebbene assegnando un valore ad  $n$  si modifica il **Graphics** che richiamando  $f(n)$  modifica (ossia valuta) la funzione  $f$ , la quale fa scattare la modifica della **Graphics** e così in un loop infinito.

Per ovviare questo problema si può usare **Module** per isolare le variabili che non devono entrare nella “giostra” dei refresh.

```
Manipulate[  
  Module[{f},  
    f[x_] := x^3;  
    Graphics[{Thickness[0.01], Line[{{0, 0}, {n, f[n]}}]}, PlotRange -> 1]],  
  {n, -1, 1}]
```

Questa volta  $f$  non fa più *scattare* il refresh perchè essendo locale al **Module**, una volta valutato il corpo la  $f$  viene rimossa ed il grafico viene restituito. In alternativa, **Manipulate** consente di esplicitare direttamente quali sono le variabili da cui deve dipendere l’aggiornamento tramite l’opzione **TrackedSymbols**

```
Manipulate[  
  f[x_] := x^3;  
  Graphics[{Thickness[0.01], Line[{{0, 0}, {n, f[n]}}]}, PlotRange -> 1],  
  {n, -1, 1}, TrackedSymbols -> {n}]
```



## Alcuni trucchi

### » **Dynamic** al posto di **Manipulate**

**Manipulate** ha una sua struttura ben precisa del tipo

```
Manipulate[
    espressione(i)_da_animare,
    controller
]
```

In alcuni casi questa struttura potrebbe essere eccessivamente rigida perchè, ad esempio, vorremmo poter collocare un controller in una posizione diversa.

Una possibile alternativa in molti casi è data dalla **Dynamic** stessa come sostituto della **Manipulate** (d'altronde **Manipulate** viene convertita automaticamente in **Dynamic** dal sistema).

Esempio 6: un controller dentro il grafico

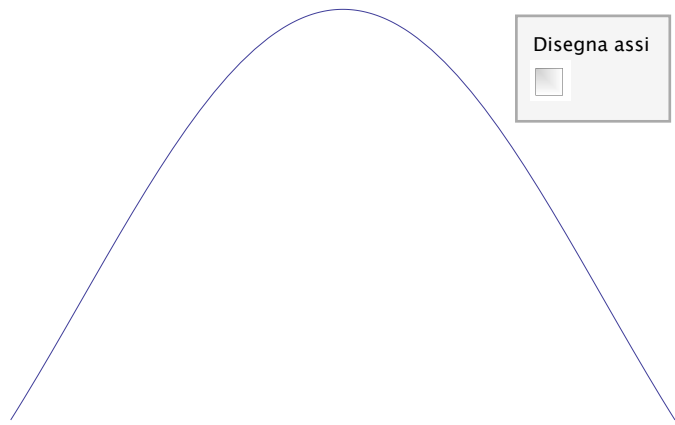
Voglio creare un grafico dinamico che mi permetta di scegliere se vedere o meno alcune proprietà, ma non voglio una sovra-struttura di tipo **Manipulate**

```
Manipulate[Plot[Cos[x], {x, -2, 2}, Axes → assi, ImagePadding → 0], {assi, {False, True}}
```



Se voglio il grafico senza la struttura della manipolate devo decidere quale controller utilizzare (**Checkbox**), costruire un elemento che lo contenga (**Panel**) e decidere dove collocarlo rispetto al grafico (**Inset**)

```
Dynamic[Plot[Cos[x], {x, -2, 2}, Axes → assi, ImagePadding → 0,  
  Epilog → Inset[Panel[Column[{"Disegna assi", Checkbox[Dynamic[assi]]}], {1.5, .8}]]]
```



Disegna assi



## Alcuni trucchi

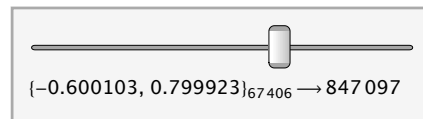
### » **Dynamic** è più flessibile di **Manipulate**

Quando si raggiunge una certa dimestichezza con le espressioni dinamiche di **Manipulate**, ci si rende conto che **Dynamic** per certi versi è più flessibile, in quanto è un costrutto più elementare e si può facilmente applicare a numerose espressioni o parti di esse.

