OpenLaszlo:
Come definire il LAYOUT

Tag e attributi

- Proprio come *HTML*, anche *LZX* è basato su *tag* e *attributi*
- I **tag** definiscono i componenti/widget da posizionare sulla GUI (finestre, pulsanti, immagini...)
  - *Ad es. l'intera applicazione è contenuta nel tag:*

    <canvas> ..... **</canvas>**
- Gli **attributi** (*proprietà*) definiscono invece l'aspetto visuale (colore, posizione, dimensione...)

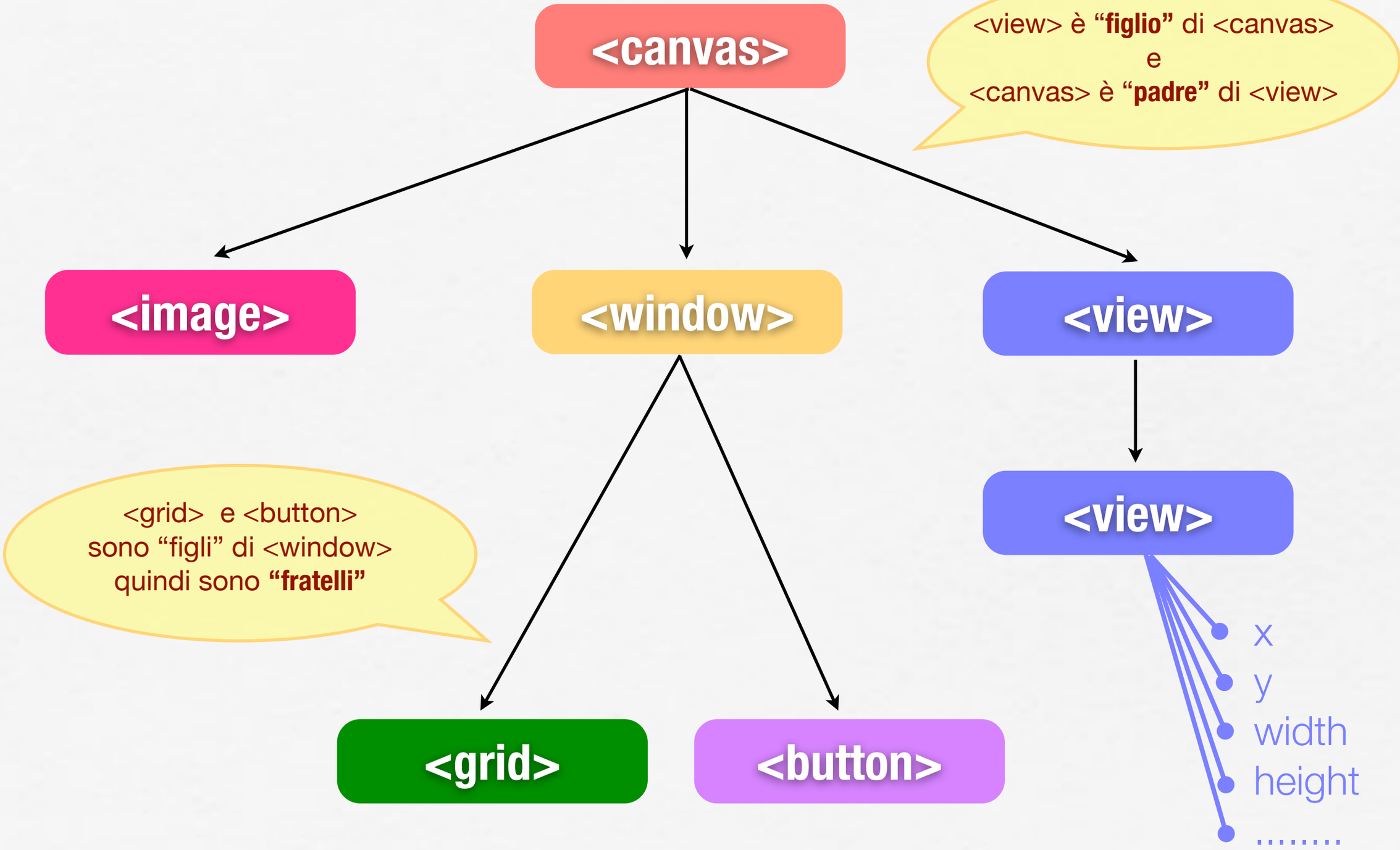
- I **tag** possono essere scritti in 2 forme:
  - **<TAG** …attributi… **>** …altri tag annidati… **</TAG>**
  - **<TAG** …attributi… **/>**
- Anche per gli **attributi** esistono 2 sintassi:
  - <TAG nomeAttributo**=**”valoreAttributo” >
  - <TAG>
    <attribute name=”nomeAttributo”
    type=”tipoAttributo”
    value=”valoreAttributo”
    />
    </TAG>

