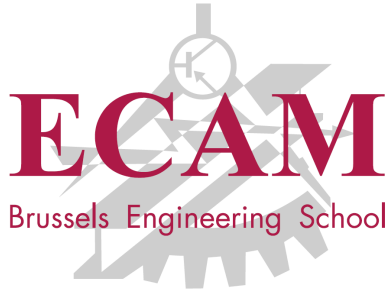


B201A Informatique appliquée

Séance 4

Interface graphique et évènements

Sébastien Combéfis, Quentin Lurkin

2017–2018

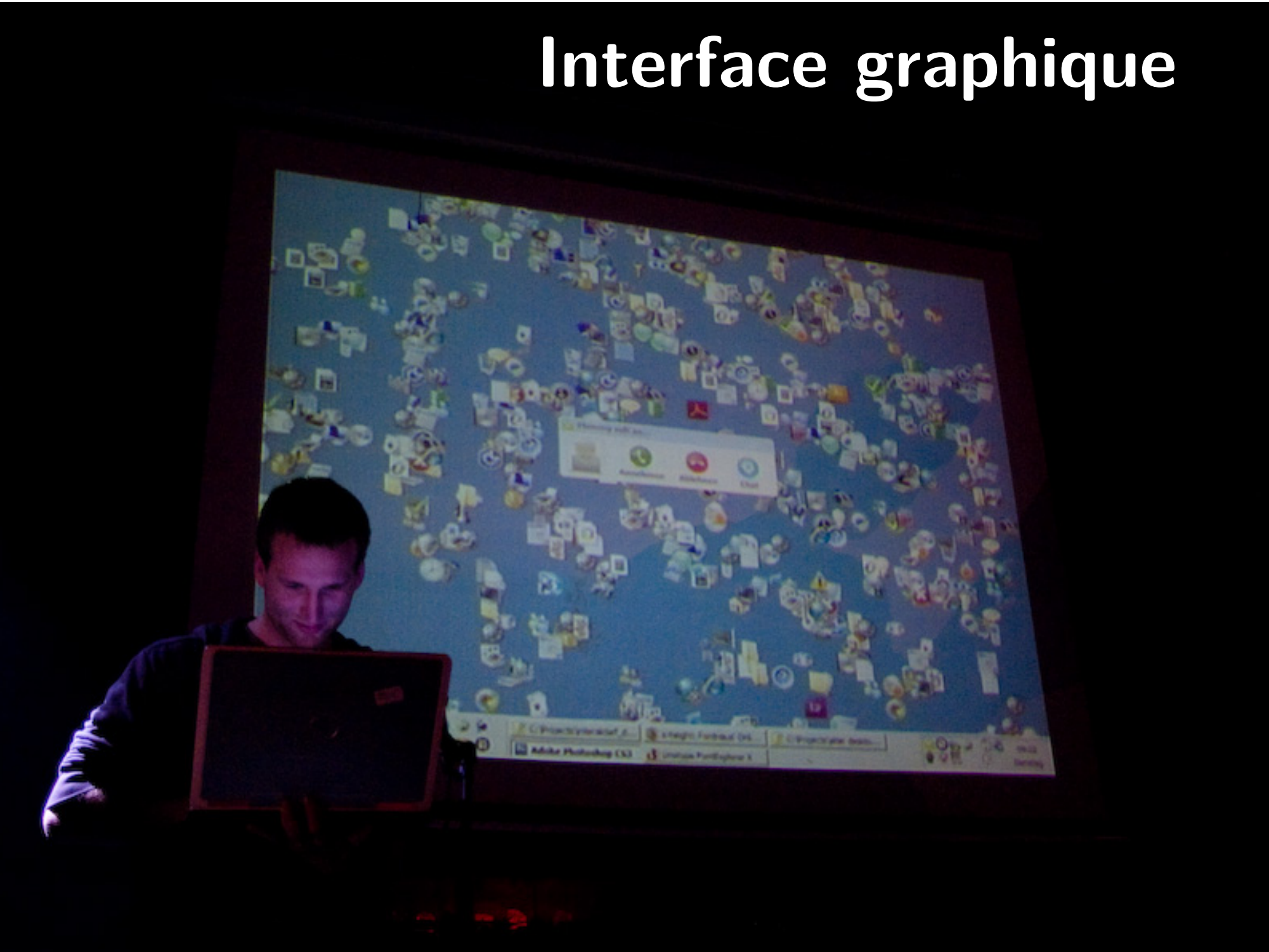
Rappels

- Définition d'une classe
  - Définition d'une classe (variable d'instance et constructeur)
  - Définition d'un constructeur et instanciation d'un objet
  - Définition de méthode et appel
- Concepts de programmation orientée objet
  - Représentation d'un objet avec `__str__`
  - Surcharge d'opérateur
  - Égalité des objets (==) et des identités (`is`)
  - Accesseur et mutateur

Objectifs

- Interface graphique
  - Construction d'une interface graphique avec Kivy
  - Widgets graphiques
  - Fichier KV
- Programmation évènementielle
  - Boucle d'évènement
  - Action et gestionnaire d'évènements
  - Canvas et dessin

Interface graphique



Librairie Kivy

- Framework open-source pour créer des interfaces utilisateur

  Application desktop ou mobile, jeux...