Esempio 7: variabili con pedici dinamici

```
{n = 1000};
```

```
Dynamic[Panel[Column[{Slider[Dynamic[n], {1, 105, 1}], pDynamic[n] → Dynamic[Prime[n]]}]]]
```

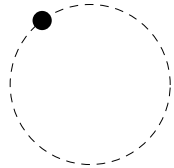


### » Applicare funzioni prima, dopo e durante la manipolazione dinamica

**Dynamic** permette di aggiungere una funzione da applicare in vari momenti dell'interazione, utile in alcuni casi per ottimizzare la modifica delle variabili dinamiche. Vediamo due esempi.

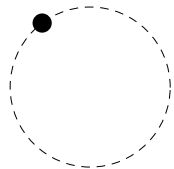
In questo primo esempio, vogliamo creare un locator all'interno di un grafico ma lo vogliamo anche vincolare a rimanere sulla circonferenza.

```
p = {0, 1};
Graphics[{Dashed, Circle[], PointSize[0.1], Point[Dynamic[p]]},
  ImageSize → Tiny, PlotRange → 1.2]
```



Di default **Point[Dynamic[p]]** ci fornisce un locator, ma ora dobbiamo vincolarlo a rimanere sulla circonferenza. Per fare questo possiamo usare **Normalize** per normalizzare il vettore rappresentato da  $p$ . Dunque abbiamo bisogno di applicare **Normalize** al valore di  $p$  ogni volta che muoviamo il locator. In casi come questi si usano gli argomenti aggiuntivi di **Dynamic**.

```
p = {0, 1};
Graphics[{Dashed, Circle[], PointSize[0.1], Point[Dynamic[p, (p = Normalize[#]) &]]},
  ImageSize → Tiny, PlotRange → 1.2]
```



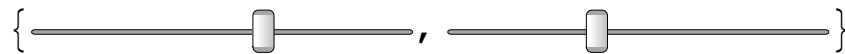


```
Graphics[Locator[Dynamic[p, (p = Normalize[#]) &]], PlotRange -> 2]
```

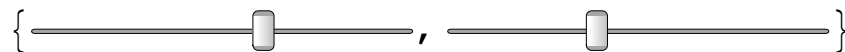


In questo altro esempio la funzione serve perchè altrimenti il secondo slider tenterebbe di assegnare un valore alla espressione  $(1-x)$

```
{Slider[Dynamic[x]], Slider[Dynamic[1 - x]]}
```



```
{Slider[Dynamic[x]], Slider[Dynamic[1 - x, (x = 1 - #) &]]}
```



## Alcuni trucchi

### » **Dynamic** è più flessibile di **Manipulate**

Esempio 8: monitorare una computazione tramite finestre con aggiornamenti dinamici

```
i = 2  $\pi$ ;  
CreatePalette[Dynamic[Plot[Cos[x], {x, -i, i}, PlotRange  $\rightarrow$  {{-10, 10}, {-1, 1}}]]];  
CreatePalette[Dynamic[Plot[Sin[x], {x, -i, i}, PlotRange  $\rightarrow$  {{-10, 10}, {-1, 1}}]]];  
CreatePalette[Dynamic[Plot[Tan[x], {x, -i, i}, PlotRange  $\rightarrow$  {{-10, 10}, {-10, 10}}]]];  
Table[i = RandomReal[{1, 10}]; Pause[1], {20}]
```

< | >

## Demonstrations: esempi avanzati

Sul sito [www.demonstrations.wolfram.com](http://www.demonstrations.wolfram.com), si possono trovare migliaia (oltre 7000) di applicazioni dinamiche ed interattive sviluppate con *Mathematica* da utenti di tutto il mondo.