Document Object Model

- Per accedere agli elementi dell'interfaccia si usa il **DOM** (**D**ocument **O**bject **M**odel)

- E' una **rappresentazione ad albero** della struttura dell'interfaccia (*tag* e *attributi*)

- E' utilizzata per accedere e aggiornare dinamicamente il contenuto, la struttura e lo stile dei componenti dell'interfaccia


<canvas>
<view> è “figlio” di <canvas>
e
<canvas> è “padre” di <view>
<image>
<window>
<view>
<view>
x
y
width
height
.....
<grid> e <button>
sono “figli” di <window>
quindi sono “fratelli”
<grid>
<button>

- Ogni oggetto può essere identificato per mezzo di un **nome locale/globale**:
  - attributo *id*: scope globale a tutta l'applicazione
  - attributo *name*: *scope locale al "padre" che lo contiene*

- Si usa la classica **sintassi puntata** "." per accedere a oggetti e proprietà
  (es. *myWindow.width*, *myButton.xoffset* ecc)

- Si può navigare nella gerarchia anche con **parole chiave**: *canvas*, *this*, *parent*, *subviews*…
  (es. *canvas.myWindow.x*, *this.bgcolor*, *myButton.parent* ecc)

Il tag <view>

- E' l'oggetto di base di *OpenLaszlo*
- Consiste in un **box rettangolare**
  (equivalente a MovieClip in Flash e in HTML)
- **Contenitore** per oggetti multimediali
  (immagini, audio, video ecc)
- Elemento visuale di base grazie al quale costruire componenti complessi
- **Tag annidati**: sistema di riferimento è relativo al "padre" (angolo alto-sinistra)

Principali attributi

- Posizione e dimensione:
  width, *height*, *x*, *y*, *rotation*, *xoffset*, *yoffset*
- Aspetto:
  bgcolor, *fgcolor*, *opacity*, *visible*, *clip*
- Risorse multimediali:
  source, *resource*, *stretches*
- *Risorse video/animazioni Flash:*
  frame, *totalframes*, *playing*

Risorse multimediali

- E’ possibile caricare **risorse multimediali** (immagini, suoni, video) dentro una *<view>* per mezzo degli attributi *source*/*resource*
- Una risorsa può essere caricata:
  - a tempo di esecuzione (**run-time**)
    La risorsa viene caricata solamente al momento dell’inizializzazione della view che la contiene
  - a tempo di compilazione (**compile-time**)
    La risorsa viene inglobata nel binary dell’applicazione ed è quindi inclusa nel download iniziale dell’applicazione stessa

- Caricamento a **run-time**:
  - Si specifica la risorsa da caricare come **url HTTP**:
    <view **resource="http:**//www.sito.com/immagine.jpeg**"** />
    <view **resource="http:**../icone/disk.jpeg**"** />
  - Oppure usando l'attributo **source** *(è sempre run-time)*:
    <view **source="**www.sito.com/immagine.jpeg**"** />
  - La risorsa viene caricata solo al momento dell'inizializzazione della *view* oppure quando l'attributo *resource* cambia valore
  - Il download iniziale dell'applicazione è più veloce, ma risorsa potrebbe tuttavia essere visualizzata con un ritardo a causa di rallentamenti nella rete

- Caricamento a **compile-time**:
  - La risorsa viene caricata attraverso un tag *<resource>* ausiliario *(identificato da un nome)* oppure specificando nell'attributo *resource* il path locale del fileda caricare:
  - **<resource name="**nomeRisorsa**" src="**media/icon.jpeg**"/>**
    <view **resource="**nomeRisorsa**"** />
    ..... oppure .....
    <view **resource="**../icone/disk.jpeg**"** />
  - La risorsa viene inglobata nell'applicazione stessa, quindi i tempi di download sono maggiori ma la visualizzazione della risorsa sarà istantanea
  - Ideale per risorse usate spesso (icone, suoni ecc)