- Plusieurs avantages offerts par la librairie
  - Multi-plateforme (Linux, Windows, OS X, Android, iOS)
  - Framework stable, API documentée...
  - Moteur graphique basé sur OpenGL ES 2 (utilisation du GPU)

- Kivy est disponible sur GitHub

  https://github.com/kivy/kivy

Graphical User Interface (1)

- Utilisation de la librairie graphique Kivy pour les GUI

  Importation de `kivy.app` et `kivy.uix.*`

- Création d'une fenêtre en définissant une classe

  Méthode `build` permet de construire l'interface

```
1  from kivy.app import App
2  from kivy.uix.label import Label
3
4  class HelloApp(App):
5      def build(self):
6          return Label(text='Hello World!', font_size='100sp')
7
8  HelloApp().run()
```

Graphical User Interface (2)

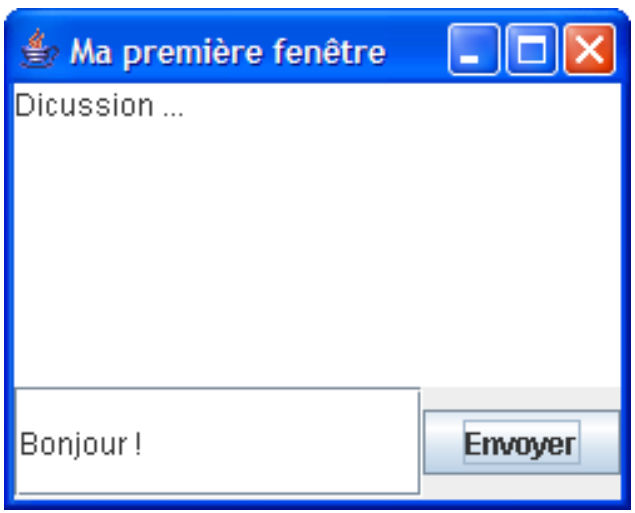
Widget

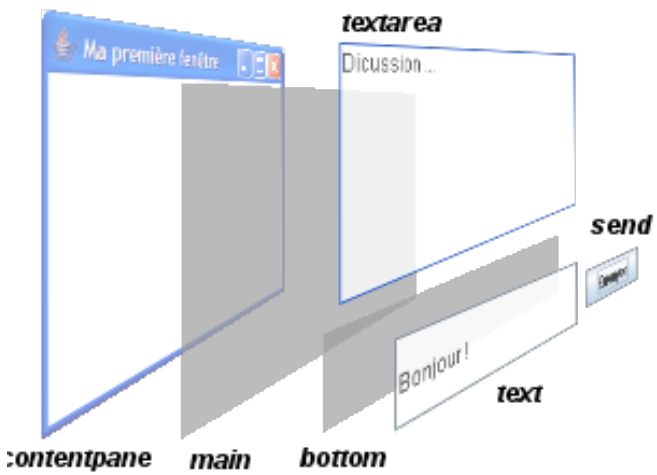
- Interface graphique construite à partir de widgets

  Composants tels que label, bouton, fenêtre, liste, case à cocher...

- Widgets placés les uns dans les autres

  Chaque composant a un composant parent





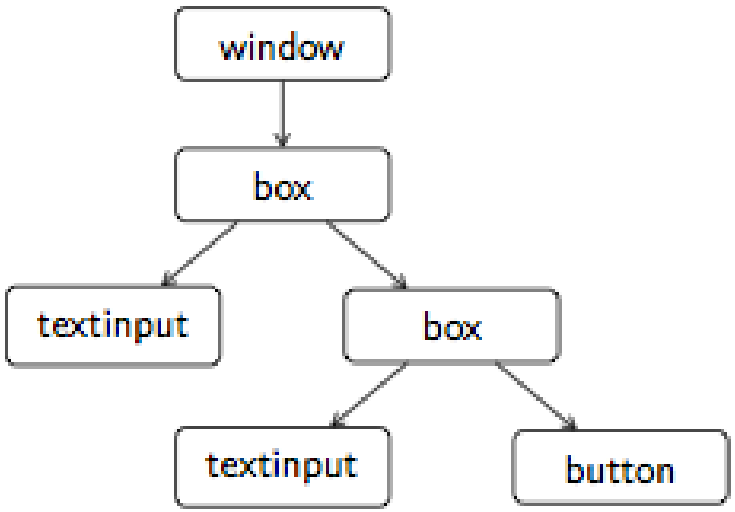
Hiérarchie de Widget

- Liens entre les composants d'une fenêtre comme un arbre

  Hiérarchie des relations de filiation des widgets

- Un composant possède un parent et une liste de fils

  L'organisation visuelle dépend du type de parent



Composants de base (1)

- Un label est une zone de texte non modifiable

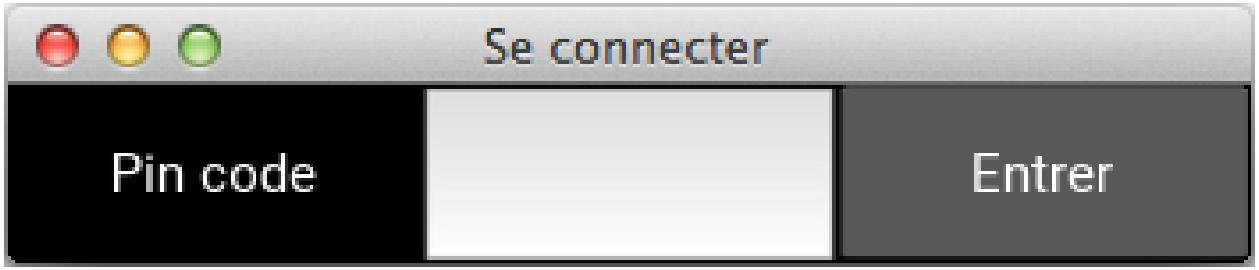
  Permet d'afficher du texte pour information à l'utilisateur

- Une zone de texte permet à l'utilisateur d'entrer un texte

  Permet de récupérer une information de l'utilisateur

- Un bouton peut être cliqué

  Permet de réagir à une action de l'utilisateur



Composants de base (2)

```
1  from kivy.app import App
2  from kivy.config import Config
3  from kivy.uix.boxlayout import BoxLayout
4  from kivy.uix.button import Button
5  from kivy.uix.label import Label
6  from kivy.uix.textinput import TextInput
7
8  class LoginApp(App):
9      def build(self):
10         self.title = 'Se connecter'
11         box = BoxLayout(orientation='horizontal')
12         box.add_widget(Label(text='Pin code'))
13         box.add_widget(TextInput())
14         box.add_widget(Button(text='Entrer'))
15         return box
16
17 # Configuration de la taille de la fenêtre
18 Config.set('graphics', 'width', '350')
19 Config.set('graphics', 'height', '50')
20
21 # Lancement de l'interface graphique
22 LoginApp().run()
```