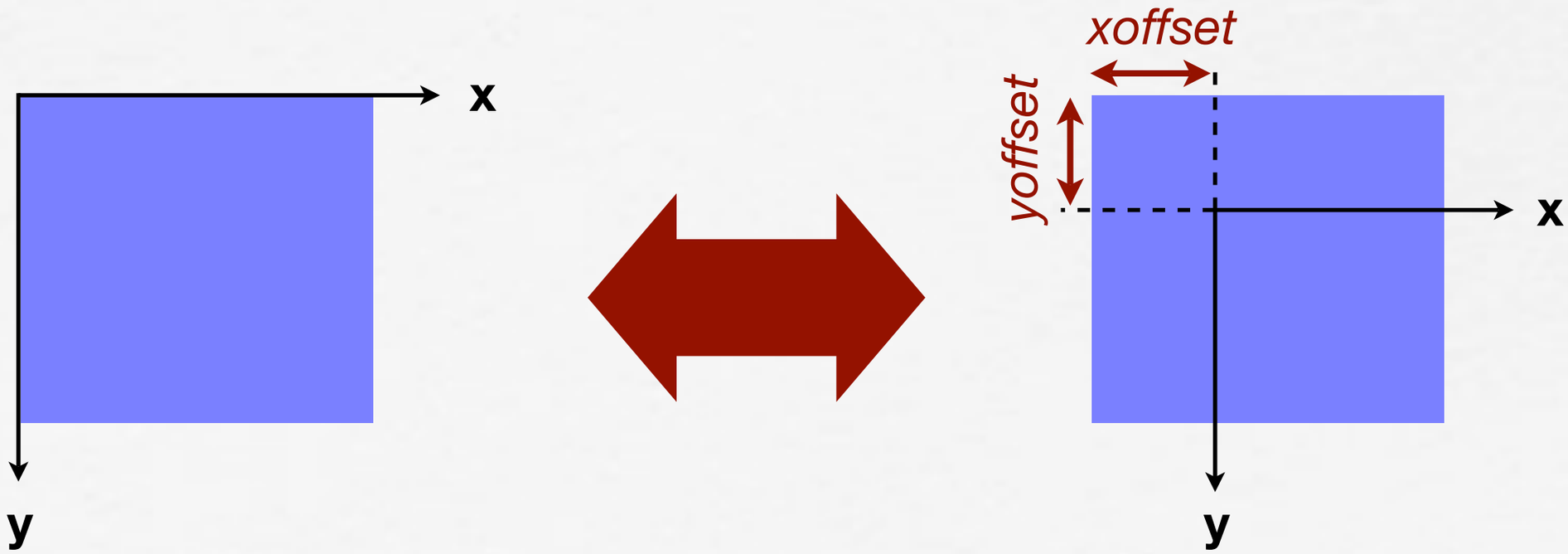
- Con il tag *<resource>* è possibile caricare **risorse multi-frame**, ovvero composte da più sotto-risorse:
  - Le sotto-risorse sono memorizzate in **frame** separati attraverso uno o più tag *<frame>*
  - Si accede ai vari frame per mezzo dell'attributo *frame*:

```
<resource name="mylcons">
      <frame src="resources/pippo.jpeg" />
      <frame src="resources/pluto.jpeg" />
</resource>

<view resource="mylcons" frame="1" />
<view resource="mylcons" frame="2" />
```

Note su *xoffset*/*yoffset*

Normalmente il **sistema di riferimento** di un oggetto è relativo all'*angolo in alto a sinistra*. *xoffset* e *yoffset* permettono di spostarlo:



Note su *stretches*

Quando si carica una immagine si possono forzare le sue dimensioni ad **adattarsi al suo contenitore**:

“none”