BoxLayout

- Un box layout n'a aucun rendu graphique

  Uniquement utilisé pour y stocker d'autres composants

- Ajout d'un composant à l'aide de la méthode `add_widget`
  - Les composants sont ajoutés l'un à la suite de l'autre
  - Organisation horizontale ou verticale des composants

```
1 box = BoxLayout(orientation='horizontal')
2 box.add_widget(Label(text='Pin code'))
3 box.add_widget(TextInput())
4 box.add_widget(Button(text='Entrer'))
```

Gestionnaire de mise en page

- Plusieurs composants de gestion de mise en page
  - `BoxLayout` aligne horizontalement ou verticalement
  - `GridLayout` organise sous forme de lignes et colonnes
  - `FloatLayout` permet un positionnement absolu
- On peut combiner plusieurs conteneurs

  Pour créer une mise en page complexe

GridLayout (1)

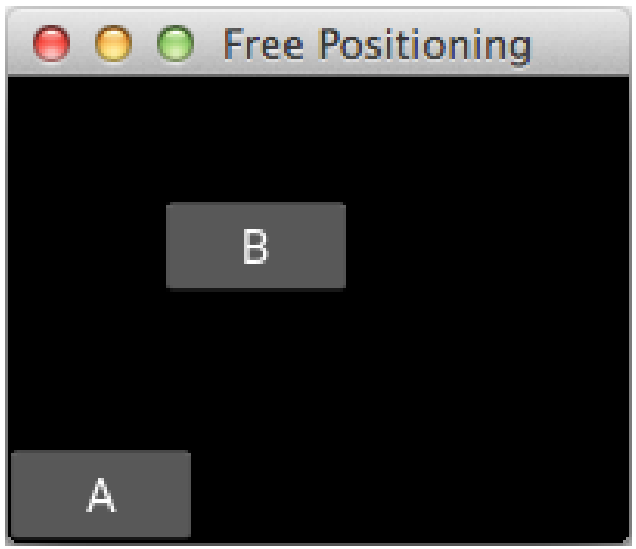
- Organisation des composants fils sous forme d'une grille
  - Choix fixe du nombre de lignes (`rows`) et colonnes (`cols`)
  - Composants fils ajoutés de haut en bas, de gauche à droite

GridLayout (2)

```
1  class GridGameApp(App):
2      def build(self):
3          self.title = 'Grid Game'
4          grid = GridLayout(rows=3, cols=4)
5          for i in range(12):
6              grid.add_widget(Button(text=str(i + 1)))
7          return grid
```

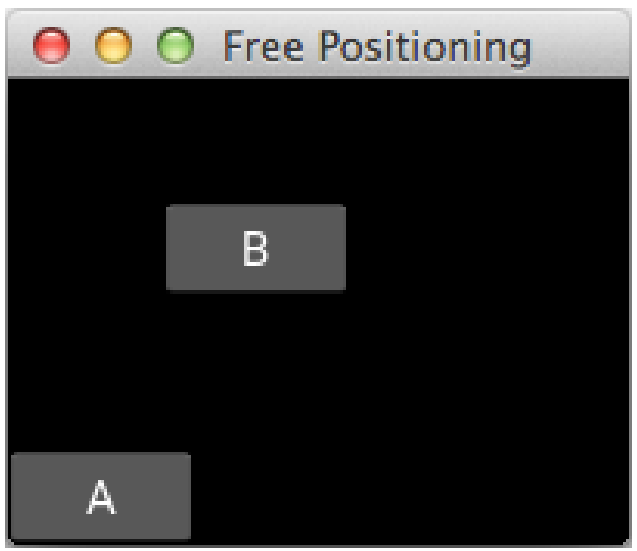
FloatLayout (1)

- Organisation des composants fils de manière libre
  - Précision de la coordonnée de positionnement (`pos`)
  - La coordonnée $(0, 0)$ est le coin inférieur gauche



FloatLayout (2)

```
1  class FreePosApp(App):
2      def build(self):
3          self.title = 'Free Positioning'
4          box = FloatLayout(size=(200, 150))
5          box.add_widget(Button(text='A', size_hint=(0.3, 0.2), pos
           =(0, 0)))
6          box.add_widget(Button(text='B', size_hint=(0.3, 0.2), pos
           =(50, 80)))
7          return box
```



Application graphique

- Code d'une application graphique placé dans une classe

  La classe doit être « de type » `App`

- Lancement de l'application par la méthode `run`

- Possibilité de décrire l'interface avec le langage KV

  Langage balisé de description d'interfaces graphiques

```
1  from kivy.app import App
2
3  class LoginApp(App):
4      pass
5
6  LoginApp().run()
```

Langage KV

- Fichier `.kv` qui porte le même nom que l'application

  Donc `login.kv` dans notre exemple

- Liste des composants avec leurs propriétés

  Label, champ texte, layout, onglets...

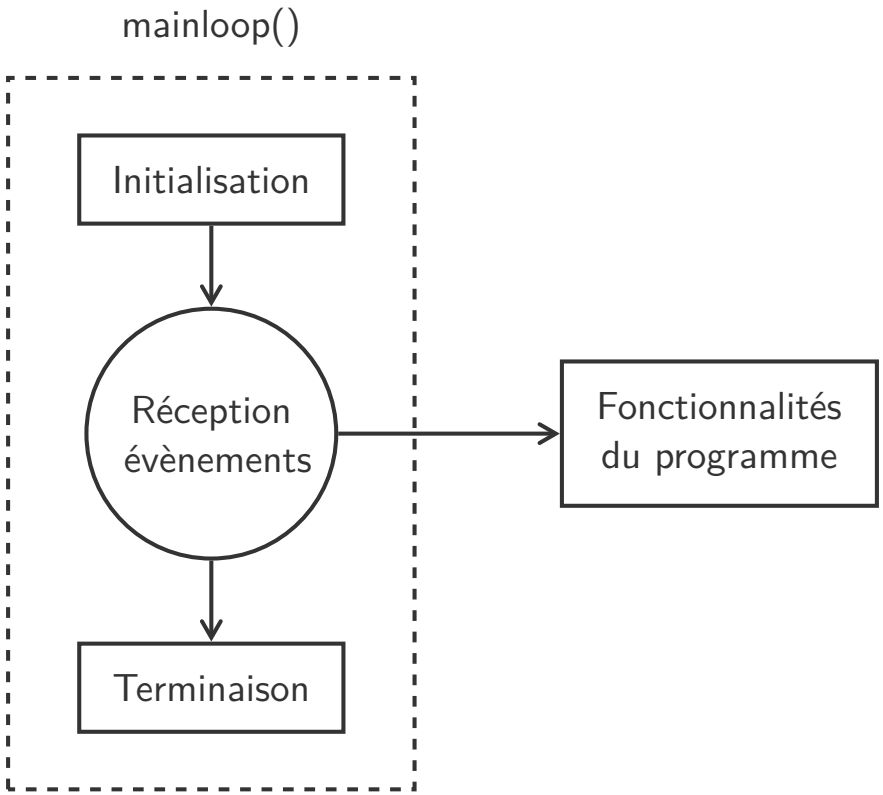
```
1 BoxLayout:
2     orientation: 'horizontal'
3     Label:
4         text: 'Pin code'
5     TextInput
6     Button:
7         text: 'Entrer'
```

Programmation évènementielle

Boucle d'évènement

- Des évènements sont produits par l'utilisateur

  Il faut associer les évènements à un gestionnaire



Gestionnaire d'évènement (1)

- Pour traiter un évènement, il faut lui associer un gestionnaire

  Une portion de code à exécuter lorsque l'évènement se produit

- Intervention de trois éléments
  - La source de l'évènement (un widget)
  - L'évènement à proprement parler (clic, pression de touche...)
  - Le gestionnaire d'évènements (le code à exécuter)

- Informations complémentaires selon l'évènement

  Coordonnées pour un clic de souris, touche pour une pression...