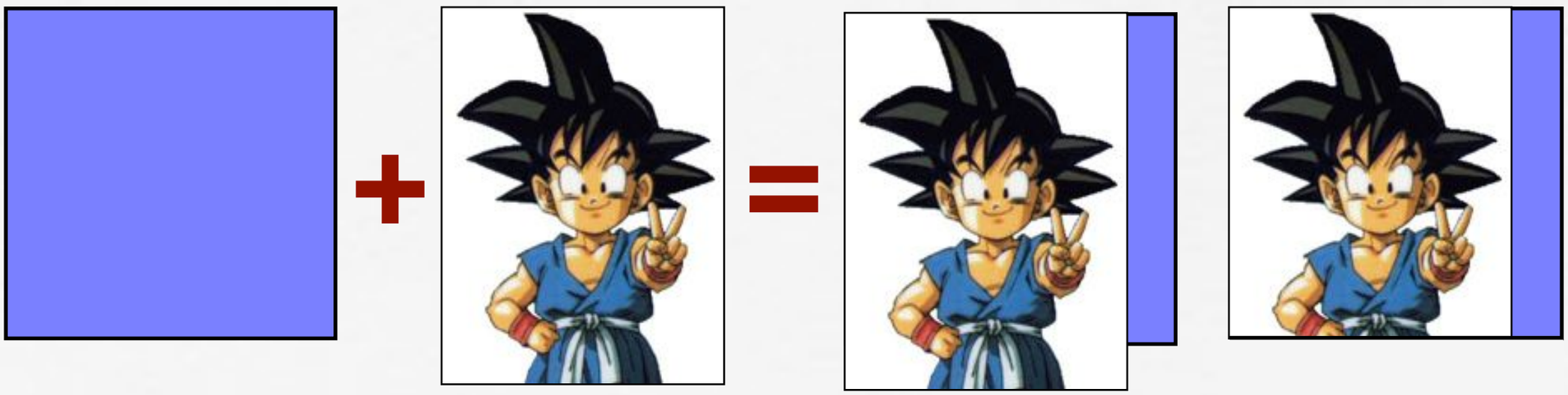
“width”

“height”

“both”

Note su *clip*

Quando il contenuto di un oggetto eccede le dimensioni dell'oggetto stesso si può **nascondere il contenuto in eccesso**:



"false" "true"

I tag <layout>

- Sono **elementi funzionali** che servono per **distribuire ed allineare automaticamente** i tag “figli” di un oggetto (es. <view>)

- Esempi:

  <simplelayout axis=”x” spacing=”10” **/>**

  Allinea tutti i componenti lungo l’asse *x*, con una spaziatura costante di 10 pixel tra di essi

  <wrappinglayout axis=”x” spacing=”10” **/>**

  Come prima, ma arrivato al margine destro dell’oggetto padre, manda a capo i restanti componenti sulla riga sottostante in automatico

Le classi

- E’ possibile definire **tag personalizzati**:
  <class name=”nomeNuovoTag**”>**
  …definizione degli attributi e dei metodi…
  </class>
- Poi si utilizzano come di consueto:
  <nomeNuovoTag …attributi… **>**
  …altri tag/compnenti annidati…
  </nomeNuovoTag>

- Esempio:

```
<view bgcolor=”red” width=”50” height=”50” x=”0” />
<view bgcolor=”red” width=”50” height=”50” x=”100” />
<view bgcolor=”red” width=”50” height=”50” x=”200” />
```

...si può riscrivere come...

```
<class name=”myView”
      bgcolor=”red” width=”50” height=”50”
/>

<myView x=”0” />
<myView x=”100” />
<myView x=”200” />
```

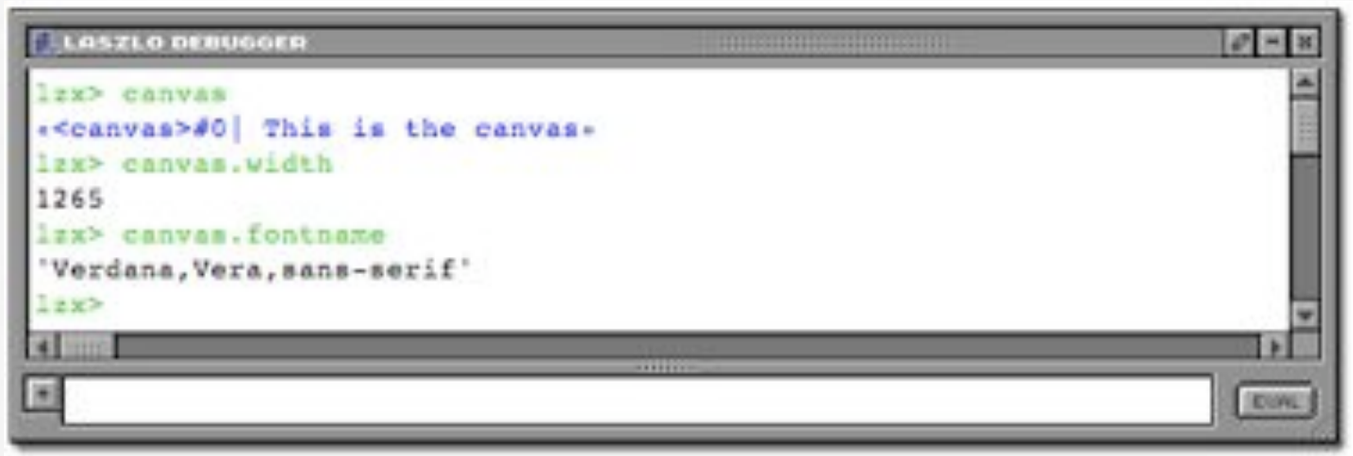
- Nella definizione è possibile inserire:
  - **attributi**: proprietà/caratteristiche dell'oggetto ← **Layout**
    <attribute name="....." type="....." value="....." />
  - **metodi**: funzioni che l'oggetto può compiere ← **Logica**
    <method name="....." > ..... </method>
  - **eventi**: porzioni di codice da eseguire al verificarsi di particolari condizioni (es. al click del mouse, al caricamento di un'immagine...)
    <handler name="....." > ..... </method>
- Le classi possono essere definite in **file esterni** attraverso il tag *<library>....</library>*. E' poi possibile includerle nel *<canvas>* tramite il tag *<include href="*...path al file lzx...*" />*