Gestionnaire d'évènement (2)

- Associer une fonction à une action d'un composant

  Via la méthode `bind` du widget

```
1  class QuitApp(App):
2      def build(self):
3          self.title = 'Quit me'
4          box = AnchorLayout(anchor_x='center', anchor_y='center')
5          quit = Button(text='Quitter', size_hint=(0.7,0.3))
6          quit.bind(on_press=self._quit)
7          box.add_widget(quit)
8          return box
9
10     def _quit(self, source):
11         sys.exit(0)
```

Gestionnaire d'évènement (3)

- Exécution de la fonction associée lors de l'évènement

  Ici, lors du clic sur le bouton quitter

Application graphique (2)

- Bind du gestionnaire d'évènements à partir du fichier KV

Accès à l'application courante depuis le fichier KV

```
1  class QuitApp(App):
2      def _quit(self, source):
3          print(source)
4          sys.exit(0)
5
6  QuitApp().run()
```

```
1  #:import App kivy.app.App
2
3  AnchorLayout:
4      anchor_x: 'center'
5      anchor_y: 'center'
6      Button:
7          text: 'Quit'
8          size_hint: (0.7,0.3)
9          on_press: App.get_running_app()._quit(self)
```

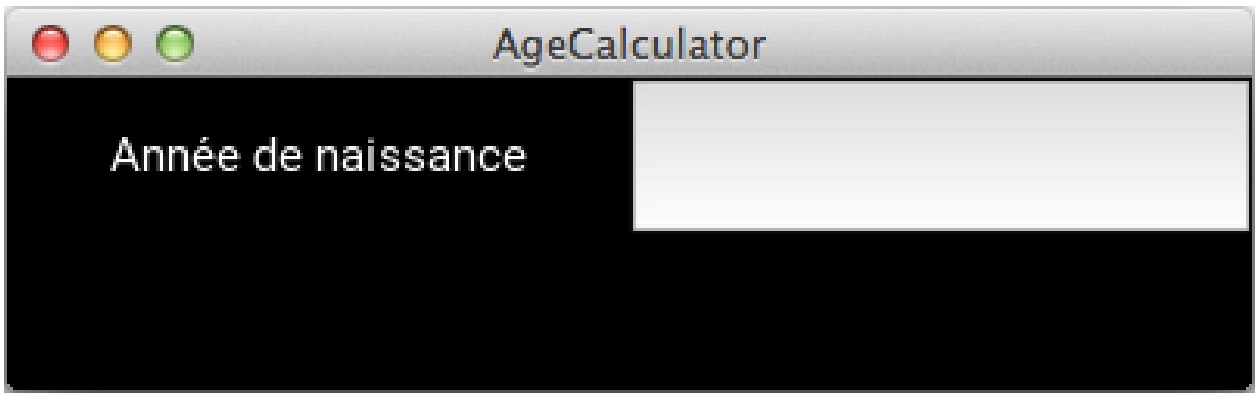
Composant personnalisé (1)

- Création d'un composant personnalisé à partir d'autres

  Très utile pour faire des applications avec un code modulaire

- Il suffit de définir une nouvelle classe, de type `BoxLayout`

  Exemple avec un programme qui permet de calculer l'âge



Composant personnalisé (2)

- Définition de deux classes vides

Contenu du composant personnalisé défini dans le fichier KV

```
class AgeCalculatorForm(BoxLayout):
    pass

class AgeCalculatorApp(App):
    pass
```

```
AgeCalculatorForm:
    orientation: 'vertical'
    BoxLayout:
        orientation: 'horizontal'
        Label:
            text: 'Année de naissance'
        TextInput:
            multiline: False
    Label
```

Définition de propriétés (1)

- Référencer les composants définis dans le fichier KV

  En utilisant des propriétés dans la classe

- Propriétés de type objet pour référencer un widget

  Une pour la zone de texte, une pour le label

```
1  class AgeCalculatorForm(BoxLayout):
2      birthyear_input = ObjectProperty()
3      age_label = ObjectProperty()
```

Définition de propriétés (2)

- Attachement des propriétés dans le fichier KV

  Identification des widgets avec une propriété `id`

```
1  AgeCalculatorForm:
2      birthyear_input: birthyear
3      age_label: age
4      orientation: 'vertical'
5      BoxLayout:
6          orientation: 'horizontal'
7          Label:
8              text: 'Année de naissance'
9          TextInput:
10             id: birthyear
11             multiline: False
12     Label:
13         id: age
```

Calcul de l'âge

- Utilisation des propriétés pour accéder aux widgets

  Lire la valeur de la zone texte, et écrire dans le label

- Gestionnaire d'évènements pour la frappe sur ENTER

  Accès direct au composant personnalisé avec `root`

```
1 on_text_validate: root._compute(self)
```

Dessin (1)

- Un canvas est un propriété dans laquelle on peut dessiner

  Des propriétés permettent de dessiner des objets graphiques

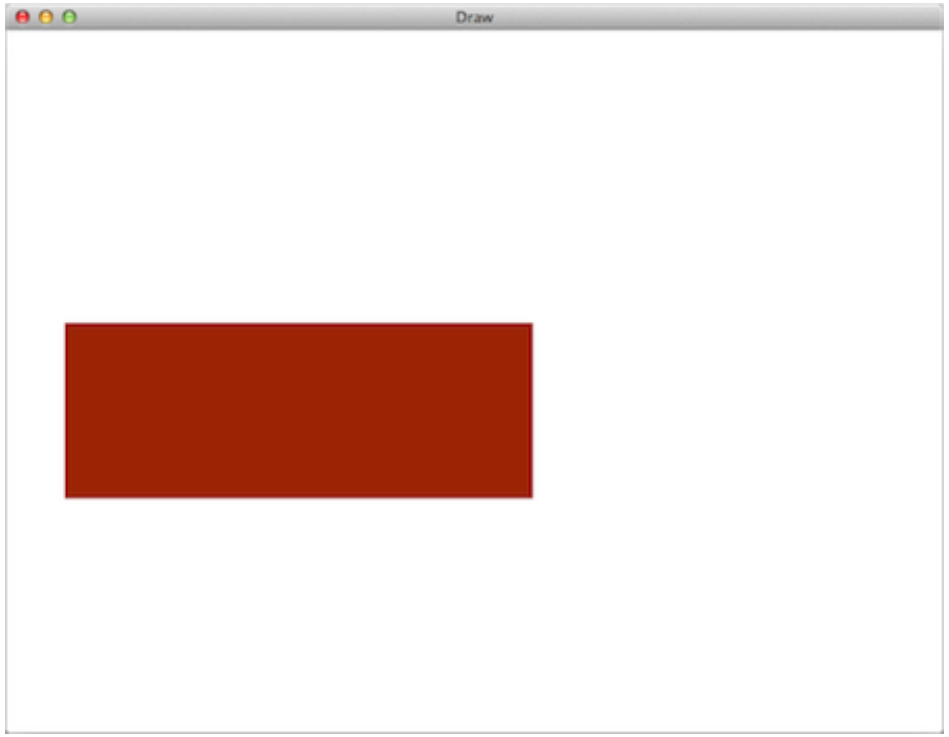
```python
class DrawForm(BoxLayout):
    pass

class DrawApp(App):
    pass
```

```
   DrawForm:
    canvas:
        Color:
            rgb: [1, 1, 1]
        Rectangle:
            pos: (0, 0)
            size: self.size
        Color:
            rgb: [0.7, 0.2, 0]
        Rectangle:
            pos: (50, 200)
            size: (400, 150)
```

Dessin (2)

- Dessin d'un rectangle blanc qui couvre tout et d'un petit rouge

  Des propriétés permettent de dessiner des objets graphiques



Transformation (1)