Debugger

- OpenLaszlo mette a disposizione un **debugger integrato**:

- Per abilitarlo è sufficiente settare l'attributo debug="true" del canvas
- Disponibile solo se si è connessi al server
  (cioè in modalità proxied)

- **Ispezionare** oggetti, attributi, espressioni:



es:
canvas
canvas.width / 2
this.bgcolor
Math.max(this.x,parent.x)

- Stampare **messaggi** informativi:

es:
Debug.info(‘....’)
Debug.warn(‘...’)
Debug.error(‘...’)
Debug.write(‘....’)

Per fare delle prove

1) Andate sul sito www.openlaszlo.org
2) DOCUMENTATON -> Reference
3) Cercare l'help del tag <view>
4) Nel primo esempio, cliccate "EDIT"
5) A questo punto avrete a disposizione un **editor/compilatore online** per fare i vostri esperimenti

1° esercitazione

1. Inserire delle **<view>** sulla GUI, modificare gli attributi di posizione, dimensione e aspetto, e vedere gli effetti che si ottengono
2. Sovrapporre due o più <view> e vedere l'effetto della proprietà **opacity**
3. Provare a replicare degli oggetti con caratteristiche simili con l'uso delle **classi**
4. Posizionare alcuni componenti a scelta con/senza l'aiuto del tag **simplelayout**

I constraints

- Servono a **vincolare il valore di un attributo** a quello di altri attributi *(anche di altri oggetti)*
- Sintassi:

  <tag …… attributo**=”${** espressione **}”** …… **/>**
- Esempio:

```
<view width=”50” height=”${this.width}”>
        <view width=”${parent.width/2}” height=”${parent.height/3}” />
</view>
```

- Molto comodi e pratici, ma non abusarne

  (un uso esagerato degrada le prestazioni dell’interfaccia)

Il tag <animator>

- Serve per **“animare” un oggetto** in un certo intervallo di tempo, ad es.
  - muovere l’oggetto:
    incrementare/diminuire i suoi attributi *x* o *y*
  - ottenere effetti di fading:
    abbassare gradualmente l’attributo *opacity*
  - ma è possibile “animare” **qualsiasi attributo** numerico!
- Sintassi:

```
<animator attribute=”nomeAttributo”
    from=”value” to=”value” duration=”msec” />
```

- Se si vogliono **animare più attributi**:

  <animatorgroup process**="**.....**">**

  <animator attribute=”attributo_1” ..... />

  <animator attribute=”attributo_2” ..... />

  </animatorgroup>

- Con process si specifica come devono essere eseguite le singole animazioni:
  - *“simultaneous”*
  - *“sequential”*
- I tag *<animatorgroup>* possono essere **annidati** per animazioni più complesse

2° esercitazione

1. Inserire un componente a scelta e permetterne la modifica dello stato **visualizzato/nascosto** a seconda che una **checkbox** sia spuntata o meno
2. Creare una **finestra** ridimensionabile, ed al suo interno posizionare qualche componente le cui dimensioni/posizioni siano legate a quelle della finestra stessa
3. Creare delle animazioni di un componente a scelta che coinvolga più attributi