- Application de transformations aux dessins réalisés

  Rotation, translation et mise à l'échelle

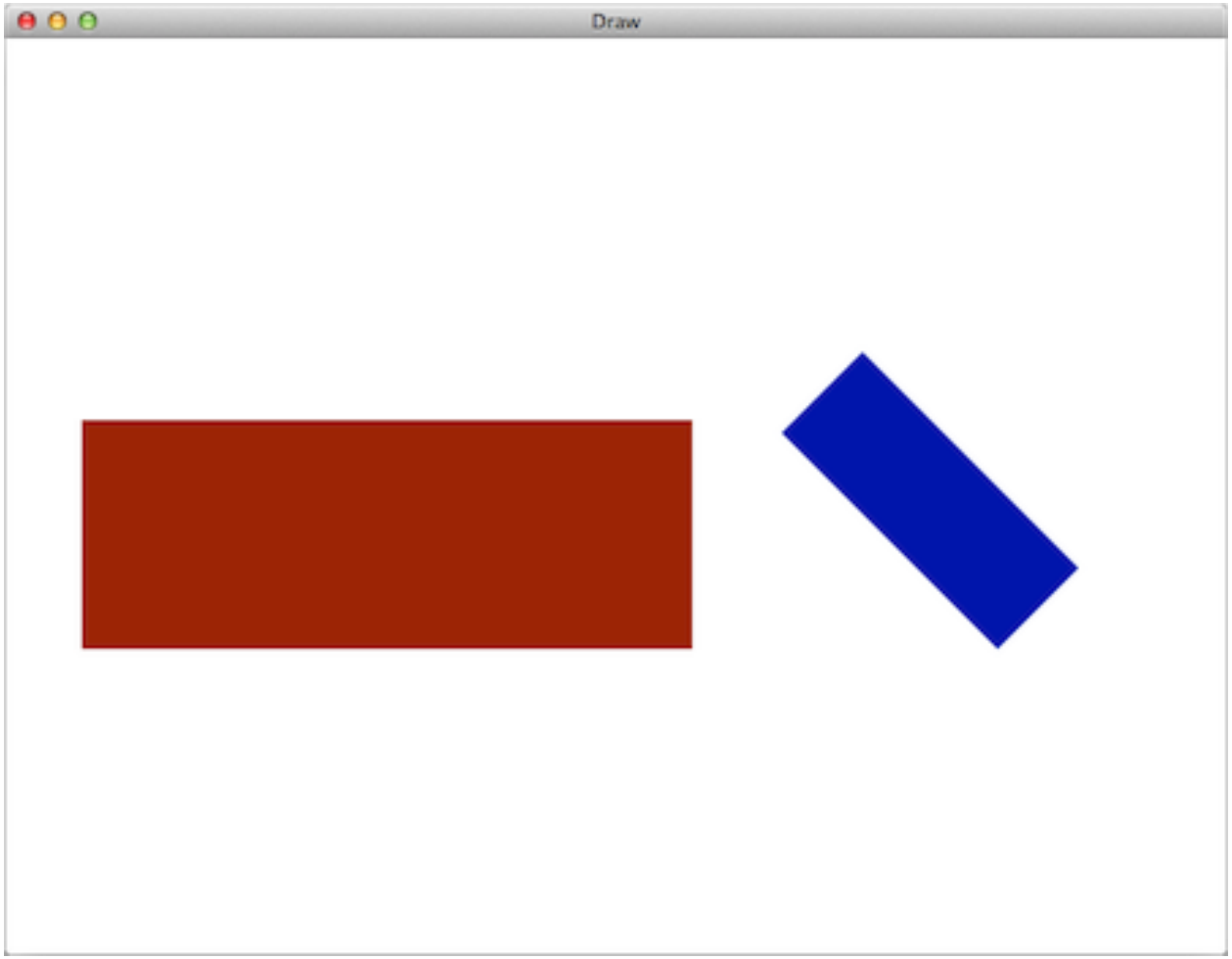
- Une transformation est définie pour tous les dessins futurs

  Si plusieurs, sont appliquées dans l'ordre inverse de l'apparition

Transformation (2)

```
1  DrawForm:
2      canvas:
3          Color:
4              rgb: [1, 1, 1]
5          Rectangle:
6              pos: (0, 0)
7              size: self.size
8          Color:
9              rgb: [0.7, 0.2, 0]
10         Rectangle:
11             pos: (50, 200)
12             size: (400, 150)
13         Translate:
14             x: 200
15             y: 0
16         Rotate:
17             origin: (450, 200)
18             angle: -45
19             axis: (0, 0, 1)
20         Scale:
21             origin: (450, 200)
22             x: 0.5
23             y: 0.5
24         Color:
25             rgb: [0, 0.2, 0.7]
26         Rectangle:
27             pos: (50, 200)
28             size: (400, 150)
```

Transformation (3)



Crédits

- https://www.flickr.com/photos/silvertje/1934375123
- https://www.flickr.com/photos/125720812@N02/15529452622